RH-SSO TripleO Federation Documentation

Release 0

John Dennis

Apr 18, 2017

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

This document will provide you with instructions and guidance for setting up OpenStack Keystone federation in a high availability TripleO environment authenticating against a Red Hat Single Sign-On (RH-SSO) server.

1.1 Operational Goals

Upon completion, authentication will be federated in OpenStack with the following characteristics:

- Federation will be SAML based.
- The Identity Provider (IdP) will be RH-SSO and will be external to the OpenStack TripleO deployment.
- The RH-SSO IdP will utilize IPA as it's Federated User backing store. In other words, users and groups are managed in IPA and RH-SSO will reference the user and group information in IPA.
- Users in IPA with the right to access OpenStack will be added to the IPA group `openstack-users`.
- OpenStack Keystone will have group named `federated_users`. Members of the `federated_users` group will have the `Member` role granting permission on the project.
- During federated authentication members of the IPA group `openstack-users` will be mapped into the OpenStack group `federated_users`. Thus to be able to login to OpenStack an IPA user will need to be a member of the `openstack-users` group, if the user is not a member of the IPA group `openstack-users` authentication will fail.

1.2 Assumptions

The following assumptions are made:

- There is a RH-SSO server and you 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. Since federated IdP's are by definition external the RH-SSO server is assumed to be external to the TripleO overcloud.
- There is an IPA instance also external to the TripleO overcloud where users and groups are managed. RH-SSO will use IPA as it's User Federation backing store.
- The OpenStack installation is OSP-10 (a.k.a. OpenStack Newton)
- The TripleO overcloud installation is high availability.
- Only the TripleO overcloud will have federation enabled, the undercloud is not federated.
- TLS will be used for *all* external communication.
- All nodes will have Fully Qualified Domain Name (FQDN).

- HAProxy terminates TLS frontend connections, servers running behind HAProxy do not utilize TLS.
- Pacemaker is used to manage some of the overcloud services, including HAProxy.
- TripleO has been run and an overcloud deployed.
- You know how to SSH into the undercloud node as well as the overcloud nodes.
- The examples in the Keystone Federation Configuration Guide will be followed.
- On the `undercloud-0` node you will 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 will install the helper files into the home directory of the `heat-admin` user and work in the `heat-admin` user home directory.

1.3 Prerequisites

- RH-SSO server external to the TripleO overcloud.
- IPA server external to the TripleO overcloud.
- TripleO has been run and an overcloud deployed.

1.4 How to login to the undercloud and overcloud nodes

1. ssh as root into the node hosting the deployment, for example:

```
ssh root@xxx
```

2. ssh into the undercloud node via:

```
ssh undercloud-0
```

3. Become the `stack` user via:

```
su - stack
```

4. Source the overcloud configuration which sets up OpenStack environment variables via:

```
source overcloudrc
```

Note: Currently TripleO sets up Keystone to use the Keystone v2 API but we will be using the Keystone v3 API. Later on in the guide we will create a `overcloudrc.v3` file. From that point on you should use the v3 version of the overcloudrc file. See *Step 8: Use the Keystone Version 3 API*

At this point you can issue commands via the `openstack` command line tool which will operate on the overcloud (even though you're currently still sitting in an undercloud node. If you wish to get into one of the overcloud nodes you will ssh into it as the `heat-admin` user. For example:

```
ssh heat-admin@controller-0
```

1.5 Understanding High Availability

Detailed information on high availability can be found in the Understanding Red Hat OpenStack Platform High Availability guide.

TripleO distributes redundant copies of various OpenStack services across the overcloud deployment. These redundant services will be deployed on the overcloud controller nodes, TripleO names these nodes controller-0, controller-1, controller-2 and so on depending on how many controller nodes TripleO was configured to set up.

The IP address of the controller nodes are private to the overcloud, they are not externally visible. This is because the services running on the controller nodes are HAProxy backend servers. There is one pubically 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 will select a backend server to service the request.

The overcloud is organized as a high availability cluster. Pacemaker manages the cluster, it performs health checks and can fail over to another cluster resource if the resource stops functioning. Pacemaker also knows how to start and stop resources.

HAProxy serves a similar role as Pacemaker, it too performs health checks on the backend servers and only forwards requests to functioning backend servers. There is a *copy of HAProxy running on all the controller nodes*. Since HAProxy is responsible for distributing frontend requests to a pool of backend servers on the controller nodes how is it possible to have a copy of HAProxy running on each controller node? Wouldn't they conflict with each other? How could multiple copies of HAProxy coordinate the distribution of requests across multiple backends? The reason HAProxy works in the cluster environment is because although there are N copies of HAProxy running only one is actually fielding requests at any given time. The active HAProxy instance is managed by Pacemaker. If Pacemaker detects HAProxy has failed it reassigns the frontend IP address to a different HAProxy instance which then becomes the controlling HAProxy instance. Think of it as high availability for high availability. The instances of HAProxy that are kept in reserve by Pacemaker are running but they never see an incoming connection because Packemaker has set the networking so that connections only route to the active HAProxy instance.

Note: Services that are under control of Pacemaker *must* not be managed by direct use of `systemctl` on a controller node. Instead of

```
sudo systemctl restart haproxy
```

the `pcs` Pacemaker command should be used instead.

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

We know `haproxy-clone` is the resource name because that is what appears when we run the Pacemaker status command

```
sudo pcs status
```

which will print something akin to this, plus other information:

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

Note: Many of the steps in this document require complicated commands to be run. To make that task easier and to allow for repeatability if you need to redo a step all the commands have been gathered into a master shell script called `configure-federation`. Each individual step can be executed by passing the name of the step to `configure-federation`. The list of possible commands can be seen by using the help option (-h or –help).

Tip: Often it will be useful to know exactly what the exact command will be after variable substitution when the `configure-federation` script executes it.

`-n` is dry run mode, nothing will be modified, the exact operation will instead be written to stdout.

`-v` is verbose mode, the exact operation will be written to stdout just prior to executing it. This is useful for logging.

Note: Some values used in this document are by necessity site specific. If site specific values were to be directly incorporated into this document it would be confusing and the source of errors when trying to replicate the steps described here. To remedy this any site specific values referenced in this document will be in the form of a variable. The variable name will start with a dollar-sign ($) and be all-caps with a prefix of `FED_`. For example the URL used to access the RH-SSO server would be:

`$FED_RHSSO_URL`

Site specific values can always be identified by searching for `$FED_`

Site specific values utilized by the `configure-federation` script are gathered into the file `fed_variables`. You will need to edit this file to adjust the parameters specific to your deployment.

CHAPTER

TWO

ISSUES WHEN A HTTP SERVER IS BEHIND A PROXY OR SSL TERMINATOR

When a server sits behind a proxy the environment it sees is different than what the client sees as the public identity of the server. A backend server may have a different hostname, listen on a different port and use a different protocol than what a client sees on the front side of the proxy. For many web apps this is not a major problem. Typically most of the problems occur when a server has to generate a self-referential URL (perhaps because it will redirect the client to a different URL on the same server). The URL's the server generates must match the public address and port as seen by the client.

Authentication protocols are especially sensitive to the host, port and protocol (http vs. https) because they often need to assure a request was targeted at a specific server, on a specific port and on a secure transport. Proxies can play havoc with this vital information because by definition a proxy transforms a request received on it's public frontend before dispatching it to a non-public server in the backend. Likewise responses from the non-public backend server sometimes need adjustment so it appears as if the response came from the pubic frontend of the proxy.

There are various approaches to solving the problem. Because SAML is sensitive to host, port and protocol and because we are configuring SAML behind a high availability proxy (HAProxy) we must deal with these issues or things will fail (often in cryptic ways).

2.1 Server/Host name

The host and port appear in several contexts:

- The host and port in the URL the client used
- The host HTTP header inserted into the HTTP request (derived from the client URL host.
- The hostname of the front facing proxy the client connects to (actually the FQDN of the IP address the proxy is listening on)
- The host and port of the backend server which actually handled the client request.
- The **virtual** host and port of the server that actually handled the client request.

It is vital to understand how each of these is utilized otherwise there is the opportunity for the wrong host and port to be used with the consequence the authentication protocols may fail because they cannot validate who the parties in the transaction are.

Let's begin with the backend server handling the request because this is where the host and port are evaluated and most of the problems occur. The backend server need to know:

- The URL of the request (including host & port)
- It's own host & port

Apache supports virtual name hosting. This allows a single server to host multiple domains. For example a server running on example.com might service requests for both bigcorp.com and littleguy.com. The latter 2 names are virtual host names. Virtual hosts in Apache are configured inside a server configuration block, for example:

```
<VirtualHost>
  ServerName bigcorp.com
</VirtualHost>
```

When Apache receives a request it deduces the host from the `HOST` HTTP header. It then tries to match the host to the `ServerName` in it's collection of virtual hosts.

The `ServerName` directive sets the request scheme, hostname and port that the server uses to identify itself. The behavior of the `ServerName` directive is modified by the `UseCanonicalName` directive. When `UseCanonicalName` is enabled Apache will use the hostname and port specified in the `ServerName` directive to construct the canonical name for the server. This name is used in all self-referential URLs, and for the values of SERVER_NAME and SERVER_PORT in CGIs. If `UseCanonicalName` is Off Apache will form self-referential URLs using the hostname and port supplied by the client, if any are supplied.

If no port is specified in the `ServerName`, then the server will use the port from the incoming request. For optimal reliability and predictability, you should specify an explicit hostname and port using the `ServerName` directive. If no `ServerName` is specified, the server attempts to deduce the host by first asking the operating system for the system hostname, and if that fails, performing a reverse lookup on an IP address present on the system. Obviously this will produce the wrong host information when the server is behind a proxy therefore use of the `ServerName` directive is essential.

The Apache ServerName doc is very clear concerning the need to fully specify the scheme, host, and port in the `Server` name directive when the server is behind a proxy, it states:

> Sometimes, the server runs behind a device that processes SSL, such as a reverse proxy, load balancer or SSL offload appliance. When this is the case, specify the https:// scheme and the port number to which the clients connect in the ServerName directive to make sure that the server generates the correct self-referential URLs.

When proxies are in effect the `X-Forwarded-*` HTTP headers come into play. These are set by proxies and are meant to allow an entity processing a request to recognize the request was forwarded and what the original values were *before* being forwarded.

The TripleO HAProxy configuration sets the `X-Forwarded-Proto` HTTP header based on whether the front connection utilized SSL/TLS or not via this configuration:

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

To make matters interesting core Apache **does not** interpret this header thus responsibility falls to someone else to process it. In our situation where HAProxy terminates SSL prior to the backend server processing the request the fact the `X-Forwarded-Proto` HTTP header is set to https is **irrelevant** because Apache does not utilize the header when an extension module such as mellon asks for the protocol scheme of the request. This is why it is **essential** to have the `ServerName` directive include the scheme::///host:port and `UseCanonicalName` is enabled otherwise Apache extension modules such as `mod_auth_mellon` will not function properly behind a proxy.

But what about web apps hosted by Apache behind a proxy? It turns out it's the web app (or rather the web app framework) responsibility to process the forwarded header. Thus apps handle the protocol scheme of a forwarded request differently than Apache extension modules do.

Since Horizon is a Django web app it's Django responsibility to process the `X-Forwarded-Proto` header. This issue aries with the `origin` query parameter used by Horizon during authentication. Horizon adds a `origin` query parameter to the Keystone URL it invokes to perform authentication. The `origin` parameter is used by Horizon to redirect back to original resource.

The `origin` parameter generated by Horizon may incorrectly specify http as the scheme instead of https despite the fact Horizon is running with https enabled. This occurs because Horizon calls function `build_absolute_uri()` to form the `origin` parameter. It is entirely up to the Django to identify the scheme because `build_absolute_url()` is ultimately implemented by Django. You can force Django to process the `X-Forwarded-Proto` via a special configuration directive. This is documented in the Django secure-proxy-ssl-header documentation.

This can be enabled in the `/etc/openstack-dashboard/local_settings` file by uncommenting this line:

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

Note, Django prefixes the header with "HTTP_" and converts hyphens to underscores.

After uncommenting this the Origin parameter will correctly use the https scheme.

Note that even when the `ServerName` directive includes the https scheme the Django call `build_absolute_url()` will not use the https scheme. Thus for Django you *must* use the `SECURE_PROXY_SSL_HEADER` override, specifying the scheme in `ServerName` directive will not work.

The critical thing to note is is that Apache extension modules and web apps process the request scheme of a forwarded request differently demanding **both** the `ServerName` and `X-Forwarded-Proto` HTTP header techniques be utilized.

CHAPTER

THREE

SET UP IPA

IPA will be external to the OpenStack TripleO deployment and will be the source of all user and group information. RH-SSO will be configured to use IPA as it's User Federation. RH-SSO will perform an LDAP search against IPA to obtain user and group information.

3.1 Create IPA system account for RH-SSO

Although IPA permits anonymous binds some information will be withheld for security reasons. Some of the information withheld during anonymous binds is essential for RH-SSO User Federation therefore RH-SSO will need to bind to the IPA LDAP server with sufficient privileges to obtain the required information.

To this end we will need to set up a service account in IPA dedicated to RH-SSO. Unfortunately IPA does not provide a command to perform this, instead IPA's LDAP server will need to be modified directly by use of the `ldapmodify` command.

This can be performed like this:

```
ldap_url="ldaps://$FED_IPA_HOST"
dir_mgr_dn="cn=Directory Manager"
service_name="rhsso"
service_dn="uid=$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
```

Tip: Use `configure-federation` script to perform the above.

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

3.2 Create a test user

You'll need a user in IPA to test with. You can either use an existing user or create a new user to test with. The examples in this document use the user John Doe with a uid of `jdoe`. You can create the `jdoe` user in IPA like this:

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

Assign a password to the user:

```
$ ipa passwd jdoe
```

3.3 Create IPA group for OpenStack users

Create the `openstack-users` group in IPA.

1. Make sure the `openstack-users` group does not already exist:

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

2. Add the `openstack-users` group to IPA:

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

Add the test user to the `openstack-users` group like this:

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

Verify 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

FOUR

SET UP RH-SSO

Installing RH-SSO is beyond the scope of this document. It is assumed you have already installed RH-SSO on a node independent of the OpenStack TripleO deployment. The RH-SSO URL will be identified by the `$FED_RHSSO_URL` variable.

RH-SSO supports multi-tenancy. To provide separation between tenants of RH-SSO realms are used, thus in RH-SSO operations always occur in the context of a realm. This document will use the site specific variable `$FED_RHSSO_REALM` to identity the RH-SSO realm being used.

Note: The RH-SSO realm can either be created ahead of time as would be typical when RH-SSO is administered by an IT group or the `keycloak-httpd-client-install` tool can create it for you if you have administrator privileges on the RH-SSO server.

Once the RH-SSO realm is available it will be necessary to configure that realm for User Federation against IPA.

In the RH-SSO administration web console perform the following actions:

1. Select $FED_RHSSO_REALM from drop down list in upper left corner
2. Select `User Federation` from the left side `Configure` panel
3. From the `Add provider ...` drop down list in the upper right corner of the `User Federation` panel select `ldap`.
4. Fill in the following fields with these values, be sure to substitute any `$FED_` site specific variable.

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
Connection URL	LDAPS://$FED_IPA_HOST
Users DN	cn=users,cn=accounts,$FED_IPA_BASE_DN
Authentication Type	simple
Bind DN	uid=rhsso,cn=sysaccounts,cn=etc,$FED_IPA_BASE_DN
Bind Credential	$FED_IPA_RHSSO_SERVICE_PASSWD

5. Use the `Test connection` and `Test authentication` buttons to assure user federation is working.
6. Click `Save` at the bottom of the `User Federation` panel to save the new user federation provider.
7. Now click on the `Mappers` tab at the top of the `Red Hat IDM` user federtion page you just created.

8. Create a mapper to retrieve user's group information so that the groups a user is a member of will be returned in the SAML assertion. Later we'll be using group membership to provide authorization in OpenStack.
9. Click on the `Create` button in the upper right hand corner of the `Mappers` page.
10. On the `Add user federation mapper` page select `group-ldap-mapper` from the `Mapper Type` drop down list and give it the name "Group Mapper".
11. Fill in the following fields with these values, be sure to substitute any `$FED_` site specific variable.

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

12. Click `Save`.

4.1 Add user attributes to be returned in SAML assertion

The SAML assertion can convey properties bound to the user (e.g. user metadata), these are called attributes in SAML. We must instruct RH-SSO to return specific attributes in the assertion that we depend upon. When Keystone receives the SAML assertion it will map those attributes into metadata about the user which Keystone can understand. The process of mapping IdP attributes into Keystone data is called Federated Mapping and will be discussed elsewhere in this document, see *Step 21: Create mapping file and upload into Keystone*

RH-SSO calls the process of adding returned attributes "Protocol Mapping". Protocol mapping is a property of the RH-SSO client (e.g. the service provider (SP) added to the RH-SSO realm). The process for adding SAML any given attribute follows a very similar pattern.

In RH-SSO administration web console perform the following actions:

1. Select $FED_RHSSO_REALM from drop down list in upper left corner
2. Select `Clients` from the left side `Configure` panel
3. Select the SP client that was setup by `keycloak-httpd-client-install`. It will be identified by it's SAML `EntityId`.
4. Select the `Mappers` tab from the horizontal list of tabs appearing at the top of the client panel.
5. In the `Mappers` panel in the upper right are 2 buttons, `Create` and `Add Builtin`. Use one of these buttons to add a protocol mapper to the client.

You can add any attributes you wish but for this exercise we'll only need the list of groups the user is a member of because group membership is how we will authorize the user.

4.1.1 Add group information to assertion

1. Click on the `Create` button in the `Mappers` panel.
2. In the `Create Protocol Mapper` panel select `Group list` from the `Mapper type` 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 will appear in the SAML assertion. When the Keystone mapper looks for names in the `Remote` section of the mapping declaration it is the SAML attribute names it is looking for. Whenever you add an attribute in RH-SSO to be passed in the assertion you will need to specify the SAML attribute name, it is the RH-SSO protocol mapper where that name is defined.

5. In the `SAML Attribute NameFormat` field select `Basic`.
6. In the `Single Group Attribute` toggle box select `On`.
7. Click `Save` at the bottom of the panel.

Note: `keycloak-httpd-client-install` adds a group mapper when it runs.

CHAPTER

FIVE

STEPS

5.1 Step 1: Determine IP address and matching FQDN

The following nodes will need an FQDN:

- host running the OpenStack Horizon Dashboard
- host running the OpenStack Keystone service (`$FED_KEYSTONE_HOST`)[1]
- host running RH-SSO
- host running IPA.

The OSP-10 TripleO deployment does not set up DNS nor assign FQDN's to the nodes. The authentication protocols require the use of FQDN's as does TLS. Therefore you must determine the external public IP address of the overcloud, yes that's correct, you're looking for the IP address of the overcloud *not* the IP address of an individual node in the overcloud running an overcloud service (e.g. controller-0, controller-1 etc.)

Note: For some steps you will need to know the IP address of the controller nodes, See *Determine controller nodes IP Addresses*

What is going on here? You may be used to running a service on a particular node. If you're not familiar with high availability clusters IP addresses are assigned to a cluster as opposed to a node might seem strange. Pacemaker and HAProxy work in conjunction to provide the illusion there is one IP address and that IP address is entirely distinct from the individual IP address of any given node in the cluster. So the right way to think about what the IP address is for a service in OpenStack is not in terms of what node that service is running on but rather what is the effective IP address the cluster is advertising for that service (e.g. VIP).

But how do you find out what that IP address is? You'll need to assign a name to it in lieu of DNS.

There are two ways go about this. First of all is the observation that TripleO 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 know public IP address of one service in the OpenStack cluster then you know all of them (unfortunately 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 us the public Keystone IP address is 10.0.0.101 and 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.

[1] In a high availablity environment more than one host will run a service therefore the IP address is not a host address but rather the IP address bound to the service.

But a more robust way to determine the IP addresses and port numbers is by examining the HAProxy configuration file (`/etc/haproxy/haproxy.cfg`) 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.

To examine the `haproxy.cfg` ssh into one of the cluster's controller nodes and examine the file `/etc/haproxy/haproxy.cfg`. As noted above it does not matter which controller node you select.

The `haproxy.cfg` file is divided into sections, each section begins with a `listen` and then 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 some are cluster internal. 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.

Of the several `bind` lines in the section how do you know which is the public IP address and port of the service? TripleO seems to always put the public IP address as the first `bind`. Also the public IP address should support TLS and as such the `bind` line will have the `ssl` keyword. Also the IP address should match the IP address in the `OS_AUTH_URL` located in the `overstackrc` file. For example lets look at the `keystone_public` section in `haproxy.cfg` (this is an example only):

```
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
```

The first `bind` line has the ssl keyword and the IP address matches that of the `OS_AUTH_URL` located in the `overstackrc` file. Therefore we're confident that Keystone is publicly accessed at the IP address of 10.0.0.101 on port 13000. The second `bind` line is cluster internal, used by other OpenStack services running in the cluster (note it does not use TLS because it's not public).

`mode http` indicates the protocol in use will be HTTP, this allows HAProxy to examine HTTP headers and so forth.

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 }
```

deserve attention and will be covered in more detail in *Server/Host name*. They guarantee that the HTTP header `X-Forwarded-Proto` will be set and seen by the backend server. The backend server in many cases needs to know if the client was using `HTTPS`. But HAProxy terminates TLS and the backend server will see the connection as non-TLS. The `X-Forwarded-Proto` HTTP header is a mechanism which allows the backend server identify what protocol the client was actually using instead of what protocol the request arrived on. It is *essential* that a client 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 received from the client or the proxy can forcefully set it thus preventing any malicious use by the client. This is what occurs here, `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 so the backend server can 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 *Server/Host name* for more information on forwarded proto, redirects, ServerName, etc.

The following line will assure 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 says if the request was received on the public IP address (e.g. 10.0.0.101) and it wasn't https then perform a 301 redirect and set the scheme to HTTPS.

HTTP servers (e.g. Apache) often generate self referential URL's for redirect purposes. The redirect location must indicate the correct protocol. But if server is behind a TLS terminator it will think it's redirection URL should use http not https. This line:

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

says if you see a Location header in the response and it has the http scheme then rewrite it to use the https scheme.

5.1.1 Set host variables and name the host

Using either this line in the `overcloudrc`:

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

or this line from the `keystone_public section in the ``haproxy.cfg` file:

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

determine the IP address and port (in this example the IP address is 10.0.0.101 and the port is 13000). We must also give the IP address a FQDN. In our example will use the FQDN `overcloud.localdomain`. Because DNS is not being used the FQDN for the IP address should be put in the `/etc/hosts` file like this:

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

Note: TripleO will probably have already done this on the overcloud nodes but you may need to add the host entry on any external hosts that participate.

The `$FED_KEYSTONE_HOST` and `$FED_KEYSTONE_HTTPS_PORT` need to be set in the `fed_variables` file. Using the example values from above it would be:

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

Note: Because Mellon is running on the Apache server hosting Keystone the Mellon host:port and the Keystone host:port will be the same.

Warning: If you do `hostname` on one of the controller nodes it will proablaby be something like this: `controller-0.localdomain`, but that is it's *internal cluster* name, not it's public IP address. You need to use the *public IP address*.

Determine controller nodes IP Addresses

To determine the IP addresses of the controller nodes you will use `openstack` command line tool on the `undercloud` node. To do this:

```
% ssh undercloud-0
% sudo -l stack
% source stackrc
% openstack server list
```

You should get a result that looks something like this:

```
+--------------------------------------+--------------+--------+----------------------
↪--+----------------+
| ID                                   | Name         | Status | Networks             ␣
↪  | Image Name     |
+--------------------------------------+--------------+--------+----------------------
↪--+----------------+
| 9cc6eeb8-fa08-481b-8acd-4741caa193d7 | controller-2 | ACTIVE | ctlplane=192.168.24.
↪18 | overcloud-full |
| 9756f531-08f2-496d-b64f-72342fdb771c | controller-1 | ACTIVE | ctlplane=192.168.24.
↪9  | overcloud-full |
| b66e7236-ed58-49ac-9956-6e2dcdc5117d | controller-0 | ACTIVE | ctlplane=192.168.24.
↪13 | overcloud-full |
| 33e3ac1e-0606-4138-9700-ca5f7d4c7ae8 | compute-0    | ACTIVE | ctlplane=192.168.24.
↪15 | overcloud-full |
+--------------------------------------+--------------+--------+----------------------
↪--+----------------+
```

The `Name` column identifies the server and the `Networks` column provides the IP address. You may want to add these to your `/etc/hosts` file to make life easier in steps that require you to ssh into a controller node. For example using the above you would add the following lines to your `/etc/hosts` file:

```
192.168.24.13 controller-0
192.168.24.9  controller-1
192.168.24.18 controller-2
```

5.2 Step 2: Install helper files on undercloud-0

Copy the `configure-federation` and `fed_variables` files into the `~stack` home directory on undercloud-0

5.3 Step 3: Set your deployment variables

The file `fed_variables` contains variables specific to your federation deployment. These are referenced in this document 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 utilize the $ variable syntax, e.g. $FED_. Make sure every FED_ variable in fed_variables is provided a value.

5.4 Step 4: Copy helper files from undercloud-0 to controller-0

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, this can be done like this:

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

Tip: Use configure-federation script to perform the above.

./configure-federation copy-helper-to-controller

5.5 Step 5: Initialize working environment on undercloud node

On the undercloud node:

1. Become the stack user.
2. Create the‘‘fed_deployment‘‘ directory, this is where we will stash files as we work.

This can be done like this:

```
% su - stack
% mkdir fed_deployment
```

Tip: Use configure-federation script to perform the above.

./configure-federation initialize

5.6 Step 6: Initialize working environment on controller-0

From the undercloud node:

1. ssh into the controller-0 node as the heat-admin user
2. Create the‘‘fed_deployment‘‘ directory, this is where we will stash files as we work.

This can be done like this:

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

Tip: Use configure-federation script to perform the above. Execute from the controller-0 node.

./configure-federation initialize

5.7 Step 7: Install mod_auth_mellon on each controller node

From the undercloud node:

1. ssh into the controller-n node as the `heat-admin` user
2. install the `mod_auth_mellon` RPM

This can be done like this:

```
% ssh heat-admin@controller-n # replace n with controller number
% sudo yum -y install mod_auth_mellon
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation install-mod-auth-mellon

5.8 Step 8: Use the Keystone Version 3 API

In order to use the `openstack` command line client to work with the overcloud certain parameters must be setup that provide you access. Normally this is done by *sourcing* an *rc* file in your shell that sets environment variables. TripleO will have created an `overcloudrc` file for this purpose in the home directory of the stack user in the undercloud-0 node. Unfortunately the `overcloudrc` file is setup to use the v2 version of the Keystone API. But federation requires the use of the `v3` Keystone API. Therefore we need to create a new *rc* file targeting the `v3` Keystone API. This can be done like this:

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

Then 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
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation create-v3-rcfile

From this point forward to work with the overcloud you will use the `overcloudrc.v3` file. The basic steps are:

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

5.9 Step 9: Add the RH-SSO FQDN to /etc/hosts on each controller

mellon will be running on each controller node. mellon will be configured to connect to the RH-SSO IdP. If the FQDN of the RH-SSO IdP is not resolvable via DNS then you will have to manually add the FQDN to the `/etc/hosts` file on *each* controller node after the Heat Hosts Section:

```
% ssh heat-admin@controller-n
% sudo $EDITOR /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
```

5.10 Step 10: Install & configure mellon on 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.

Recall this is a high availability deployment and as such there will be multiple overcloud controller nodes each running identical copies. Therefore the mellon setup will need to be replicated across each controller node. The way we will tackle this is to install and configure mellon on controller-0 and then gather up all the configuration files the `keycloak-httpd-client-install` tool created into an archive (i.e. tar file) and then let swift copy the archive over to each controller and unarchive the files there.

This can be done like this:

```
% ssh heat-admin@controller-0
% yum -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 \
 -l "/v3/auth/OS-FEDERATION/websso/mapped" \
 -l "/v3/auth/OS-FEDERATION/identity_providers/rhsso/protocols/mapped/websso" \
 -l "/v3/OS-FEDERATION/identity_providers/rhsso/protocols/mapped/auth"
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation client-install

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
```

5.11 Step 11: Adjust the mellon configuration

Although `keycloak-httpd-client-install` does a good job of configuring mellon it cannot know all the needs of a particular deployment.

We will be utilizing a list of groups during the IdP assertion to Keystone mapping phase. The Keystone mapping engine expects lists to be one value with items separated by a semicolon (;). Therefore we must tell mellon when it receives multiple values for an attribute it should combine the multiple attributes into a single value with items separated by a semicolon, this mellon directive will accomplish that:

```
MellonMergeEnvVars On ";"
```

To do this:

```
% $EDITOR /etc/httpd/conf.d/v3_mellon_keycloak_openstack.conf
```

Find the `<Location /v3>` block and add a line to that block, for example:

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

5.12 Step 12: Make an archive of the generated configuration files

Because the mellon configuration need to be replicated across all controller nodes we will create an archive of the files thus allowing us to install the exact same file contents on each controller node. We will locate the archive in the `~heat-admin/fed_deployment` sub-directory

You can create a compressed tar archive like this:

```
% mkdir fed_deployment
% tar -cvzf rhsso_config.tar.gz \
--exclude '*.orig' \
--exclude '*~' \
/etc/httpd/saml2 \
/etc/httpd/conf.d/v3_mellon_keycloak_openstack.conf
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation create-sp-archive

5.13 Step 13: Retrieve the mellon configuration archive

Back on the undercloud node we need to fetch the archive we just created. We also need to unarchive the files because in subsequent steps we will need access to some of the data (e.g. the entityID of the RH-SSO IdP). This can be done like this on the undercloud-0 node:

```
% scp heat-admin@controller-0:/home/heat-admin/fed_deployment/rhsso_config.tar.gz \
~/fed_deployment
% tar -C fed_deployment -xvf fed_deployment/rhsso_config.tar.gz
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation fetch-sp-archive

5.14 Step 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 a sensible precaution, it prevents Apache from operating in any fashion other then the configuration enforced by Puppet. However this runs afoul of our 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 such time as the mellon files are under control of puppet we will have to disable `apache::purge_configs` flag.

Warning: 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 we will create a puppet file containing the override and add the override file to the list of puppet files utilized when `overcloud_deploy.sh` is run.

Create this file `fed_deployment/puppet_override_apache.yaml` with this content:

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

Then add the file just created 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
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation puppet-override-apache

5.15 Step 15: Configure Keystone for federation

We want to utilize domains in Keystone which requires some extra setup. The Keystone puppet module knows how to perform this extra configuration step provided it is enabled. In one of the puppet yaml files we will need to add:

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

Some additional values need to be set in `/etc/keystone/keystone.conf` to enable federation:

- auth:methods
- federation:trusted_dashboard
- federation:sso_callback_template
- federation:remote_id_attribute

Here is an explanation of these configuration value and their suggested values:

auth:methods A list of allowed authentication methods. By default the list is: `['external', 'password', 'token', 'oauth1']`. We need to assure SAML is included which it is not by default. SAML is enabled via the `mapped` method. Thus 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 our deployment the value would be: `https://$FED_KEYSTONE_HOST/dashboard/auth/websso/` Note: the host is $FED_KEYSTONE_HOST only because TripleO co-locates both Keystone and Horizon on the same host. If Horizon is running on a different host than Keystone adjust accordingly.

federation:sso_callback_template Absolute path to an 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 Value used to obtain the entity ID of the Identity Provider from the environment. For `mod_auth_mellon` we will use `MELLON_IDP`. Note, this is set in the mellon configuration file via the `MellonIdP IDP` directive.

Create this file `fed_deployment/puppet_override_keystone.yaml` 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
```

Then add the file just created 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_keystone.yaml \
--log-file overcloud_deployment_14.log &> overcloud_install.log
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation puppet-override-keystone

5.16 Step 16: Deploy the mellon configuration archive

We'll use swift artifacts to install the mellon configuration files on each controller node. This can be done like this:

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

Tip: Use `configure-federation` script to perform the above.

./configure-federation deploy-mellon-configuration

5.17 Step 17: Redeploy the overcloud

In prior steps we made changes to the puppet yaml configuration files and swift artifacts. These now need to be applied which can be performed like this:

```
./overcloud_deploy.sh
```

Warning: In subsequent steps other configuration changes will be made on the overcloud controller nodes. Re-running puppet via 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 to configuration files on the overcloud controller nodes.

5.18 Step 18: Use proxy persistence for Keystone on each controller

With high availability any one of multiple backend servers might field a request. Because of the number of redirections utilized in SAML and the fact each of those redirections involves state information it is vital the same server will process all the transactions. In addition a session will be established by `mod_auth_mellon`. Currently `mod_auth_mellon` is not capable of sharing it's state information across multiple server therefore we 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 via either affinity or persistence. This article on HAProxy Sticky Sessions provides good back ground material.

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

Persistence is implemented though the use of cookies. The HAProxy `cookie` directive names the cookie which will be used for persistence along with parameters controlling it's use. The HAProxy `server` directive has a `cookie` option that sets the value of the cookie, it should be set to the name of the server. If an incoming request does not have a cookie identifying the backend server then HAProxy selects a server based on it's configured balancing algorithm. HAProxy assures the cookie is set to the name of the selected server in the response. If the incoming request has a cookie identifying a backend server then HAProxy automatically selects that server to handle the request.

To enable persistence in the `keystone_public` block of the `/etc/haproxy/haproxy.cfg` configuration this line must be added:

```
cookie SERVERID insert indirect nocache
```

This says `SERVERID` will be the name of our persistence cookie. Then we 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, the other parts of the server directive have been omitted for clarity.

5.19 Step 19: Create federated resources

Recall from the introduction that we are going to follow the example setup for federation in the Create keystone groups and assign roles section of the Keystone federation documentation. Perform the following steps on the undercloud node as the `stack` user after having sourced 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 --project federated_project _member_
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation create-federated-resources

5.20 Step 20: Create the identity provider in OpenStack

We must register our IdP with Keystone. This operation provides a binding between the entityID in the SAML assertion and the name of the IdP in Keystone. First we must find the entityID of the RH-SSO IdP. This appears in the IdP metadata which was obtained when `keycloak-httpd-client-install` was run. The IdP metadata is stored in the `/etc/httpd/saml2/v3_keycloak_$FED_RHSSO_REALM_idp_metadata.xml` file. Recall from an earlier step we fetched the archive of the mellon configuration files and then unarchived it in our `fed_deployment` work area. Thus you can find the IdP metadata in `fed_deployment/etc/httpd/saml2/v3_keycloak_$FED_RHSSO_REALM_idp_metadata.xml`. In the IdP metadata file is a `<EntityDescriptor>` element with a `entityID` attribute. We need the value of the `entityID` attribute and

for illustration purposes we'll assume it's been stored in the `$FED_IDP_ENTITY_ID` variable. We will name our IdP `rhsso` which we have assigned to the variable $FED_OPENSTACK_IDP_NAME. This can be done like this:

```
openstack identity provider create --remote-id $FED_IDP_ENTITY_ID $FED_OPENSTACK_IDP_
↪NAME
```

Tip: Use `configure-federation` script to perform the above.

./configure-federation openstack-create-idp

5.21 Step 21: Create mapping file and upload into Keystone

Keystone performs a mapping from the SAML assertion it receives from the IdP to a format 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.

These are the mapping rules we will be using for our example as explained 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 2 parts `local` and `remote`. The way the mapping engine works is it iterates over the list of rules until one matches and then executes it. A rule matches only if *all* the conditions in the `remote` part of the rule match. In our 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 what this is doing is as follows: If the IdP successfully authenticates the user and the IdP asserts that user belongs to the group `openstack-users` then we will allow that user to operate in OpenStack with the privileges bound to the `federated_users` group in Keystone.

To create the mapping in Keystone you must create a file containing the mapping rules and then upload it into Keystone giving it a name so it can be referenced. We will create the mapping file in our fed_deployment directory, e.g. `fed_deployment/mapping_${FED_OPENSTACK_IDP_NAME}_saml2.json` and assign the mapping rules the name `$FED_OPENSTACK_MAPPING_NAME`. The mapping file can then be uploaded like this:

```
openstack mapping create --rules fed_deployment/mapping_rhsso_saml2.json $FED_
↪OPENSTACK_MAPPING_NAME
```

Tip: Use `configure-federation` script to perform the above as 2 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

---

5.22 Step 22: Create a Keystone federation protocol

Keystone binds an IdP using a specific protocol (e.g. mapped) to a mapping via a Keystone protocol definition. To establish this binding do the following:

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

Tip: Use `configure-federation` script to perform the above.

./configure-federation openstack-create-protocol

5.23 Step 23: Fully qualify the Keystone scheme, host, and port

On each controller node edit `/etc/httpd/conf.d/10-keystone_wsgi_main.conf` to assure 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>
```

being sure to substitute the correct values for the `$FED_` variables with the values specific to your deployment.

5.24 Step 24: Configure Horizon to use federation

On each controller node edit `/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 = (
    ("saml2", _("RH-SSO")),
    ("credentials", _("Keystone Credentials")),
)
```

being sure to substitute the correct values for the `$FED_` variables with the values specific to your deployment.

5.25 Step 25: Set Horizon to use `X-Forwarded-Proto` HTTP header

On each controller node edit `/etc/openstack-dashboard/local_settings` and uncomment the line:

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

CHAPTER

SIX

TROUBLESHOOTING

6.1 How to test the Keystone mapping rules

It is a good idea to verify 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:

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"]
            }
        ]
    }
]
```

The file `assertion_data.txt` has this content:

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

If you then run this command:

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

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"
    }
  ]
}
```

Tip: If you can also supply the `--engine-debug` command line argument which will emit diagnostic information concerning how the mapping rules are being evaluated.

6.2 How to determine actual assertion values seen by Keystone

The *mapped* assertion values Keystone will utilize are passed as CGI environment variables. To get a dump of what those environment variables are you can do the following:

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 `/etc/httpd/conf.d/10-keystone_wsgi_main.conf` file setting it to run the `test` script by temporarily modifying the `WSGIScriptAlias` directive like this:

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

3. Restart httpd like this:

```
systemctl restart httpd
```

4. Then, try login again and review the information that the script dumps out. When finished, remember to restore the `WSGIScriptAlias` directive, and restart the httpd service again.

6.3 How to see the SAML messages exchanged between the SP and IdP

The `SAMLTracer` Firefox add-on is a wonderful 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 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

SEVEN

GLOSSARY

FQDN Fully Qualified Domain Name

IdP Identity Provider

RH-SSO Red Hat Single Sign-On server functioning as an Identity Provider (IdP)

TripleO OpenStack on OpenStack. An OpenStack installer, see https://wiki.openstack.org/wiki/TripleO

CHAPTER

EIGHT

FOOTNOTES

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search