



# Red Hat OpenStack Platform 16.1

## Federate with Identity Service

Federate with Identity Service using Red Hat Single Sign-On



# Red Hat OpenStack Platform 16.1 Federate with Identity Service

---

Federate with Identity Service using Red Hat Single Sign-On

OpenStack Team  
rhos-docs@redhat.com

## Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

Federate with Identity Service using Red Hat Single Sign-On

# Table of Contents

<b>CHAPTER 1. INTRODUCTION</b>	<b>4</b>
1.1. OVERVIEW	4
1.2. PREREQUISITES	4
1.3. ACCESSING THE RED HAT OPENSTACK PLATFORM NODES	5
1.4. OVERVIEW OF TECHNOLOGIES	5
1.4.1. High availability	5
1.4.1.1. Managing Pacemaker Services	6
1.4.2. HAProxy Overview	6
1.5. USING A CONFIGURATION SCRIPT	6
1.6. USING A PROXY OR SSL TERMINATOR	7
<b>CHAPTER 2. CONFIGURING RED HAT IDENTITY MANAGEMENT</b>	<b>8</b>
2.1. CREATING THE IDM SERVICE ACCOUNT FOR RH-SSO	8
2.2. CREATING A TEST USER	8
2.3. CREATING AN IDM GROUP FOR OPENSTACK USERS	9
<b>CHAPTER 3. CONFIGURING RED HAT SINGLE SIGN-ON</b>	<b>10</b>
3.1. CONFIGURING THE RH-SSO REALM	10
3.2. ADDING USER ATTRIBUTES USING SAML ASSERTION	11
3.3. ADDING GROUP INFORMATION TO THE SAML ASSERTION	12
<b>CHAPTER 4. CONFIGURE OPENSTACK FOR FEDERATION</b>	<b>14</b>
4.1. DETERMINE THE IP ADDRESS AND FQDN SETTINGS	14
4.1.1. Retrieve the IP address	14
4.1.2. Set the Host Variables and Name the Host	16
4.2. INSTALL HELPER FILES ON UNDERCLOUD-0	17
4.3. SET YOUR DEPLOYMENT VARIABLES	17
4.4. COPY THE HELPER FILES FROM UNDERCLOUD-0 TO CONTROLLER-0	17
4.5. INITIALIZE THE WORKING ENVIRONMENT ON THE UNDERCLOUD	17
4.6. INITIALIZE THE WORKING ENVIRONMENT ON CONTROLLER-0	18
4.7. INSTALL MOD_AUTH_MELLON ON EACH CONTROLLER NODE	18
4.8. USE THE KEYSTONE VERSION 3 API	18
4.9. ADD THE RH-SSO FQDN TO EACH CONTROLLER	19
4.10. INSTALL AND CONFIGURE MELLON ON THE CONTROLLER NODE	19
4.11. EDIT THE MELLON CONFIGURATION	20
4.12. CREATE AN ARCHIVE OF THE GENERATED CONFIGURATION FILES	21
4.13. RETRIEVE THE MELLON CONFIGURATION ARCHIVE	21
4.14. PREVENT PUPPET FROM DELETING UNMANAGED HTTPD FILES	21
4.15. CONFIGURE KEYSTONE FOR FEDERATION	22
4.16. DEPLOY THE MELLON CONFIGURATION ARCHIVE	23
4.17. REDEPLOY THE OVERCLOUD	24
4.18. USE PROXY PERSISTENCE FOR KEYSTONE ON EACH CONTROLLER	24
4.19. CREATE FEDERATED RESOURCES	25
4.20. CREATE THE IDENTITY PROVIDER IN OPENSTACK	25
4.21. CREATE THE MAPPING FILE AND UPLOAD TO KEYSTONE	26
4.21.1. Create the mapping	27
4.22. CREATE A KEYSTONE FEDERATION PROTOCOL	27
4.23. FULLY-QUALIFY THE KEYSTONE SETTINGS	27
4.24. CONFIGURE HORIZON TO USE FEDERATION	28
4.25. CONFIGURE HORIZON TO USE THE X-FORWARDED-PROTO HTTP HEADER	28
<b>CHAPTER 5. TROUBLESHOOTING</b>	<b>29</b>

5.1. TEST THE KEYSTONE MAPPING RULES	29
5.2. DETERMINE THE ACTUAL ASSERTION VALUES RECEIVED BY KEYSTONE	30
5.3. REVIEW THE SAML MESSAGES EXCHANGED BETWEEN THE SP AND IDP	30
<b>CHAPTER 6. THE CONFIGURE-FEDERATION FILE</b> .....	<b>32</b>
<b>CHAPTER 7. THE FED_VARIABLES FILE</b> .....	<b>47</b>



# CHAPTER 1. INTRODUCTION

To configure federation in a high availability Red Hat OpenStack Platform director environment, you must configure the following:

- Red Hat Identity Management
- Red Hat single sign-on (RH-SSO)
- The Red Hat OpenStack Platform overcloud

## 1.1. OVERVIEW

Federated authentication is a method of providing authentication across disparate services. This authentication solution relies on an identity provider (IdP), a service provider (SP), and is based on the Security Assertion Markup Language (SAML).

When OpenStack is the service provider in a federated authentication solution, members of the Red Hat Identity Management (IdM) group **openstack-users** are mapped into OpenStack Keystone group **federated\_users** with the **Member** role for project access. Consequently, you are able to grant users access to OpenStack by adding those users to the IdM group **openstack-users**.

## 1.2. PREREQUISITES

You will need the following completed before deploying federated authentication:

- You have deployed Red Hat OpenStack Platform director and the overcloud with the following properties:
  - You can use SSH to connect to both Red Hat OpenStack Platform director, and each of the overcloud nodes.
  - All nodes have a fully qualified domain name (FQDN).
  - TLS encryption is used for all external communications.
  - HAProxy terminates TLS front-end connections, and servers running behind HAProxy do not use TLS.
- An RH-SSO server is present, and you either have administrative privileges on the server, or the RH-SSO administrator has created a realm for you and given you administrative privileges on that realm. Because federated IdPs are external by definition, the RH-SSO server is assumed to be external to the Red Hat OpenStack Platform director overcloud. For more information, see [Installing and configuring RH-SSO](#) and [Creating a realm and user](#).
- An IdM server is present, and also external to the Red Hat OpenStack Platform director overcloud where users and groups are managed. RH-SSO uses IdM as its User Federation backing store.
- You follow the examples described in the [Keystone Federation Configuration Guide](#).
- On the **undercloud-0** node, you install the helper files into the home directory of the **stack** user, and work in the **stack** user home directory.
- On the **controller-0** node, you install the helper files into the home directory of the **heat-admin** user, and work in the **heat-admin** user home directory.



- If **mod\_auth\_mellon** was previously installed on your controller nodes, you must reinstall it as the Puppet Apache class will remove any Apache configuration files not under Puppet's control.



## NOTE

Only the Red Hat OpenStack overcloud has federation enabled. The director is not federated.

## 1.3. ACCESSING THE RED HAT OPENSTACK PLATFORM NODES

By default, you must login to Red Hat OpenStack Platform director to access the overcloud nodes.

1. Use SSH to connect to Red Hat OpenStack director:

```
# ssh undercloud-0
```

2. Become the **stack** user:

```
$ su - stack
```

3. Source the **stackrc** configuration to enable the required OpenStack environment variables:

```
$ source stackrc
```

4. After you source **stackrc**, you can issue commands using the **openstack** command line tool, which operates against Red Hat OpenStack Platform director. To directly access one of the overcloud nodes, retrieve the ip address by using **openstack server list** and then using SSH to connect:

```
(undercloud) [stack@director ~]$ openstack server list -c Name -c Networks
```

```
+-----+-----+
| Name           | Networks           |
+-----+-----+
| rhosp-controller-0 | ctlplane=10.94.101.11 |
| rhosp-controller-1 | ctlplane=10.94.101.14 |
| rhosp-controller-2 | ctlplane=10.94.101.17 |
| rhosp-hypervisor-0 | ctlplane=10.94.101.18 |
| rhosp-hypervisor-1 | ctlplane=10.94.101.20 |
+-----+-----+
```

```
$ ssh heat-admin@10.94.101.11
```

## 1.4. OVERVIEW OF TECHNOLOGIES

The following technologies are a part of Red Hat OpenStack Platform.

### 1.4.1. High availability

Red Hat OpenStack Platform director distributes redundant copies of various OpenStack services across the overcloud deployment. These redundant services are deployed on the overcloud controller nodes, with director naming these nodes **controller-0**, **controller-1**, **controller-2**, and so on, depending on how many controller nodes Red Hat OpenStack Platform director has configured.

The IP addresses of the Controller nodes are not externally visible because the services running on the Controller nodes are HAProxy back-end servers. There is one publicly visible IP address for the set of controller nodes; this is HAProxy's front end. When a request arrives for a service on the public IP address, HAProxy selects a back-end server to service the request.

The overcloud is organized as a high availability cluster. [Pacemaker](#) manages the cluster, performs health checks, and can failover to another cluster resource if the resource stops functioning. You use Pacemaker to start and stop these resources.

For more information about high availability, see the {defaultURL}-single/high\_availability\_deployment\_and\_usage/[High Availability Deployment and Usage] guide.

#### 1.4.1.1. Managing Pacemaker Services

Do not use the **systemctl** command on a Controller node to manage services that Pacemaker manages. Use the Pacemaker **pcs** command:

+

```
sudo pcs resource restart haproxy-clone
```

To determine the resource name, use the Pacemaker **status** command:

+

```
sudo pcs status
```

```
Clone Set: haproxy-clone [haproxy]
Started: [controller-1]
Stopped: [controller-0]
```

#### 1.4.2. HAProxy Overview

HAProxy serves a similar role to Pacemaker. It performs health checks on the back-end servers and forwards requests to functioning back-end servers. There is a copy of HAProxy running on all Controller nodes.

Although there are  $N$  copies of HAProxy running, only one is actually fielding requests at any given time; this active HAProxy instance is managed by Pacemaker. This approach prevents conflicts from occurring, and allows multiple copies of HAProxy to coordinate the distribution of requests across multiple back-ends. If Pacemaker detects that HAProxy has failed, it reassigns the front-end IP address to a different HAProxy instance. This HAProxy instance then becomes the controlling HAProxy instance.

### 1.5. USING A CONFIGURATION SCRIPT

To configure federated authentication, you will need to run long and complex commands. To make that task easier and to allow for repeatability, the commands are saved to a shell script called **configure-federation**. You can execute a specific step if you pass the name of the step to **configure-federation**. To view the list of possible commands, use the **help** option (**-h** or **--help**).



#### NOTE

For more information on the contents of the script, see [Chapter 6, The configure-federation file](#).

To view the commands that are executed after variable substitution, use the following options:

**-n**

This option provides a dry-run mode that writes its operations to stdout without making changes on the system.

**-v**

This option provides a verbose mode that writes its operations to stdout before executing. This is useful for logging.

## 1.6. USING A PROXY OR SSL TERMINATOR

Consider the following key features for environments behind a proxy.

- A back-end server might have a different hostname, listen on different port, or use a different protocol than what a client sees on the front side of the proxy.  
Problems can occur when a server generates a self-referential URL, for example if the server redirects the client to a different URL on the same server. The URL that the server generates must match the public address and port as seen by the client.
- Authentication protocols such as HTTP and HTTPS are sensitive to the host, port, and protocol, because they often need to ensure a request was targeted for a specific server, port and on a secure transport. Proxies can interfere with this information.
  - A proxy transforms a request received on its public front-end before dispatching it to a non-public server in the back-end.
  - Responses from the non-public back-end server sometimes need adjustment so that it appears as if the response came from the public front-end of the proxy.  
There are various approaches to solving this problem. Because SAML is sensitive to host, port, and protocol information, and because you are configuring SAML behind a high availability proxy (HAProxy), you must deal with these issues or your configuration will likely fail.

## CHAPTER 2. CONFIGURING RED HAT IDENTITY MANAGEMENT

You can configure Red Hat OpenStack Platform with federated user management with the following features:

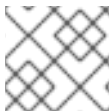
- Red Hat Identity Management (IdM) is external to Red Hat OpenStack Platform
- Red Hat IdM is the source of all user and group information
- Red Hat Single Signon (RH-SSO) is configured to use Red Hat IdM for user Federation

### 2.1. CREATING THE IDM SERVICE ACCOUNT FOR RH-SSO

If you use anonymous binds, some information that is essential for Red Hat Single Sign-On (RH-SSO) is withheld for security reasons. As a result, you need provide the appropriate privileges for RH-SSO in the form of a dedicated account to query the IdM LDAP server for this information:

```
LDAP_URL="ldaps://$FED_IPA_HOST"
DIR_MGR_DN="cn=Directory Manager"
SERVICE_NAME="rhssso"
SERVICE_DN="uid=$service_name,cn=sysaccounts,cn=etc,$FED_IPA_BASE_DN"

$ ldapmodify -H "${LDAP_URL}" -x -D "${DIR_MGR_DN}" -w <_FED_IPA_ADMIN_PASSWD_>
<<EOF
dn: ${SERVICE_DN}
changetype: add
objectclass: account
objectclass: simplesecurityobject
uid: ${SERVICE_NAME}
userPassword: <_FED_IPA_RHSSO_SERVICE_PASSWD_>
passwordExpirationTime: 20380119031407Z
nsIdleTimeout: 0
EOF
```



#### NOTE

You can use the `configure-federation` script to perform the above step:

+

```
$ ./configure-federation create-ipa-service-account
```

### 2.2. CREATING A TEST USER

Create a user account in IdM for testing:

#### Procedure

1. Create a user **jdoe** in IdM:

```
$ipa user-add --first John --last Doe --email jdoe@example.com jdoe
```

2. Assign a password to the user:

```
$ipa passwd jdoe
```

## 2.3. CREATING AN IDM GROUP FOR OPENSTACK USERS

You must have an IdM group **openstack-users** to map to the Keystone group **federated\_users**. Map the test user to this group.

Create the **openstack-users** group in Red Hat Identity Management (IdM):

### Procedure

1. Ensure that the **openstack-users** group does not exist:

```
$ ipa group-show openstack-users
ipa: ERROR: openstack-users: group not found
```

2. Add the openstack-users group to IdM:

```
ipa group-add openstack-users
```

3. Add the test users to the **openstack-users** group:

```
ipa group-add-member --users jdoe openstack-users
```

4. Verify that the **openstack-users** group exists and has the test user as a member:

```
$ ipa group-show openstack-users
Group name: openstack-users
GID: 331400001
Member users: jdoe
```

## CHAPTER 3. CONFIGURING RED HAT SINGLE SIGN-ON

Red Hat Single Sign-On (RH-SSO) supports multi-tenancy, and uses *realms* to allow for separation between tenants. As a result RH-SSO operations always occur within the context of a realm. You can either create the realm manually, or with the **keycloak-httpd-client-install** tool if you have administrative privileges on the RH-SSO server.

### Prerequisites

You must have a fully installed RH-SSO server. For more information on installing RH-SSO, see [Server installation and configuration guide](#).

You need definitions for the following variables as they appear below:

<_RH_RHSSO_URL_>	The Red Hat Single Sign-On URL
<_FED_RHSSO_REALM_>	Identifies the RH-SSO realm in use

### 3.1. CONFIGURING THE RH-SSO REALM

When the Red Hat Single Sign-On (RH-SSO) realm is available, use the RH-SSO web console to configure the realm for user federation against IdM:

#### Procedure

1. From the drop-down list in the upper left corner, select your RH-SSO realm.
2. From the **Configure** panel, select **User Federation**.
3. From the **Add provider** drop-down list in the **User Federation** panel, select **Idap**.
4. Provide values for the following parameters. Substitute all site-specific values with values relevant to your environment.

Property	Value
Console Display Name	Red Hat IDM
Edit Mode	READ_ONLY
Sync Registrations	Off
Vendor	Red Hat Directory Server
Username LDAP attribute	uid
RDN LDAP attribute	uid
UUID LDAP attribute	ipaUniqueID
User Object Classes	inetOrgPerson, organizationalPerson

Property	Value
Connection URL	LDAPS://<_FED_IPA_HOST_>
Users DN	cn=users,cn=accounts,<_FED_IPA_BASE_DN_>
Authentication Type	simple
Bind DN	uid=rhssso,cn=sysaccounts,cn=etc,<_FED_IPA_BASE_DN_>
Bind Credential	<_FED_IPA_RHSSO_SERVICE_PASSWD_>

- Use the Test connection and Test authentication buttons to ensure that user federation is working.
- Click **Save** to save the new user federation provider.
- Click the **Mappers** tab at the top of the Red Hat IdM user federation page you created.
- Create a mapper to retrieve the user group information. A user's group membership returns the SAM assertion. Use group membership later to provide authorization in OpenStack.
- Click **Create** in the Mappers page.
- On the **Add user federation mapper** page, select **group-ldap-mapper** from the *Mapper Type* drop-down list, and name it **Group Mapper**. Provide values for the following parameters. Substitute all site-specific values with values relevant to your environment.

Property	Value
LDAP Groups DN	cn=groups,cn=accounts,,<_FED_IPA_BASE_DN_>
Group Name LDAP Attribute	cn
Group Object Classes	groupOfNames
Membership LDAP Attribute	member
Membership Attribute Type	DN
Mode	READ_ONLY
User Groups Retrieve Strategy	GET_GROUPS_FROM_USER_MEMBEROF_ATTRIBUTE

- Click **Save**.

## 3.2. ADDING USER ATTRIBUTES USING SAML ASSERTION

Security Assertion Markup Language (SAML) is an open standard that allows the communication of user attributes and authorization credentials between the identity provider (IdP) and a service provider (SP).

You can configure Red Hat Single Sign-On (RH-SSO) to return the attributes that you require in the assertion. When the OpenStack Identity service receives the SAML assertion, it maps those attributes onto OpenStack users. The process of mapping IdP attributes into Identity Service data is called Federated Mapping. For more information, see [Section 4.21, "Create the Mapping File and Upload to Keystone"](#).

Use the following process to add attributes to SAML:

#### Procedure

1. In the RH-SSO administration web console, select <\_FED\_RHSSO\_REALM\_> from the drop-down list in the upper left corner.
2. Select **Clients** from the **Configure** panel.
3. Select the service provider client that keycloak-httpd-client-install configured. You can identify the client with the SAML **EntityId**.
4. Select the mappers tab from the horizontal list of tabs.
5. In the Mappers panel, select **Create** or **Add Builtin** to add a protocol mapper to the client.

You can add additional attributes, but you only need the list of groups for which the user is a member. Group membership is how you authorize the user.

### 3.3. ADDING GROUP INFORMATION TO THE SAML ASSERTION

#### Procedure

1. Click the **Create** button in the Mappers Panel.
2. In the **Create Protocol Mapper** panel, select Group list from the Mapper tpe drop-down list.
3. Enter Group List as a name in the **Name** field.
4. Enter groups as the name of the SAML attribute in the Group attribute **Name** field.

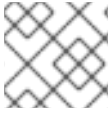


#### NOTE

This is the name of the attribute as it appears in the SAML assertion. When the keystone mapper searches for names in the **Remote** section of the mapping declaration, it searches for the SAML attribute name. When you add an attribute in RH-SSO to be passed in the assertion, specify the SAML attribute name. You define the name in the RH-SSO protocol mapper.

5. In the SAML Attribute NameFormat parameter, select **Basic**.
6. In the Single Group Attribute toggle box, select **On**.
7. Click **Save**.



**NOTE**

When you run the **keycloak-httpd-client-install** tool, the process adds a group mapper.

## CHAPTER 4. CONFIGURE OPENSTACK FOR FEDERATION

### 4.1. DETERMINE THE IP ADDRESS AND FQDN SETTINGS

The following nodes require an assigned Fully-Qualified Domain Name (FQDN):

- The host running the Dashboard (horizon).
- The host running the Identity Service (keystone), referenced in this guide as **\$FED\_KEYSTONE\_HOST**. Note that more than one host will run a service in a high-availability environment, so the IP address is not a host address but rather the IP address bound to the service.
- The host running RH-SSO.
- The host running IdM.

The Red Hat OpenStack Platform director deployment does not configure DNS or assign FQDNs to the nodes, however, the authentication protocols (and TLS) require the use of FQDNs. As a result, you must determine the external public IP address of the overcloud. Note that you need the IP address of the overcloud, which is not the same as the IP address allocated to an individual node in the overcloud, such as *controller-0*, *controller-1*.

You will need the external public IP address of the overcloud because IP addresses are assigned to a high availability cluster, instead of an individual node. Pacemaker and HAProxy work together to provide the appearance of a single IP address; this IP address is entirely distinct from the individual IP address of any given node in the cluster. As a result, the correct way to think about the IP address of an OpenStack service is not in terms of which node that service is running on, but rather to consider the effective IP address that the cluster is advertising for that service (for example, the VIP).

#### 4.1.1. Retrieve the IP address

In order to determine the correct IP address, you will need to assign a name to it, instead of using DNS. There are two ways to do this:

1. Red Hat OpenStack Platform director uses one common public IP address for all OpenStack services, and separates those services on the single public IP address by port number; if you the know public IP address of one service in the OpenStack cluster then you know all of them (however that does not also tell you the port number of a service). You can examine the Keystone URL in the **overcloudrc** file located in the **~stack** home directory on the undercloud. For example:

```
export OS_AUTH_URL=https://10.0.0.101:13000/v2.0
```

This tells you that the public keystone IP address is **10.0.0.101** and that keystone is available on port **13000**. By extension, all other OpenStack services are also available on the **10.0.0.101** IP address with their own unique port number.

2. However, the more accurate way of determining the IP addresses and port number information is to examine the HAProxy configuration file (**/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg**), which is located on each of the overcloud nodes. The **haproxy.cfg** file is an *identical* copy on each of the overcloud controller nodes; this is essential because Pacemaker will assign one controller node the responsibility of running

HAProxy for the cluster, in the event of an HAProxy failure Pacemaker will reassign a different overcloud controller to run HAProxy. No matter which controller node is *currently* running HAProxy, it must act identically; therefore the **haproxy.cfg** files must be identical.

- a. To examine the **haproxy.cfg** file, SSH into one of the cluster's controller nodes and review **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg**. As noted above it does not matter which controller node you select.
- b. The **haproxy.cfg** file is divided into sections, with each beginning with a **listen** statement followed by the name of the service. Immediately inside the service section is a **bind** statement; these are the *front* IP addresses, some of which are public, and others are internal to the cluster. The **server** lines are the *back* IP addresses where the service is actually running, there should be one **server** line for each controller node in the cluster.
- c. To determine the public IP address and port of the service from the multiple **bind** entries in the section:

Red Hat OpenStack Platform director puts the public IP address as the first **bind** entry. In addition, the public IP address should support TLS, so the **bind** entry will have the **ssl** keyword. The IP address should also match the IP address set in the **OS\_AUTH\_URL** located in the **overstackrc** file. For example, here is a sample **keystone\_public** section from a **haproxy.cfg**:

```
listen keystone_public
  bind 10.0.0.101:13000 transparent ssl crt /etc/pki/tls/private/overcloud_endpoint.pem
  bind 172.17.1.19:5000 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option forwardfor
  redirect scheme https code 301 if { hdr(host) -i 10.0.0.101 } !{ ssl_fc }
  rsprep ^Location:\ http://(.*) Location:\ https://\1
  server controller-0.internalapi.localdomain 172.17.1.13:5000 check fall 5 inter 2000 rise
  2 cookie controller-0.internalapi.localdomain
  server controller-1.internalapi.localdomain 172.17.1.22:5000 check fall 5 inter 2000 rise
  2 cookie controller-1.internalapi.localdomain
```

- d. The first **bind** line has the **ssl** keyword, and the IP address matches that of the **OS\_AUTH\_URL** located in the **overstackrc** file. As a result, you can be confident that keystone is publicly accessed at the IP address of **10.0.0.101** on port **13000**.
- e. The second **bind** line is internal to the cluster, and is used by other OpenStack services running in the cluster (note that it does not use TLS because it is not public).
- f. The **mode http** setting indicates that the protocol in use is **HTTP**, this allows HAProxy to examine HTTP headers, among other tasks.
- g. The **X-Forwarded-Proto** lines:

```
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
```

These settings require particular attention and will be covered in more detail in [Section 4.1.2, "Set the Host Variables and Name the Host"](#). They guarantee that the HTTP header **X-Forwarded-Proto** will be set and seen by the back-end server. The back-end server in many cases needs to know if the client was using **HTTPS**. However, HAProxy terminates TLS so the back-end server will see the connection as non-TLS. The **X-**

**Forwarded-Proto** HTTP header is a mechanism that allows the back-end server identify which protocol the client was actually using, instead of which protocol the request arrived on. It is essential that a client can not be able to send a **X-Forwarded-Proto** HTTP header, because that would allow the client to maliciously spoof that the protocol was **HTTPS**. The **X-Forwarded-Proto** HTTP header can either be deleted by the proxy when it is received from the client, or the proxy can forcefully set it and so mitigate any malicious use by the client. This is why **X-Forwarded-Proto** will always be set to one of **https** or **http**.

The X-Forwarded-For HTTP header is used to track the client, which allows the back-end server to identify who the requesting client was instead of it appearing to be the proxy. This option causes the **X-Forwarded-For HTTP** header to be inserted into the request:

```
option forwardfor
```

See [Section 4.1.2, “Set the Host Variables and Name the Host”](#) for more information on forwarded *proto*, redirects, *ServerName*, among others.

- h. The following line will confirm that only HTTPS is used on the public IP address:

```
redirect scheme https code 301 if { hdr(host) -i 10.0.0.101 } !{ ssl_fc }
```

This setting identifies if the request was received on the public IP address (for example **10.0.0.101**) and it was not HTTPS, then performs a **301 redirect** and sets the scheme to HTTPS.

- i. HTTP servers (such as Apache) often generate self-referential URLs for redirect purposes. This redirect location must indicate the correct protocol, but if the server is behind a TLS terminator it will think its redirection URL should be HTTP and not HTTPS. This line identifies if a *Location* header appears in the response that uses the HTTP scheme, then rewrites it to use the HTTPS scheme:

```
rsprep ^Location:\ http://(.*) Location:\ https://\1
```

## 4.1.2. Set the Host Variables and Name the Host

You will need to determine the IP address and port to use. In this example the IP address is **10.0.0.101** and the port is **13000**.

1. This value can be confirmed in **overcloudrc**:

```
export OS_AUTH_URL=https://10.0.0.101:13000/v2.0
```

2. And in the **keystone\_public** section of the **haproxy.cfg** file:

```
bind 10.0.0.101:13000 transparent ssl crt /etc/pki/tls/private/overcloud_endpoint.pem
```

3. You must also give the IP address a FQDN. This example uses **overcloud.localdomain**. Note that the IP address should be put in the **/etc/hosts** file since DNS is not being used:

```
10.0.0.101 overcloud.localdomain # FQDN of the external VIP
```

**NOTE**

Red Hat OpenStack Platform director is expected to have already configured the *hosts* files on the overcloud nodes, but you may need to add the host entry on any external hosts that participate.

4. The **\$FED\_KEYSTONE\_HOST** and **\$FED\_KEYSTONE\_HTTPS\_PORT** must be set in the **fed\_variables** file. Using the above example values:

```
FED_KEYSTONE_HOST="overcloud.localdomain"
FED_KEYSTONE_HTTPS_PORT=13000
```

Because Mellon is running on the Apache server that hosts keystone, the Mellon **host:port** and keystone **host:port** values will match.

**NOTE**

If you run **hostname** on one of the controller nodes it will likely be similar to this: **controller-0.localdomain**, but note that this is its *internal cluster* name, not its public name. You will instead need to use the *public IP address*.

## 4.2. INSTALL HELPER FILES ON UNDERCLOUD-0

1. Copy the **configure-federation** and **fed\_variables** files into the **~stack** home directory on **undercloud-0**. You will have created these files as part of [Section 1.5, "Using a configuration script"](#).

## 4.3. SET YOUR DEPLOYMENT VARIABLES

1. The file **fed\_variables** contains variables specific to your federation deployment. These variables are referenced in this guide as well as in the **configure-federation** helper script. Each site-specific federation variable is prefixed with **FED\_** and (when used as a variable) will use the **\$** variable syntax, such as **\$FED\_**. Make sure every **FED\_** variable in **fed\_variables** is provided a value.

## 4.4. COPY THE HELPER FILES FROM UNDERCLOUD-0 TO CONTROLLER-0

1. Copy the **configure-federation** and the edited **fed\_variables** from the **~stack** home directory on **undercloud-0** to the **~heat-admin** home directory on **controller-0**. For example:

```
$ scp configure-federation fed_variables heat-admin@controller-0:/home/heat-admin
```

**NOTE**

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation copy-helper-to-controller**

## 4.5. INITIALIZE THE WORKING ENVIRONMENT ON THE UNDERCLOUD

1. On the undercloud node, as the *stack* user, create the **fed\_deployment** directory. This location will be the file stash. For example:

```
$ su - stack
$ mkdir fed_deployment
```

**NOTE**

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation initialize**

## 4.6. INITIALIZE THE WORKING ENVIRONMENT ON CONTROLLER-0

1. From the undercloud node, SSH into the **controller-0** node as the **heat-admin** user and create the **fed\_deployment** directory. This location will be the file stash. For example:

```
$ ssh heat-admin@controller-0
$ mkdir fed_deployment
```

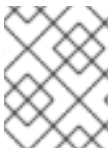
**NOTE**

You can use the **configure-federation** script to perform the above step. From the **controller-0** node: **\$ ./configure-federation initialize**

## 4.7. INSTALL MOD\_AUTH\_MELLON ON EACH CONTROLLER NODE

1. From the undercloud node, SSH into the controller-n node as the **heat-admin** user and install the **mod\_auth\_mellon**. For example:

```
$ ssh heat-admin@controller-n # replace n with controller number
$ sudo dnf reinstall mod_auth_mellon
```

**NOTE**

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation install-mod-auth-mellon**

## 4.8. USE THE KEYSTONE VERSION 3 API

Before you can use the **openstack** command line client to administer the overcloud, you will need to configure certain parameters. Normally this is done by *sourcing* an *rc* file within your shell session, which sets the required environment variables. Red Hat OpenStack Platform director will have created an **overcloudrc** file for this purpose in the home directory of the *stack* user, in the **undercloud-0** node. By default, the **overcloudrc** file is set to use the v2 version of the keystone API, however, federation requires the use of the **v3** keystone API. As a result, you need to create a new *rc* file that uses the **v3** keystone API.

1. For example:

```
$ source overcloudrc
$ NEW_OS_AUTH_URL=`echo $OS_AUTH_URL | sed 's!v2.0!v3!'`
```

2. Write the following contents to **overcloudrc.v3**:

```
for key in $( set | sed 's!=.*!!g' | grep -E '^OS_') ; do unset $key ; done
export OS_AUTH_URL=$NEW_OS_AUTH_URL
export OS_USERNAME=$OS_USERNAME
export OS_PASSWORD=$OS_PASSWORD
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default
export OS_PROJECT_NAME=$OS_TENANT_NAME
export OS_IDENTITY_API_VERSION=3
```



#### NOTE

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation create-v3-rcfile**

3. From this point forward, to work with the overcloud you will use the **overcloudrc.v3** file:

```
$ ssh undercloud-0
$ su - stack
$ source overcloudrc.v3
```

## 4.9. ADD THE RH-SSO FQDN TO EACH CONTROLLER

The mellon service will be running on each controller node and configured to connect to the RH-SSO IdP.

1. If the FQDN of the RH-SSO IdP is not resolvable through DNS then you will have to manually add the FQDN to the **/etc/hosts** file on all controller nodes (after the Heat Hosts section):

```
$ ssh heat-admin@controller-n
$ sudo vi /etc/hosts

# Add this line (substituting the variables) before this line:
# HEAT_HOSTS_START - Do not edit manually within this section!
...
# HEAT_HOSTS_END
$FED_RHSSO_IP_ADDR $FED_RHSSO_FQDN
```

## 4.10. INSTALL AND CONFIGURE MELLON ON THE CONTROLLER NODE

The **keycloak-httpd-client-install** tool performs many of the steps needed to configure **mod\_auth\_mellon** and have it authenticate against the RH-SSO IdP. The **keycloak-httpd-client-install** tool should be run on the node where mellon will run. In our case this means mellon will be running on the overcloud controllers protecting Keystone.

Note that this is a high availability deployment, and as such there will be multiple overcloud controller nodes, each running identical copies. As a result, the mellon setup will need to be replicated across each controller node. You will approach this by installing and configuring mellon on **controller-0**, and then gathering up all the configuration files that the **keycloak-httpd-client-install** tool created into an archive (for example, a tar file) and then let swift copy the archive over to each controller and unarchive the files there.

1. Run the RH-SSO client installation:

```
$ ssh heat-admin@controller-0
$ dnf -y install keycloak-httpd-client-install
$ sudo keycloak-httpd-client-install \
  --client-originate-method registration \
  --mellon-https-port $FED_KEYSTONE_HTTPS_PORT \
  --mellon-hostname $FED_KEYSTONE_HOST \
  --mellon-root /v3 \
  --keycloak-server-url $FED_RHSSO_URL \
  --keycloak-admin-password $FED_RHSSO_ADMIN_PASSWORD \
  --app-name v3 \
  --keycloak-realm $FED_RHSSO_REALM \
  -I "/v3/auth/OS-FEDERATION/websso/mapped" \
  -I "/v3/auth/OS-FEDERATION/identity_providers/rhssso/protocols/mapped/websso" \
  -I "/v3/OS-FEDERATION/identity_providers/rhssso/protocols/mapped/auth"
```



#### NOTE

You can use **configure-federation** script to perform the above step: **\$ ./configure-federation client-install**

2. After the client RPM installation, you should see output similar to this:

```
[Step 1] Connect to Keycloak Server
[Step 2] Create Directories
[Step 3] Set up template environment
[Step 4] Set up Service Provider X509 Certificiates
[Step 5] Build Mellon httpd config file
[Step 6] Build Mellon SP metadata file
[Step 7] Query realms from Keycloak server
[Step 8] Create realm on Keycloak server
[Step 9] Query realm clients from Keycloak server
[Step 10] Get new initial access token
[Step 11] Creating new client using registration service
[Step 12] Enable saml.force.post.binding
[Step 13] Add group attribute mapper to client
[Step 14] Add Redirect URIs to client
[Step 15] Retrieve IdP metadata from Keycloak server
[Step 16] Completed Successfully
```

## 4.11. EDIT THE MELLON CONFIGURATION

Additional mellon configuration is required for your deployment: As you will be using a list of groups during the IdP-assertion-to-Keystone mapping phase, the keystone mapping engine expects lists to be in a certain format (one value with items separated by a semicolon (;)). As a result, you must configure mellon so that when it receives multiple values for an attribute, it must know to combine the multiple attributes into a single value with items separated by a semicolon. This mellon directive will address that:

**MellonMergeEnvVars On ";"**

1. To configure this setting in your deployment:



```
$ vi /var/lib/config-data/puppet-generated/keystone/etc/httpd/conf.d/v3_mellon_keycloak_openstack.conf
```

2. Locate the **<Location /v3>** block and add a line to it. For example:

```
<Location /v3>
...
    MellonMergeEnvVars On ";";
</Location>
```

## 4.12. CREATE AN ARCHIVE OF THE GENERATED CONFIGURATION FILES

The mellon configuration needs to be replicated across all controller nodes, so you will create an archive of the files that allows you to install the exact same file contents on each controller node. The archive will be stored in the **~heat-admin/fed\_deployment** subdirectory.

1. Create the compressed tar archive:

```
$ mkdir fed_deployment
$ tar -cvzf rhssso_config.tar.gz \
  --exclude '*.orig' \
  --exclude '*~' \
  /var/lib/config-data/puppet-generated/keystone/etc/httpd/saml2 \
  /var/lib/config-data/puppet-generated/keystone/etc/httpd/conf.d/v3_mellon_keycloak_openstack.conf
```



### NOTE

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation create-sp-archive**

## 4.13. RETRIEVE THE MELLON CONFIGURATION ARCHIVE

1. On the **undercloud-0** node, fetch the archive you just created and extract the files, as you will need access some of the data in subsequent steps (for example the **entityID** of the RH-SSO IdP).

```
$ scp heat-admin@controller-0:/home/heat-admin/fed_deployment/rhssso_config.tar.gz
~/fed_deployment
$ tar -C fed_deployment -xvf fed_deployment/rhssso_config.tar.gz
```



### NOTE

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation fetch-sp-archive**

## 4.14. PREVENT PUPPET FROM DELETING UNMANAGED HTTPD FILES

By default, the Puppet Apache module will purge any files in the Apache configuration directories it is not managing. This is considered a reasonable precaution, as it prevents Apache from operating in any

manner other than the configuration enforced by Puppet. However, this conflicts with the manual configuration of mellon in the HTTPD configuration directories. When the Apache Puppet **apache::purge\_configs** flag is enabled (which it is by default), Puppet will delete files belonging to the **mod\_auth\_mellon** RPM when the **mod\_auth\_mellon** RPM is installed. It will also delete the configuration files generated by **keycloak-httpd-client-install** when it is run. Until the mellon files are under Puppet control, you will have to disable the **apache::purge\_configs** flag.



#### NOTE

Disabling the **apache::purge\_configs** flag opens the controller nodes to vulnerabilities. Do not forget to re-enable it when Puppet adds support for managing mellon.

To override the **apache::purge\_configs** flag, create a Puppet file containing the override and add the override file to the list of Puppet files used when **overcloud\_deploy.sh** is run.

1. Create the file **fed\_deployment/puppet\_override\_apache.yaml** and add this content:

```
parameter_defaults:
  ControllerExtraConfig:
    apache::purge_configs: false
```

2. Add the file near the end of the **overcloud\_deploy.sh** script. It should be the last **-e** argument. For example:

```
-e /home/stack/fed_deployment/puppet_override_apache.yaml \
--log-file overcloud_deployment_14.log &> overcloud_install.log
```



#### NOTE

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation puppet-override-apache**

## 4.15. CONFIGURE KEYSTONE FOR FEDERATION

This guide uses keystone domains, which require some extra configuration. If enabled, the keystone Puppet module can perform this extra configuration step.

1. In one of the Puppet YAML files, add the following:

```
keystone::using_domain_config: true
```

Some additional values must be set in **/etc/keystone/keystone.conf** to enable federation:

- **auth:methods**
- **federation:trusted\_dashboard**
- **federation:sso\_callback\_template**
- **federation:remote\_id\_attribute**

An explanation of these configuration settings and their suggested values:

- **auth:methods** – A list of allowed authentication methods. By default the list is: **['external', 'password', 'token', 'oauth1']**. You will need to enable SAML using the **mapped** method, so this value should be: **external,password,token,oauth1,mapped**.
- **federation:trusted\_dashboard** – A list of trusted dashboard hosts. Before accepting a Single Sign-On request to return a token, the origin host must be a member of this list. This configuration option may be repeated for multiple values. You must set this in order to use web-based SSO flows. For this deployment the value would be:  
**https://\$FED\_KEYSTONE\_HOST/dashboard/auth/websso/** Note that the host is **\$FED\_KEYSTONE\_HOST** only because Red Hat OpenStack Platform director co-locates both keystone and horizon on the same host. If horizon is running on a different host to keystone, then you will need to adjust accordingly.
- **federation:sso\_callback\_template** – The absolute path to a HTML file used as a Single Sign-On callback handler. This page is expected to redirect the user from keystone back to a trusted dashboard host, by form encoding a token in a **POST** request. Keystone's default value should be sufficient for most deployments: **/etc/keystone/sso\_callback\_template.html**
- **federation:remote\_id\_attribute** – The value used to obtain the entity ID of the Identity Provider. For **mod\_auth\_mellon** you will use **MELLON\_IDP**. Note that this is set in the mellon configuration file using the **MellonIdP IDP** directive.

1. Create the **fed\_deployment/puppet\_override\_keystone.yaml** file with this content:

```
parameter_defaults:
  controllerExtraConfig:
    keystone::using_domain_config: true
    keystone::config::keystone_config:
      identity/domain_configurations_from_database:
        value: true
      auth/methods:
        value: external,password,token,oauth1,mapped
      federation/trusted_dashboard:
        value: https://$FED_KEYSTONE_HOST/dashboard/auth/websso/
      federation/sso_callback_template:
        value: /etc/keystone/sso_callback_template.html
      federation/remote_id_attribute:
        value: MELLON_IDP
```

2. Towards the end of the **overcloud\_deploy.sh** script, add the file you just created. It should be the last **-e** argument. For example:

```
-e /home/stack/fed_deployment/puppet_override_keystone.yaml \
--log-file overcloud_deployment_14.log &> overcloud_install.log
```



#### NOTE

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation puppet-override-keystone**

## 4.16. DEPLOY THE MELLON CONFIGURATION ARCHIVE

You will use swift artifacts to install the mellon configuration files on each controller node. For example:

```
$ source ~/stackrc
$ upload-swift-artifacts -f fed_deployment/rhsso_config.tar.gz
```



#### NOTE

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation deploy-mellon-configuration**

## 4.17. REDEPLOY THE OVERCLOUD

In earlier steps you made changes to the Puppet YAML configuration files and swift artifacts. These changes can now be applied using this command:

```
$ ./overcloud_deploy.sh
```



#### NOTE

In later steps, other configuration changes will be made to the overcloud controller nodes. Re-running Puppet using the **overcloud\_deploy.sh** script may overwrite some of these changes. You should avoid applying the Puppet configuration from this point forward to avoid losing any manual edits that were made to the configuration files on the overcloud controller nodes.

## 4.18. USE PROXY PERSISTENCE FOR KEYSTONE ON EACH CONTROLLER

With high availability, any one of the multiple back-end servers can be expected to field a request. Because of the number of redirections used by SAML, and the fact each of those redirections involves state information, it is vital that the same server processes all the transactions. In addition, a session will be established by **mod\_auth\_mellon**. Currently **mod\_auth\_mellon** is not capable of sharing its state information across multiple servers, so you must configure HAProxy to always direct requests from a client to the same server each time.

HAProxy can bind a client to the same server using either affinity or persistence. This article on [HAProxy Sticky Sessions](#) provides valuable background material.

The difference between *persistence* and *affinity* is that affinity is used when information from a layer below the application layer is used to pin a client request to a single server. Persistence is used when the application layer information binds a client to a single server sticky session. The main advantage of persistence over affinity is that it is much more accurate.

Persistence is implemented through the use of cookies. The HAProxy **cookie** directive names the cookie that will be used for persistence, along with parameters controlling its use. The HAProxy **server** directive has a **cookie** option that sets the value of the cookie, which should be set to the name of the server. If an incoming request does not have a cookie identifying the back-end server, then HAProxy selects a server based on its configured balancing algorithm. HAProxy ensures that the cookie is set to the name of the selected server in the response. If the incoming request has a cookie identifying a back-end server then HAProxy automatically selects that server to handle the request.

1. To enable persistence in the **keystone\_public** block of the **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** configuration file, add this line:

```
cookie SERVERID insert indirect nocache
```

-

This setting states that **SERVERID** will be the name of the persistence cookie.

2. Next, you must edit each **server** line and add **cookie <server-name>** as an additional option. For example:

```
server controller-0 cookie controller-0
server controller-1 cookie controller-1
```

Note that the other parts of the server directive have been omitted for clarity.

## 4.19. CREATE FEDERATED RESOURCES

You might recall from the introduction that you are going to follow the federation example in the [Create keystone groups and assign roles](#) section of the keystone federation documentation.

1. Perform the following steps on the undercloud node as the **stack** user (after sourcing the **overcloudrc.v3** file):

```
$ openstack domain create federated_domain
$ openstack project create --domain federated_domain federated_project
$ openstack group create federated_users --domain federated_domain
$ openstack role add --group federated_users --group-domain federated_domain --domain
federated_domain _member_
$ openstack role add --group federated_users --group-domain federated_domain --project
federated_project _member_
```



### NOTE

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation create-federated-resources**

## 4.20. CREATE THE IDENTITY PROVIDER IN OPENSTACK

The IdP needs to be registered in keystone, which creates a binding between the **entityID** in the SAML assertion and the name of the IdP in keystone.

You will need to locate the **entityID** of the RH-SSO IdP. This value is located in the IdP metadata which was obtained when **keycloak-httpd-client-install** was run. The IdP metadata is stored in the **/var/lib/config-data/puppet-generated/keystone/etc/httpd/saml2/v3\_keycloak\_\${FED\_RHSSO\_REALM}\_idp\_metadata.xml** file. In an earlier step you retrieved the mellon configuration archive and extracted it to the **fed\_deployment** work area. As a result, you can find the IdP metadata in **fed\_deployment/var/lib/config-data/puppet-generated/keystone/etc/httpd/saml2/v3\_keycloak\_\${FED\_RHSSO\_REALM}\_idp\_metadata.xml**. In the IdP metadata file, you will find a **<EntityDescriptor>** element with a **entityID** attribute. You need the value of the **entityID** attribute, and for example purposes this guide will assume it has been stored in the **\$FED\_IDP\_ENTITY\_ID** variable. You can name your IdP **rhssso**, which is assigned to the variable **\$FED\_OPENSTACK\_IDP\_NAME**. For example:

```
$ openstack identity provider create --remote-id $FED_IDP_ENTITY_ID
$FED_OPENSTACK_IDP_NAME
```

**NOTE**

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation openstack-create-idp**

## 4.21. CREATE THE MAPPING FILE AND UPLOAD TO KEYSTONE

Keystone performs a mapping to match the IdP's SAML assertion into a format that keystone can understand. The mapping is performed by keystone's mapping engine and is based on a set of mapping rules that are bound to the IdP.

1. These are the mapping rules used in this example (as described in the introduction):

```
[
  {
    "local": [
      {
        "user": {
          "name": "{0}"
        },
        "group": {
          "domain": {
            "name": "federated_domain"
          },
          "name": "federated_users"
        }
      }
    ],
    "remote": [
      {
        "type": "MELLON_NAME_ID"
      },
      {
        "type": "MELLON_groups",
        "any_one_of": ["openstack-users"]
      }
    ]
  }
]
```

This mapping file contains only one rule. Rules are divided into two parts: **local** and **remote**. The mapping engine works by iterating over the list of rules until one matches, and then executing it. A rule is considered a match only if *all* the conditions in the **remote** part of the rule match. In this example the **remote** conditions specify:

1. The assertion must contain a value called **MELLON\_NAME\_ID**.
2. The assertion must contain a values called **MELLON\_groups** and at least one of the groups in the group list must be **openstack-users**.

If the rule matches, then:

1. The keystone **user** name will be assigned the value from **MELLON\_NAME\_ID**.
2. The user will be assigned to the keystone group **federated\_users** in the **Default** domain.

In summary, if the IdP successfully authenticates the user, and the IdP asserts that user belongs to the group **openstack-users**, then keystone will allow that user to access OpenStack with the privileges bound to the **federated\_users** group in keystone.

### 4.21.1. Create the mapping

1. To create the mapping in keystone, create a file containing the mapping rules and then upload it into keystone, giving it a reference name. Create the mapping file in the **fed\_deployment** directory (for example, in **fed\_deployment/mapping\_\${FED\_OPENSTACK\_IDP\_NAME}\_saml2.json**), and assign the name **\$FED\_OPENSTACK\_MAPPING\_NAME** to the mapping rules. For example:

```
$ openstack mapping create --rules fed_deployment/mapping_rhsso_saml2.json
$FED_OPENSTACK_MAPPING_NAME
```

#### NOTE

You can use the **configure-federation** script to perform the above procedure as two steps:

```
$ ./configure-federation create-mapping
$ ./configure-federation openstack-create-mapping
```

- **create-mapping** - creates the mapping file.
- **openstack-create-mapping** - performs the upload of the file.

## 4.22. CREATE A KEYSTONE FEDERATION PROTOCOL

1. Keystone uses the **Mapped** protocol to bind an IdP to a mapping. To establish this binding:

```
$ openstack federation protocol create \
--identity-provider $FED_OPENSTACK_IDP_NAME \
--mapping $FED_OPENSTACK_MAPPING_NAME \
mapped"
```

#### NOTE

You can use the **configure-federation** script to perform the above step: **\$ ./configure-federation openstack-create-protocol**

## 4.23. FULLY-QUALIFY THE KEYSTONE SETTINGS

1. On each controller node, edit **/var/lib/config-data/puppet-generated/keystone/etc/httpd/conf.d/10-keystone\_wsgi\_main.conf** to confirm that the **ServerName** directive inside the **VirtualHost** block includes the HTTPS scheme, the public hostname, and the public port. You must also enable the **UseCanonicalName** directive. For example:

```
<VirtualHost>
ServerName https:$FED_KEYSTONE_HOST:$FED_KEYSTONE_HTTPS_PORT
UseCanonicalName On
```

```
...
</VirtualHost>
```

**NOTE**

Be sure to substitute the **\$FED\_** variables with the values specific to your deployment.

## 4.24. CONFIGURE HORIZON TO USE FEDERATION

1. On each controller node, edit **/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/local\_settings** and make sure the following configuration values are set:

```
OPENSTACK_KEYSTONE_URL =
"https://$FED_KEYSTONE_HOST:$FED_KEYSTONE_HTTPS_PORT/v3"
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "_member_"
WEBSSO_ENABLED = True
WEBSSO_INITIAL_CHOICE = "mapped"
WEBSSO_CHOICES = (
    ("mapped", _("RH-SSO")),
    ("credentials", _("Keystone Credentials")),
)
```

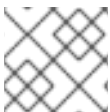
**NOTE**

Be sure to substitute the **\$FED\_** variables with the values specific to your deployment.

## 4.25. CONFIGURE HORIZON TO USE THE X-FORWARDED-PROTO HTTP HEADER

1. On each controller node, edit **/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/local\_settings** and uncomment the line:

```
#SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
```

**NOTE**

You must restart a container for configuration changes to take effect.



## CHAPTER 5. TROUBLESHOOTING

### 5.1. TEST THE KEYSTONE MAPPING RULES

It is recommended you verify that your mapping rules work as expected. The **keystone-manage** command line tool allows you to exercise a set of mapping rules (read from a file) against assertion data which is also read from a file. For example:

1. The file **mapping\_rules.json** has this content:

```
[
  {
    "local": [
      {
        "user": {
          "name": "{0}"
        },
        "group": {
          "domain": {
            "name": "Default"
          },
          "name": "federated_users"
        }
      }
    ],
    "remote": [
      {
        "type": "MELLON_NAME_ID"
      },
      {
        "type": "MELLON_groups",
        "any_one_of": ["openstack-users"]
      }
    ]
  }
]
```

2. The file **assertion\_data.txt** has this content:

```
MELLON_NAME_ID: 'G-90eb44bc-06dc-4a90-aa6e-fb2aa5d5b0de'
MELLON_groups: openstack-users;ipausers
```

3. If you then run this command:

```
$ keystone-manage mapping_engine --rules mapping_rules.json --input assertion_data.txt
```

4. You should get this mapped result:

```
{
  "group_ids": [],
  "user": {
    "domain": {
      "id": "Federated"
    },
  },
```

```

    "type": "ephemeral",
    "name": "G-90eb44bc-06dc-4a90-aa6e-fb2aa5d5b0de"
  },
  "group_names": [
    {
      "domain": {
        "name": "Default"
      },
      "name": "federated_users"
    }
  ]
}

```



## NOTE

You can also include the **--engine-debug** command line argument, which will output diagnostic information describing how the mapping rules are being evaluated.

## 5.2. DETERMINE THE ACTUAL ASSERTION VALUES RECEIVED BY KEYSTONE

The *mapped* assertion values that keystone will use are passed as CGI environment variables. To retrieve a dump of those environment variables:

1. Create the following test script in **/var/www/cgi-bin/keystone/test** with the following content:

```

import pprint
import webob
import webob.dec

@webob.dec.wsgify
def application(req):
    return webob.Response(pprint.pformat(req.environ),
                           content_type='application/json')

```

2. Edit the **/var/lib/config-data/puppet-generated/keystone/etc/httpd/conf.d/10-keystone\_wsgi\_main.conf** file setting it to run the **test** script by temporarily modifying the **WSGIScriptAlias** directive:

```

WSGIScriptAlias "/v3/auth/OS-FEDERATION/websso/mapped" "/var/www/cgi-
bin/keystone/test"

```

3. Restart *httpd*:

```
systemctl restart httpd
```

4. Attempt to login, and review the information that the script dumps out. When finished, remember to restore the **WSGIScriptAlias** directive, and restart the HTTPD service again.

## 5.3. REVIEW THE SAML MESSAGES EXCHANGED BETWEEN THE SP AND IDP

The **SAMLTracer** Firefox add-on is a useful tool for capturing and displaying the SAML messages exchanged between the SP and the IdP.

1. Install **SAMLTracer** from this URL: <https://addons.mozilla.org/en-US/firefox/addon/saml-tracer/>
2. Enable **SAMLTracer** from the Firefox menu. A **SAMLTracer** pop-up window will appear in which all browser requests are displayed. If a request is detected as a SAML message a special **SAML** icon is added to the request.
3. Initiate a SSO login from the Firefox browser.
4. In the **SAMLTracer** window find the first **SAML** message and click on it. Use the **SAML** tab in the window to see the decoded SAML message (note, the tool is not capable of decrypting encrypted content in the body of the message, if you need to see encrypted content you must disable encryption in the metadata). The first SAML message should be an **AuthnRequest** sent by the SP to the IdP. The second SAML message should be the assertion response sent by the IdP. Since the SAML HTTP-Redirect profile is being used the Assertion response will be wrapped in a POST. Click on the **SAML** tab to see the contents of the assertion.

## CHAPTER 6. THE CONFIGURE-FEDERATION FILE

```
#!/bin/sh

prog_name=`basename $0`
action=
dry_run=0
verbose=0

base_dir=$(pwd)
stage_dir="${base_dir}/fed_deployment"

mellon_root="/v3"
mellon_endpoint="mellon"
mellon_app_name="v3"

overcloud_deploy_script="overcloud_deploy.sh"
overcloudrc_file="./overcloudrc"

function cmd_template {
    local status=0
    local cmd="$1"
    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi
    return $status
}

function cmds_template {
    local return_status=0
    declare -a cmds=(
        "date"
        "ls xxx"
        "head $0"
    )

    if [ $dry_run -ne 0 ]; then
        for cmd in "${cmds[@]}"; do
            echo $cmd
        done
    else
        for cmd in "${cmds[@]}"; do
            if [ $verbose -ne 0 ]; then
                echo $cmd
            fi
        done
    fi
}
```

```

    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
        return_status=$status
    fi
done
fi
return $return_status
}

function show_variables {
    echo "base_dir: $base_dir"
    echo "stage_dir: $stage_dir"
    echo "config_tar_filename: $config_tar_filename"
    echo "config_tar_pathname: $config_tar_pathname"
    echo "overcloud_deploy_script: $overcloud_deploy_script"
    echo "overcloudrc_file: $overcloudrc_file"

    echo "puppet_override_apache_pathname: $puppet_override_apache_pathname"
    echo "puppet_override_keystone_pathname: $puppet_override_keystone_pathname"

    echo

    echo "FED_RHSSO_URL: $FED_RHSSO_URL"
    echo "FED_RHSSO_ADMIN_PASSWORD: $FED_RHSSO_ADMIN_PASSWORD"
    echo "FED_RHSSO_REALM: $FED_RHSSO_REALM"

    echo

    echo "FED_KEYSTONE_HOST: $FED_KEYSTONE_HOST"
    echo "FED_KEYSTONE_HTTPS_PORT: $FED_KEYSTONE_HTTPS_PORT"
    echo "mellon_http_url: $mellon_http_url"
    echo "mellon_root: $mellon_root"
    echo "mellon_endpoint: $mellon_endpoint"
    echo "mellon_app_name: $mellon_app_name"
    echo "mellon_endpoint_path: $mellon_endpoint_path"
    echo "mellon_entity_id: $mellon_entity_id"

    echo

    echo "FED_OPENSTACK_IDP_NAME: $FED_OPENSTACK_IDP_NAME"
    echo "openstack_mapping_pathname: $openstack_mapping_pathname"
    echo "FED_OPENSTACK_MAPPING_NAME: $FED_OPENSTACK_MAPPING_NAME"

    echo

    echo "idp_metadata_filename: $idp_metadata_filename"
    echo "mellon_httpd_config_filename: $mellon_httpd_config_filename"
}

function initialize {
    local return_status=0
    declare -a cmds=(
        "mkdir -p $stage_dir"
    )

```

```

    if [ $dry_run -ne 0 ]; then
        for cmd in "${cmds[@}"; do
            echo $cmd
        done
    else
        for cmd in "${cmds[@}"; do
            if [ $verbose -ne 0 ]; then
                echo $cmd
            fi
            $cmd
            status=$?
            if [ $status -ne 0 ]; then
                (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
                return_status=$status
            fi
        done
    fi
    return $return_status
}

function copy_helper_to_controller {
    local status=0
    local controller=${1:-"controller-0"}
    local cmd="scp configure-federation fed_variables heat-admin@${controller}:/home/heat-admin"
    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi
    return $status
}

function install_mod_auth_mellon {
    local status=0
    local cmd="sudo dnf -y install mod_auth_mellon"

    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi
}

```

```

    return $status
}

function create_ipa_service_account {
    # Note, after setting up the service account it can be tested
    # by performing a user search like this:
    # ldapsearch -H $ldap_url -x -D "$service_dn" -w "$FED_IPA_RHSSO_SERVICE_PASSWD" -b
    "cn=users,cn=accounts,$FED_IPA_BASE_DN"

    local status=0
    local ldap_url="ldaps://$FED_IPA_HOST"
    local dir_mgr_dn="cn=Directory Manager"
    local service_name="rhssso"
    local service_dn="uid=$service_name,cn=sysaccounts,cn=etc,$FED_IPA_BASE_DN"
    local cmd="ldapmodify -H \"$ldap_url\" -x -D \"$dir_mgr_dn\" -w \"$FED_IPA_ADMIN_PASSWD\""

    read -r -d " " contents <<EOF
dn: $service_dn
changetype: add
objectclass: account
objectclass: simplesecurityobject
uid: $service_name
userPassword: $FED_IPA_RHSSO_SERVICE_PASSWD
passwordExpirationTime: 20380119031407Z
nsIdleTimeout: 0

EOF

    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
        echo -e "$contents"
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    sh <<< "$cmd <<< \"$contents\""
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi

    return $status
}

function client_install {
    local status=0
    local cmd_client_install="sudo dnf -y install keycloak-httpd-client-install"
    local cmd="sudo keycloak-httpd-client-install \
--client-originate-method registration \
--mellon-https-port $FED_KEYSTONE_HTTPS_PORT \
--mellon-hostname $FED_KEYSTONE_HOST \
--mellon-root $mellon_root \
--keycloak-server-url $FED_RHSSO_URL \

```

```

--keycloak-admin-password $FED_RHSSO_ADMIN_PASSWORD \
--app-name $mellon_app_name \
--keycloak-realm $FED_RHSSO_REALM \
-l "/v3/auth/OS-FEDERATION/websso/mapped" \
-l "/v3/auth/OS-FEDERATION/identity_providers/rhssso/protocols/mapped/websso" \
-l "/v3/OS-FEDERATION/identity_providers/rhssso/protocols/mapped/auth"
"

if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
    echo $cmd_client_install
    echo $cmd
fi
if [ $dry_run -ne 0 ]; then
    return $status
fi

$cmd_client_install
status=$?
if [ $status -ne 0 ]; then
    (>&2 echo -e "ERROR cmd \"$cmd_client_install\" failed\nstatus = $status")
else
    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi
fi
return $status
}

function create_sp_archive {
    # Note, we put the exclude patterns in a file because it is
    # insanely difficult to put --exclude pattern in the $cmd shell
    # variable and get the final quoting correct.

    local status=0
    local cmd="tar -cvzf $config_tar_pathname --exclude-from $stage_dir/tar_excludes /var/lib/config-
data/puppet-generated/keystone/etc/httpd/saml2 /var/lib/config-data/puppet-
generated/keystone/etc/httpd/conf.d/$mellon_httpd_config_filename"
    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    cat <<'EOF' > $stage_dir/tar_excludes
*.orig
*~
EOF

    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi
    return $status
}

```



```

}

function fetch_sp_archive {
    local return_status=0
    declare -a cmds=(
        "scp heat-admin@controller-0:/home/heat-admin/fed_deployment/$config_tar_filename
$stage_dir"
        "tar -C $stage_dir -xvf $config_tar_pathname"
    )

    if [ $dry_run -ne 0 ]; then
        for cmd in "${cmds[@]"; do
            echo $cmd
        done
    else
        for cmd in "${cmds[@]"; do
            if [ $verbose -ne 0 ]; then
                echo $cmd
            fi
            $cmd
            status=$?
            if [ $status -ne 0 ]; then
                (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
                return_status=$status
            fi
        done
    fi
    return $return_status
}

function deploy_mellon_configuration {
    local status=0
    local cmd="upload-swift-artifacts -f $config_tar_pathname"
    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi
    return $status
}

function idp_entity_id {
    local metadata_file=${1:-$idp_metadata_filename}

    # Extract the entitiID from the metadata file, should really be parsed
    # with an XML xpath but a simple string match is probably OK

    entity_id=`sed -rne 's/^.entityID="([^\"]*)".*$/\1/p' ${metadata_file}`
    status=$?

```

```

if [ $status -ne 0 -o "$entity_id"x = "x" ]; then
    (>&2 echo -e "ERROR search for entityID in ${metadata_file} failed\nstatus = $status")
    return 1
fi
echo $entity_id
return 0
}

function append_deploy_script {
    local status=0
    local deploy_script=$1
    local extra_line=$2
    local count

    count=$(grep -c -e "$extra_line" $deploy_script)
    if [ $count -eq 1 ]; then
        echo -e "SKIP appending:\n$extra_line"
        echo "already present in $deploy_script"
        return $status
    elif [ $count -gt 1 ]; then
        status=1
        (>&2 echo -e "ERROR multiple copies of line in ${deploy_script}\nstatus =
$status\nline=$extra_line")
        return $status
    fi

    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo "appending $deploy_script with:"
        echo -e $extra_line
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    # insert line after last -e line already in script
    #
    # This is not easy with sed, we'll use tac and awk instead. Here
    # is how this works: The logic is easier if you insert before the
    # first line rather than trying to find the last line and insert
    # after it. We use tac to reverse the lines in the file. Then the
    # awk script looks for the candidate line. If found it outputs the
    # line we're adding, sets a flag (p) to indicate it's already been
    # printed. The "; 1" pattern always output the input line. Then we
    # run the output through tac again to set things back in the
    # original order.

    local tmp_file=$(mktemp)

    tac $deploy_script | awk '!p && /^-e/{print "\"${extra_line}\" \\\\\""; p=1}; 1' | tac > $tmp_file

    count=$(grep -c -e "${extra_line}" $tmp_file)
    if [ $count -ne 1 ]; then
        status=1
    fi
    if [ $status -ne 0 ]; then
        rm $tmp_file
    fi
}

```

```

        (>&2 echo -e "ERROR failed to append ${deploy_script}\nstatus = $status\nline=$extra_line")
    else
        mv $tmp_file $deploy_script
    fi

    return $status
}

function puppet_override_apache {
    local status=0
    local pathname=${1:-$puppet_override_apache_pathname}
    local deploy_cmd="-e $pathname"

    read -r -d " contents <<'EOF'
parameter_defaults:
  ControllerExtraConfig:
    apache::purge_configs: false
EOF

    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo "writing pathname = $pathname with contents"
        echo -e "$contents"
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    echo -e "$contents" > $pathname
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR failed to write ${pathname}\nstatus = $status")
    fi

    append_deploy_script $overcloud_deploy_script "$deploy_cmd"
    status=$?

    return $status
}

function puppet_override_keystone {
    local status=0
    local pathname=${1:-$puppet_override_keystone_pathname}
    local deploy_cmd="-e $pathname"

    read -r -d " contents <<EOF
parameter_defaults:
  controllerExtraConfig:
    keystone::using_domain_config: true
    keystone::config::keystone_config:
      identity/domain_configurations_from_database:
        value: true
    auth/methods:
      value: external,password,token,oauth1,mapped
    federation/trusted_dashboard:
      value: https://$FED_KEYSTONE_HOST/dashboard/auth/websso/

```

```
federation/sso_callback_template:
  value: /etc/keystone/sso_callback_template.html
federation/remote_id_attribute:
  value: MELLON_IDP
```

EOF

```
if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
    echo "writing pathname = $pathname with contents"
    echo -e "$contents"
fi
if [ $dry_run -ne 0 ]; then
    return $status
fi

echo -e "$contents" > $pathname
status=$?
if [ $status -ne 0 ]; then
    (>&2 echo -e "ERROR failed to write ${pathname}\nstatus = $status")
fi

append_deploy_script $overcloud_deploy_script "$deploy_cmd"
status=$?

return $status
}

function create_federated_resources {
    # follow example in Keystone federation documentation
    # http://docs.openstack.org/developer/keystone/federation/federated_identity.html#create-
    keystone-groups-and-assign-roles
    local return_status=0
    declare -a cmds=(
        "openstack domain create federated_domain"
        "openstack project create --domain federated_domain federated_project"
        "openstack group create federated_users --domain federated_domain"
        "openstack role add --group federated_users --group-domain federated_domain --domain
federated_domain _member_"
        "openstack role add --group federated_users --project federated_project Member"
    )

    if [ $dry_run -ne 0 ]; then
        for cmd in "${cmds[@]"; do
            echo $cmd
        done
    else
        for cmd in "${cmds[@]"; do
            if [ $verbose -ne 0 ]; then
                echo $cmd
            fi
            $cmd
            status=$?
            if [ $status -ne 0 ]; then
                (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
                return_status=$status
            fi
        done
    fi
}
```

```

done
fi
return $return_status
}

function create_mapping {
    # Matches documentation
    # http://docs.openstack.org/developer/keystone/federation/federated_identity.html#create-
    keystone-groups-and-assign-roles
    local status=0
    local pathname=${1:-$openstack_mapping_pathname}

    read -r -d " contents <<'EOF'
[
{
    "local": [
        {
            "user": {
                "name": "{0}"
            },
            "group": {
                "domain": {
                    "name": "federated_domain"
                },
                "name": "federated_users"
            }
        }
    ],
    "remote": [
        {
            "type": "MELLON_NAME_ID"
        },
        {
            "type": "MELLON_groups",
            "any_one_of": ["openstack-users"]
        }
    ]
}
]
}
EOF

if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
    echo "writing pathname = $pathname with contents"
    echo -e "$contents"
fi
if [ $dry_run -ne 0 ]; then
    return $status
fi

echo -e "$contents" > $pathname
status=$?
if [ $status -ne 0 ]; then
    (>&2 echo -e "ERROR failed to write ${pathname}\nstatus = $status")
fi

```

```

    return $status
}

function create_v3_rcfile {
    local status=0
    local input_file=${1:-$overclouddrc_file}
    local output_file="${input_file}.v3"

    source $input_file
    #clear the old environment
    NEW_OS_AUTH_URL=`echo $OS_AUTH_URL | sed 's!v2.0!v3!'`

    read -r -d " contents <<EOF
for key in `$( set | sed 's! =. *!!g' | grep -E '^OS_')` ; do unset $key ; done
export OS_AUTH_URL=$NEW_OS_AUTH_URL
export OS_USERNAME=$OS_USERNAME
export OS_PASSWORD=$OS_PASSWORD
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default
export OS_PROJECT_NAME=$OS_TENANT_NAME
export OS_IDENTITY_API_VERSION=3
EOF

    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo "writeing output_file = $output_file with contents:"
        echo -e "$contents"
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    echo -e "$contents" > $output_file
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR failed to write ${output_file}\nstatus = $status")
    fi

    return $status
}

function openstack_create_idp {
    local status=0
    local metadata_file="$stage_dir/var/lib/config-data/puppet-generated/keystone/etc/httpd/saml2/$idp_metadata_filename"
    local entity_id
    entity_id=$(idp_entity_id $metadata_file)
    status=$?
    if [ $status -ne 0 ]; then
        return $status
    fi

    local cmd="openstack identity provider create --remote-id $entity_id
$FED_OPENSTACK_IDP_NAME"

    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
    fi
}

```

```

fi
if [ $dry_run -ne 0 ]; then
    return $status
fi

$cmd
status=$?
if [ $status -ne 0 ]; then
    (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
fi
return $status
}

function openstack_create_mapping {
    local status=0
    local mapping_file=${1:-$openstack_mapping_pathname}
    local mapping_name=${2:-$FED_OPENSTACK_MAPPING_NAME}
    cmd="openstack mapping create --rules $mapping_file $mapping_name"

    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi
    return $status
}

function openstack_create_protocol {
    local status=0
    local idp_name=${1:-$FED_OPENSTACK_IDP_NAME}
    local mapping_name=${2:-$FED_OPENSTACK_MAPPING_NAME}
    cmd="openstack federation protocol create --identity-provider $idp_name --mapping
    $mapping_name mapped"

    if [ $verbose -ne 0 -o $dry_run -ne 0 ]; then
        echo $cmd
    fi
    if [ $dry_run -ne 0 ]; then
        return $status
    fi

    $cmd
    status=$?
    if [ $status -ne 0 ]; then
        (>&2 echo -e "ERROR cmd \"$cmd\" failed\nstatus = $status")
    fi
    return $status
}

```

```

function usage {
cat <<EOF
$prog_name action

-h --help      print usage
-n --dry-run   dry run, just print computed command
-v --verbose   be chatty

action may be one of:

show-variables
initialize
copy-helper-to-controller
install-mod-auth-mellon
create-ipa-service-account
client-install
create-sp-archive
fetch-sp-archive
deploy-mellon-configuration
puppet-override-apache
puppet-override-keystone
create-federated-resources
create-mapping
create-v3-rcfile
openstack-create-idp
openstack-create-mapping
openstack-create-protocol

EOF
}

#-----
# options may be followed by one colon to indicate they have a required argument
if ! options=$(getopt -o hnv -l help,dry-run,verbose -- "$@")
then
    # something went wrong, getopt will put out an error message for us
    exit 1
fi

eval set -- "$options"

while [ $# -gt 0 ]
do
    case $1 in
        -h|--help) usage; exit 1 ;;
        -n|--dry-run) dry_run=1 ;;
        -v|--verbose) verbose=1 ;;
        # for options with required arguments, an additional shift is required
        --) shift; break;;
        (-*) echo "$0: error - unrecognized option $1" 1>&2; exit 1;;
        (*) break;;
    esac
    shift
done
#-----
source ./fed_variables

```



```

# Strip leading and trailing space and slash from these variables
mellon_root=`echo ${mellon_root} | perl -pe 's!^[ /]*(.*)[ /]*$!\1!'`
mellon_endpoint=`echo ${mellon_endpoint} | perl -pe 's!^[ /]*(.*)[ /]*$!\1!'`

mellon_root="/${mellon_root}"

mellon_endpoint_path="${mellon_root}/${mellon_endpoint}"
mellon_http_url="https://${FED_KEYSTONE_HOST}:${FED_KEYSTONE_HTTPS_PORT}"
mellon_entity_id="${mellon_http_url}${mellon_endpoint_path}/metadata"

openstack_mapping_pathname="${stage_dir}/mapping_${FED_OPENSTACK_IDP_NAME}_saml2.json"
idp_metadata_filename="${mellon_app_name}_keycloak_${FED_RHSSO_REALM}_idp_metadata.xml"

mellon_httpd_config_filename="${mellon_app_name}_mellon_keycloak_${FED_RHSSO_REALM}.conf"

config_tar_filename="rhssso_config.tar.gz"
config_tar_pathname="${stage_dir}/${config_tar_filename}"
puppet_override_apache_pathname="${stage_dir}/puppet_override_apache.yaml"
puppet_override_keystone_pathname="${stage_dir}/puppet_override_keystone.yaml"

#-----

if [ $# -lt 1 ]; then
    echo "ERROR: no action specified"
    exit 1
fi
action="$1"; shift

if [ $dry_run -ne 0 ]; then
    echo "Dry Run Enabled!"
fi

case $action in
    show-var*)
        show_variables ;;
    initialize)
        initialize ;;
    copy-helper-to-controller)
        copy_helper_to_controller "$1" ;;
    install-mod-auth-mellon)
        install_mod_auth_mellon ;;
    create-ipa-service-account)
        create_ipa_service_account ;;
    client-install)
        client_install ;;
    create-sp-archive)
        create_sp_archive ;;
    fetch-sp-archive)
        fetch_sp_archive ;;
    deploy-mellon-configuration)
        deploy_mellon_configuration ;;
    create-v3-rcfile)
        create_v3_rcfile "$1" ;;

```

```
puppet-override-apache)
    puppet_override_apache "$1" ;;
puppet-override-keystone)
    puppet_override_keystone "$1" ;;
create-federated-resources)
    create_federated_resources ;;
create-mapping)
    create_mapping "$1" ;;
openstack-create-idp)
    openstack_create_idp "$1" ;;
openstack-create-mapping)
    openstack_create_mapping "$1" "$2" ;;
openstack-create-protocol)
    openstack_create_protocol "$1" "$2" ;;
*)
    echo "unknown action: $action"
    usage
    exit 1
;;
esac
```

## CHAPTER 7. THE FED\_VARIABLES FILE

```
# FQDN of IPA server
FED_IPA_HOST="jdennis-ipa.example.com"

# Base DN of IPA server
FED_IPA_BASE_DN="dc=example,dc=com"

# IPA administrator password
FED_IPA_ADMIN_PASSWD="FreeIPA4All"

# Password used by RH-SSO service to authenticate to IPA
# when RH-SSO obtains user/group information from IPA as part of
# RH-SSO's User Federation.
FED_IPA_RHSSO_SERVICE_PASSWD="rhssso-passwd"

# RH-SSO server IP address
FED_RHSSO_IP_ADDR="10.0.0.12"

# RH-SSO server FQDN
FED_RHSSO_FQDN="jdennis-rhssso-7"

# URL used to access the RH-SSO server
FED_RHSSO_URL="https://$FED_RHSSO_FQDN"

# Administrator password for RH-SSO server
FED_RHSSO_ADMIN_PASSWORD=FreeIPA4All

# Name of the RH-SSO realm
FED_RHSSO_REALM="openstack"

# Host name of the mellon server
# Note, this is identical to the Keystone server since Keystone is
# being front by Apache which is protecting it's resources with mellon.
FED_KEYSTONE_HOST="overcloud.localdomain"

# Port number mellon is running on the FED_KEYSTONE_HOST
# Note, this is identical to the Keystone server port
FED_KEYSTONE_HTTPS_PORT=13000

# Name assigned in OpenStack to our IdP
FED_OPENSTACK_IDP_NAME="rhssso"

# Name of our Keystone mapping rules
FED_OPENSTACK_MAPPING_NAME="${FED_OPENSTACK_IDP_NAME}_mapping"
```