



Red Hat OpenStack Platform 16.1

Identity サービスとのフェデレーション

Red Hat Single Sign-On を使用した Identity サービスとのフェデレーション

Red Hat OpenStack Platform 16.1 Identity サービスとのフェデレーション

Red Hat Single Sign-On を使用した Identity サービスとのフェデレーション

OpenStack Team
rhos-docs@redhat.com

法律上の通知

Copyright © 2023 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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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.

概要

Red Hat Single Sign-On を使用した Identity サービスとのフェデレーション

目次

多様性を受け入れるオープンソースの強化	4
RED HAT ドキュメントへのフィードバック (英語のみ)	5
第1章 はじめに	6
1.1. 概要	6
1.2. 前提条件	6
1.3. RED HAT OPENSTACK PLATFORM ノードへのアクセス	7
1.4. テクノロジーの概要	8
1.5. CONFIGURATION SCRIPT の使用	9
1.6. プロキシまたは SSL 用語の使用	9
第2章 RED HAT IDENTITY MANAGEMENT の設定	10
2.1. RH-SSO 用の IDM サービスアカウントの作成	10
2.2. テストユーザーの作成	10
2.3. OPENSTACK ユーザー向けの IDM グループの作成	11
第3章 RED HAT SINGLE SIGN-ON の設定	12
3.1. RH-SSO レルムの設定	12
3.2. SAML アサーションを使用したユーザー属性の追加	14
3.3. グループ情報の SAML アサーションへの追加	14
第4章 RED HAT OPENSTACK PLATFORM でのフェデレーションの設定	16
4.1. IP アドレスの取得	16
4.2. ホスト変数の設定およびホストの命名	16
4.3. ヘルパーファイルのインストール	17
4.4. デプロイメント変数の設定	17
4.5. ヘルパーファイルのコピー	17
4.6. 作業環境の初期化	18
4.7. MOD_AUTH_MELLON のインストール	18
4.8. 各コントローラーへの RH-SSO FQDN の追加	18
4.9. コントローラーノードでの MELLON のインストールと設定	19
4.10. MELLON 設定の編集	20
4.11. 生成された設定ファイルのアーカイブの作成	20
4.12. MELLON 設定アーカイブの取得	21
4.13. PUPPET が管理外の HTTPD ファイルを削除しない	21
4.14. IDENTITY サービス (KEYSTONE) でのフェデレーションの設定	22
4.15. MELLON 設定アーカイブのデプロイ	23
4.16. オーバークラウドの再デプロイ	23
4.17. 各コントローラーで IDENTITY サービス (KEYSTONE) にプロキシの永続性を使用する	23
4.18. フェデレーションリソースの作成	24
4.19. RED HAT OPENSTACK PLATFORM でのアイデンティティプロバイダーの作成	24
4.20. マッピングファイルの作成および KEYSTONE にアップロード	25
4.21. KEYSTONE フェデレーションプロトコルの作成	26
4.22. KEYSTONE 設定を完全に修飾します。	27
4.23. HORIZON がフェデレーションを使用するように設定する	27
4.24. X-FORWARDED-PROTO HTTP ヘッダーを使用するように設定する	27
第5章 トラブルシューティング	29
5.1. KEYSTONE マッピングルールのテスト	29
5.2. KEYSTONE で受信されるアサート値を確認する	30
5.3. SP と IDP の間で交換される SAML メッセージを確認します。	30

第6章 CONFIGURE-FEDERATION ファイル 32

第7章 FED_VARIABLES ファイル 47

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[Red Hat CTO である Chris Wright のメッセージ](#) を参照してください。:leveloffset: +0

RED HAT ドキュメントへのフィードバック (英語のみ)

弊社ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。

ドキュメントへのダイレクトフィードバック (DDF) 機能の使用 (英語版のみ)

特定の文章、段落、またはコードブロックに対して直接コメントを送付するには、DDF の **Add Feedback** 機能を使用してください。なお、この機能は英語版のドキュメントでのみご利用いただけます。

1. **Multi-page HTML** 形式でドキュメントを表示します。
2. ドキュメントの右上隅に **Feedback** ボタンが表示されていることを確認してください。
3. コメントするテキスト部分をハイライト表示します。
4. **Add Feedback** をクリックします。
5. **Add Feedback** フィールドにコメントを入力します。
6. (オプション) ドキュメントチームが連絡を取り問題についてお伺いできるように、ご自分のメールアドレスを追加します。
7. **Submit** をクリックします。

第1章 はじめに



警告

現時点では、Red Hat はフェデレーションをサポートしていません。これはテスト目的にのみご利用いただく機能で、実稼働環境にデプロイすべきではありません。

高可用性 Red Hat OpenStack Platform director 環境でフェデレーションを設定するには、以下を設定する必要があります。

- Red Hat Identity Management
- Red Hat Single Sign-On (RH-SSO)
- Red Hat OpenStack Platform オーバークラウド

1.1. 概要

フェデレーション認証は、複数の異なるサービス間で認証を提供する方法です。この認証ソリューションは、アイデンティティプロバイダー (IdP)、サービスプロバイダー (SP) に依存し、Security Assertion Language (SAML) に基づいています。

OpenStack がフェデレーション認証ソリューションのサービスプロバイダーの場合、Red Hat Identity Management (IdM) グループ **openstack-users** のメンバーは、プロジェクトアクセス用の **Member** ロールで OpenStack Keystone グループ **federated_users** にマッピングされます。その結果、そのユーザーを IdM グループ **openstack-users** に追加して、OpenStack へのアクセス権限を付与することができます。

1.2. 前提条件

フェデレーションされた認証をデプロイする前に、以下を完了する必要があります。

- Red Hat OpenStack Platform director とオーバークラウドを、以下の属性でデプロイしている。
 - SSH を使用して、Red Hat OpenStack Platform director と各オーバークラウドノードに接続することができます。
 - 全ノードには完全修飾ドメイン名 (FQDN) があります。
 - TLS 暗号化は、すべての外部通信に使用されます。
 - HAProxy は TLS フロントエンド接続を終了し、HAProxy の背後で実行しているサーバーは TLS を使用しません。
- RH-SSO サーバーが存在し、サーバーに管理者権限があるか、RH-SSO 管理者が独自のレルムを作成し、そのレルムに対する管理者権限をユーザーに付与している。フェデレーションされた IdP は定義によって外部にあるため、RH-SSO サーバーは Red Hat OpenStack Platform director オーバークラウドの外部にあることを前提とします。詳細は、[Installing and configuring RH-SSO](#) および [Creating a realm and user](#) を参照してください。

- IdM サーバーが存在し、ユーザーおよびグループが管理される Red Hat OpenStack Platform director オーバークラウド外にも存在します。RH-SSO は、IdM をユーザーフェデレーション バッキングストアとして使用します。
- [Keystone Federation Configuration Guide](#) に記載の例に従います。
- **undercloud-0** ノードで、ヘルパーファイルを **stack** ユーザーのホームディレクトリーにインストールし、**stack** ユーザーのホームディレクトリーで操作します。
- **controller-0** ノードで、ヘルパーファイルを **heat-admin** ユーザーのホームディレクトリーにインストールし、**heat-admin** ユーザーのホームディレクトリーで動作します。
- **mod_auth_mellon** がコントローラーノードに以前にインストールされている場合は、Puppet Apache クラスにより Puppet の管理下でない Apache 設定ファイルが削除されるため、再インストールが必要になります。



注記

Red Hat OpenStack オーバークラウドでは、フェデレーションのみが有効化されています。Director はフェデレーションされていません。

1.3. RED HAT OPENSTACK PLATFORM ノードへのアクセス

デフォルトでは、オーバークラウドノードにアクセスするには、Red Hat OpenStack Platform director にログインする必要があります。

1. SSH を使用して Red Hat OpenStack director に接続します。

```
# ssh undercloud-0
```

2. **stack** ユーザーになります。

```
$ su - stack
```

3. source コマンドで **stackrc** 設定を読み込み、必要な OpenStack 環境変数を有効にします。

```
$ source stackrc
```

4. source コマンドで **stackrc** を読み込んだあと、Red Hat OpenStack Platform director に対して動作する **openstack** コマンドラインツールを使用してコマンドを発行することができます。オーバークラウドノードのいずれかに直接アクセスするには、**openstack server list** を使用して IP アドレスを取得し、SSH を使用して接続します。

```
(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. テクノロジーの概要

以下のテクノロジーは、Red Hat OpenStack Platform の一部です。

1.4.1. 高可用性

Red Hat OpenStack Platform director は、オーバークラウドのデプロイメント全体にわたって、さまざまな OpenStack サービスの冗長コピーを配信します。これらの冗長サービスは、Red Hat OpenStack Platform director が設定されたコントローラーノードの数に応じて、director がノード **controller-0**、**controller-1**、**controller-2** などの命名を行い、オーバークラウドのコントローラーノードにデプロイされます。

コントローラーノードで実行中のサービスは HAProxy バックエンドサーバーであるため、コントローラーノードの IP アドレスは外部では表示されません。コントローラーノードのセットに公開されている IP アドレスが1つあります。これは HAProxy のフロントエンドです。要求がパブリック IP アドレスでサービスに到達すると、HAProxy は要求に対応するバックエンドサーバーを選択します。

オーバークラウドは高可用性クラスターとして編成されます。[Pacemaker](#) はクラスターを管理し、ヘルスチェックを実行し、リソースが機能停止した場合に別のクラスターリソースにフェイルオーバーすることができます。Pacemaker を使用して、これらのリソースを起動および停止します。

高可用性に関する詳しい情報は、[High Availability Deployment and Usage](#) ガイドを参照してください。

1.4.1.1. Pacemaker サービスの管理

Pacemaker が管理するサービスについては、コントローラーノードで **podman** コマンドを使用して管理しないでください。Pacemaker **pcs** コマンドを使用します。

```
sudo pcs resource restart haproxy-bundle
```

リソース名を確認するには、Pacemaker **status** コマンドを使用します。

```
sudo pcs status
...
* Container bundle set: haproxy-bundle [cluster.common.tag/openstack-haproxy:pcmklatest]:
  * haproxy-bundle-podman-0 (ocf::heartbeat:podman): Started rhosp13-controller-0
  * haproxy-bundle-podman-1 (ocf::heartbeat:podman): Started rhosp13-controller-1
  * haproxy-bundle-podman-2 (ocf::heartbeat:podman): Started rhosp13-controller-2
...
```

1.4.2. HAProxy の概要

HAProxy は Pacemaker に同様のルールを提供します。バックエンドサーバーでヘルスチェックを実行し、機能しているバックエンドサーバーに要求を転送します。すべてのコントローラーノードで HAProxy が実行している。

実行中の HAProxy のコピーは N 個ありますが、実際には1つのみで、要求に対するフィールドが設定されます。このアクティブな HAProxy インスタンスは Pacemaker によって管理されます。この手法により競合が発生するのを防ぎ、HAProxy の複数のコピーが複数のバックエンド間で要求の分散を調整で

きるようにします。Pacemaker は HAProxy が失敗したことを検知すると、フロントエンドの IP アドレスを別の HAProxy インスタンスに再割り当てします。その後、この HAProxy インスタンスは制御する HAProxy インスタンスになります。

1.5. CONFIGURATION SCRIPT の使用

フェデレーション認証を設定するには、長くて複雑なコマンドを実行する必要があります。このタスクを容易にし、反復性を確保するために、コマンドは **configure-federation** と呼ばれるシェルスクリプトに保存されます。**configure-federation** にステップの名前を渡すと、特定のステップを実行できます。使用できるコマンドの一覧を表示するには、**help** オプション (`-h` または `--help`) を使用します。



注記

スクリプトの内容に関する詳しい情報は、[6章configure-federation ファイル](#)を参照してください。

変数の置き換え後に実行されるコマンドを表示するには、以下のオプションを使用します。

-n

このオプションは、システムに変更せずに操作を標準出力に書き込むドライランモードを提供します。

-v

このオプションは、実行前にその操作を stdout に書き込む詳細モードを提供します。これはログインに役立ちます。

1.6. プロキシまたは SSL 用語の使用

プロキシの背後にある環境について、以下の主要な機能について検討してください。

- バックエンドサーバーには異なるホスト名が設定されるか、別のポートでリッスンするか、またはプロキシのフロントエンドにクライアントに表示されるプロトコルとは異なるプロトコルを使用する可能性があります。
サーバーが自己参照 URL を生成する際に問題が発生する場合があります。たとえば、サーバーが同じサーバー上の別の URL にクライアントをリダイレクトする場合などです。サーバーが生成する URL は、クライアントが確認できるパブリックアドレスおよびポートと一致させる必要があります。
- HTTP や HTTPS などの認証プロトコルは、特定のサーバー、ポート、およびセキュアなトランスポート用に要求がターゲットとなっていることを確認する必要があるため、ホスト、ポート、プロトコルの影響を受けやすくなります。プロキシはこの情報に干渉する可能性があります。
 - プロキシは、公開フロントエンドで受信する要求を変換してから、バックエンドのパブリック以外のサーバーにディスパッチします。
 - パブリック以外のバックエンドサーバーからの応答には、プロキシのパブリックフロントエンドからの応答であるかのように調整が必要になる場合があります。
この問題には、さまざまなアプローチがあります。SAML はホスト、ポート、およびプロトコル情報を区別し、高可用性プロキシ (HAProxy) の背後で SAML を設定しているため、これらの問題に対処する必要があります。そうしないと、設定が失敗する可能性が高くなります。

第2章 RED HAT IDENTITY MANAGEMENT の設定

以下の機能を使用して、フェデレーションされたユーザー管理で Red Hat OpenStack Platform を設定することができます。

- Red Hat Identity Management(IdM) は Red Hat OpenStack Platform の外部にあります。
- Red Hat IdM は、すべてのユーザーおよびグループ情報のソースです。
- Red Hat Single Signon (RH-SSO) は、ユーザーのフェデレーションに Red Hat IdM を使用するよう設定されています。

2.1. RH-SSO 用の IDM サービスアカウントの作成

匿名バインドを使用する場合、セキュリティ上の理由から十分な情報が Red Hat Single Sign-On (RH-SSO) に不可欠となる一部情報は非表示になっています。その結果、IdM LDAP サーバーにこの情報を照会するには、専用アカウントの形式で RH-SSO に適切な特権を指定する必要があります。

```
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
```



注記

上記の手順は、configure-federation スクリプトで実行できます (**\$./configure-federation create-ipa-service-account**)。

2.2. テストユーザーの作成

IdM でユーザーアカウントを作成してテストします。

手順

1. IdM でユーザー **jdoe** を作成します。

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

2. ユーザーにパスワードを割り当てます。

```
$ipa passwd jdoe
```

2.3. OPENSTACK ユーザー向けの IDM グループの作成

Keystone グループ **federated_users** にマッピングするには、IdM グループ **openstack-users** が必要です。テストユーザーをこのグループにマッピングします。

Red Hat Identity Management (IdM) で **openstack-users** グループを作成します。

手順

1. **openstack-users** グループが存在しないことを確認します。

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

2. openstack-users グループを IdM に追加します。

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

3. テストユーザーを **openstack-users** グループに追加します。

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

4. **openstack-users** グループが存在し、test ユーザーをメンバーとして持つことを確認します。

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

第3章 RED HAT SINGLE SIGN-ON の設定

Red Hat Single Sign-On (RH-SSO) はマルチテナンシーをサポートし、**レルム**を使用してテナント間の分離を可能にします。その結果、RH-SSO 操作は常にレルムのコンテキスト内で実行されます。レルムは手動で作成することも、RH-SSO サーバーで管理者権限がある場合は **keycloak-httpd-client-install** ツールを使用して作成することもできます。

前提条件

RH-SSO サーバーが完全にインストールされている必要があります。RH-SSO のインストールに関する詳細は、[Server installation and configuration guide](#) を参照してください。

以下に示すように、次の変数の定義が必要です。

<_RH_RHSSO_URL_>	The Red Hat Single Sign-On URL
<_FED_RHSSO_REALM_>	使用中の RH-SSO レルムを特定します。

3.1. RH-SSO レルムの設定

Red Hat Single Sign-On (RH-SSO) レルムが利用できる状態になったら、RH-SSO Web コンソールを使用して、IdM に対してユーザーフェデレーションのレルムを設定します。

手順

1. 左上隅のドロップダウンリストから、RH-SSO レルムを選択します。
2. **Configure** パネルから、**User federated** を選択します。
3. **User Federation** パネルの **Add provider** ドロップダウンリストから、**Idap** を選択します。
4. 以下のパラメーターの値を指定します。サイト固有の値はすべて、実際の環境に適した値に置き換えます。

プロパティ	Value
コンソール表示名	Red Hat IDM
編集モード	READ_ONLY
登録の同期	Off
Vendor	Red Hat Directory Server
ユーザー名 LDAP 属性	uid
RDN LDAP 属性	uid
UUID LDAP 属性	ipaUniqueID
ユーザーオブジェクトクラス	inetOrgPerson, organizationalPerson

プロパティ	Value
接続 URL	LDAPS://<_FED_IPA_HOST_>
ユーザー DN	cn=users,cn=accounts,<_FED_IPA_BASE_DN_>
認証タイプ	simple
バインド DN	uid=rhssso,cn=sysaccounts,cn=etc,<_FED_IPA_BASE_DN_>
バインド認証情報	<_FED_IPA_RHSSO_SERVICE_PASSWD_>

- Test connection および Test authentication ボタンを使用して、ユーザーのフェデレーションが機能していることを確認します。
- Save** をクリックして、新規ユーザーフェデレーションプロバイダーを保存します。
- 作成した Red Hat IdM ユーザーフェデレーションページの上にある **Mappers** タブをクリックします。
- デバイスマッパーを作成してユーザーグループ情報を取得します。ユーザーのグループメンバーシップは SAM アサーションを返します。グループメンバーシップを使用して OpenStack で認証を提供します。
- Mappers ページで **Create** をクリックします。
- Add user federation mapper** ページで、**Mapper Type** ドロップダウンリストから **group-ldap-mapper** を選択し、**Group Mapper** という名前を付けます。以下のパラメーターの値を指定します。サイト固有の値はすべて、実際の環境に適した値に置き換えます。

プロパティ	Value
LDAP グループ DN	cn=groups,cn=accounts,<_FED_IPA_BASE_DN_>
グループ名 LDAP 属性	cn
グループオブジェクトクラス	groupOfNames
メンバーシップ LDAP 属性	member
メンバーシップ属性タイプ	DN
Mode	READ_ONLY
ユーザーグループの取得ストラテジー	GET_GROUPS_FROM_USER_MEMBEROF_ATTRIBUTE

- Save** をクリックします。

3.2. SAML アサーションを使用したユーザー属性の追加

セキュリティーベンダー言語 (SAML) は、アイデンティティプロバイダー (IdP) とサービスプロバイダー (SP) 間のユーザー属性および承認認証情報の通信を可能にするオープン標準です。

Red Hat Single Sign-On (RH-SSO) を設定して、アサーションに必要な属性を返すことができます。OpenStack Identity サービスが SAML アサーションを受信すると、それらの属性を OpenStack ユーザーにマッピングします。IdP 属性を Identity サービスデータにマッピングするプロセスは、Federated Mapping と呼ばれます。詳しくは、「[マッピングファイルの作成および Keystone にアップロード](#)」をご覧ください。

以下のプロセスを使用して属性を SAML に追加します。

手順

1. RH-SSO 管理 Web コンソールで、左上隅のドロップダウンリストから `<_FED_RHSSO_REALM_>` を選択します。
2. **Configure** パネルから **Clients** を選択します。
3. `keycloak-httpd-client-install` が設定されたサービスプロバイダークライアントを選択します。SAML **EntityId** でクライアントを特定できます。
4. タブの水平一覧から Mappers タブを選択します。
5. Mappers パネルで **Create** または **Add Builtin** を選択し、プロトコルマッパーをクライアントに追加します。

他の属性を追加できますが、必要なのはユーザーがメンバーとなっているグループの一覧だけです。グループメンバーシップはユーザーの承認方法になります。

3.3. グループ情報の SAML アサーションへの追加

手順

1. Mappers パネルの **Create** ボタンをクリックします。
2. **Create Protocol Mapper** パネルで、Mapper tpe ドロップダウンリストから Group リストを選択します。
3. **Name** フィールドに名前として Group List を入力します。
4. Group 属性の **Name** フィールドに、グループを SAML 属性の名前として入力します。

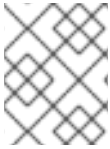


注記

これは、SAML アサーションに表示される属性の名前です。Keystone マッパーがマッピング宣言の **Remote** セクションで名前を検索すると、SAML 属性名を検索します。アサーションに渡す RH-SSO の属性を追加する場合は、SAML 属性名を指定します。RH-SSO プロトコルマッパーで名前を定義します。

5. SAML Attribute NameFormat パラメーターで、**Basic** を選択します。
6. Single Group Attribute トグルボックスで、**On** を選択します。

7. **Save** をクリックします。



注記

keycloak-httpd-client-install ツールを実行すると、プロセスではグループマッパーを追加します。

第4章 RED HAT OPENSTACK PLATFORM でのフェデレーションの設定

以下のノードには、割り当てられた完全修飾ドメイン名 (FQDN) が必要です。

- Dashboard (horizon) を実行しているホスト。
- Identity サービス (keystone) を実行中のホスト。本書では **\$FED_KEYSTONE_HOST** として参照されています。複数のホストが高可用性環境でサービスを実行するため、IP アドレスはホストアドレスではなく、サービスにバインドされる IP アドレスであることに注意してください。
- RH-SSO を実行するホスト。
- IdM を実行するホスト。

Red Hat OpenStack Platform director のデプロイメントでは DNS が設定されていないか、FQDN をノードに割り当てます。ただし、認証プロトコル (および TLS) には FQDN を使用する必要があります。

4.1. IP アドレスの取得

Red Hat OpenStack Platform には、ポート番号で区切られた、すべての OpenStack サービスに共通するパブリック IP アドレスが1つあります。オーバークラウドサービスのパブリック IP アドレスを確認するには、**openstack endpoint list** コマンドを使用します。

```
(overcloud) [stack@director ~]$ openstack endpoint list -c "Service Name" -c Interface -c URL | grep public
```

```
| swift      | public | http://10.0.0.101:8080/v1/AUTH_%(tenant_id)s |
| panko      | public | http://10.0.0.101:8977                      |
| nova       | public | http://10.0.0.101:8774/v2.1                  |
| glance     | public | http://10.0.0.101:9292                      |
| neutron    | public | http://10.0.0.101:9696                      |
| keystone    | public | http://10.0.0.101:5000                      |
| cinderv2    | public | http://10.0.0.101:8776/v2/%(tenant_id)s      |
| placement  | public | http://10.0.0.101:8778/placement             |
| cinderv3    | public | http://10.0.0.101:8776/v3/%(tenant_id)s      |
| heat       | public | http://10.0.0.101:8004/v1/%(tenant_id)s      |
| heat-cfn    | public | http://10.0.0.101:8000/v1                    |
| gnocchi     | public | http://10.0.0.101:8041                      |
| aodh        | public | http://10.0.0.101:8042                      |
| cinderv3    | public | http://10.0.0.101:8776/v3/%(tenant_id)s      |
```

4.2. ホスト変数の設定およびホストの命名

使用する IP アドレスとポートを判断する必要があります。この例では、IP アドレスは 10.0.0.101 で、ポートは 13000 です。

1. overcloudrc でこの値を確認します。

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

2. IP アドレスに完全修飾ドメイン名 (FQDN) を割り当て、これを **/etc/hosts** ファイルに書き込みます。以下の例では、overcloud.localdomain を使用しています。

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



注記

Red Hat OpenStack Platform director はオーバークラウドノードの host ファイルを設定しますが、参加する外部ホストにホストのエントリーを追加しなければならない場合があります。

3. fed_variables ファイルに \$FED_KEYSTONE_HOST および \$FED_KEYSTONE_HTTPS_PORT を設定します。この例では、同じ値を使用します。

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

Mellon は Identity サービス (keystone) をホストする Apache サーバーで実行するため、Mellon の host:port と keystone の host:port の値が一致する必要があります。



注記

コントローラーノードの1つで **hostname** コマンドを実行すると、出力が **controller-0.localdomain** のようになります。これは内部クラスター名ですが、そのパブリック名ではありません。代わりにパブリック IP アドレスを使用してください。

4.3. ヘルパーファイルのインストール

設定の一部としてヘルパーファイルをインストールする必要があります。

- 「[configuration script の使用](#)」の一部として作成した **configure-federation** および **fed_variables** ファイルを、**undercloud-0** の **stack** ホームディレクトリーにコピーします。

4.4. デプロイメント変数の設定

ファイル **fed_variables** には、フェデレーションのデプロイメントに固有の変数が含まれます。これらの変数は、本書および **configure-federation** ヘルパースクリプトで参照されます。サイト固有のフェデレーション変数には、**FED_** の接頭辞が付けられます。fed_variables のすべての **FED_** 変数に値が指定されていることを確認します。

4.5. ヘルパーファイルのコピー

続行するには、controller-0 に設定ファイルおよび変数ファイルが必要です。

- configure-federation と、編集した fed_variables を **undercloud-0** の **~/stack** ホームディレクトリーから **controller-0** の **~/heat-admin** ホームディレクトリーにコピーします。

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

**注記**

上記の手順は、configure-federation スクリプトで実行できます (`$./configure-federation copy-helper-to-controller`)。

4.6. 作業環境の初期化

1. アンダークラウドノードで、**stack** ユーザーとして **fed_deployment** ディレクトリーを作成します。この場所はファイル stash です。

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

**注記**

configure-federation スクリプトを使用して、直前の手順を実行できます。

```
$ ./configure-federation initialize
```

2. SSH を使用して **controller-0** に接続し `~/fed_deployment` ディレクトリーを **head-admin** ユーザーとして作成します。この場所はファイル stash です。

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

**注記**

configure-federation スクリプトを使用して、直前の手順を実行できます。**controller-0** ノードから以下を実行します。

```
$ ./configure-federation initialize
```

4.7. MOD_AUTH_MELLON のインストール

環境内の各コントローラーに **mod_auth_mellon** をインストールする必要があります。

- 各コントローラーで以下を実行します。

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

4.8. 各コントローラーへの RH-SSO FQDN の追加

すべてのコントローラーが完全修飾ドメイン名 (FQDN) で到達可能であることを確認します。

- mellon サービスは各コントローラーノードで実行され、RH-SSO IdP に接続します。RH-SSO IdP の FQDN が DNS 経由で解決できない場合には、**Heat Hosts** セクションの後に、すべてのコントローラーノードの `/etc/hosts` ファイルに FQDN を手動で追加します。

```
$ 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
```

4.9. コントローラーノードでの MELLON のインストールと設定

keycloak-httpd-client-install ツールは、**mod_auth_mellon** を設定し、RH-SSO IdP に対して認証するのに必要な多くの手順を実行します。mellon が実行するノードで **keycloak-httpd-client-install** ツールを実行します。以下の例では、mellon は、Identity サービス (keystone) を保護するオーバークラウドコントローラー上で実行します。



注記

Red Hat OpenStack Platform は、複数のオーバークラウドコントローラーノードを持つ高可用性デプロイメントで、それぞれ同一コピーを実行します。したがって、各コントローラーノードで mellon 設定を複製する必要があります。これを実行するには、controller-0 に mellon をインストールおよび設定し、**keycloak-httpd-client-install** ツールで作成した設定ファイルを tar ファイルに収集します。Object Storage (swift) を使用して各コントローラーにアーカイブをコピーし、アーカイブに含まれるファイルを解凍します。

- RH-SSO クライアントのインストールを実行します。

```
$ 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 \
-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"
```



注記

上記の手順は、configure-federation スクリプトで実行できます (**\$./configure-federation client-install**)。

クライアント RPM のインストール後に、以下のような出力が表示されるはずです。

```
[Step 1] Connect to Keycloak Server
[Step 2] Create Directories
[Step 3] Set up template environment
[Step 4] Set up Service Provider X509 Certificates
```

- [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

4.10. MELLON 設定の編集

IdP-assertion-to-Keystone マッピングフェーズ中は、グループを区切りリストにする必要があります。以下の手順に従い、属性に複数の値を受け取る場合に、区切りの1つの値に統合するように mellon を設定します。

手順

1. **v3_mellon_keycloak_openstack.conf** 設定ファイルを開いて編集します。

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

1. **MellonMergeEnvVars** パラメーターを <Location /v3> ブロックに追加します。

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

4.11. 生成された設定ファイルのアーカイブの作成

すべてのコントローラーノードで mellon 設定を複製するには、各コントローラーノードにインストールするファイルのアーカイブを作成します。アーカイブを **~/fed_deployment** サブディレクトリーに保存します。

1. 圧縮されたアーカイブを作成します。

```
mkdir fed_deployment && cd fed_deployment
tar -czvf rhsso_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
```



注記

configure-federation スクリプトを使用して、直前の手順を実行できます。


```
$ ./configure-federation create-sp-archive
```

4.12. MELLON 設定アーカイブの取得

- **undercloud-0** ノードで、作成したアーカイブを取得してファイルを展開し、後続のステップで必要に応じてデータにアクセスできるようにします。

```
$ 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
```

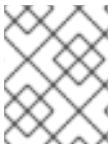


注記

上記の手順は、**configure-federation** スクリプトで実行できます (**\$./configure-federation fetch-sp-archive**)。

4.13. PUPPET が管理外の HTTPD ファイルを削除しない

デフォルトでは、Puppet Apache モジュールは、管理しない Apache 設定ディレクトリーのファイルをパージします。これにより、Apache が Puppet が実施する設定に対して動作しなくなります。ただし、これは HTTPD 設定ディレクトリーの **mellon** の手動設定と競合します。Apache Puppet の **apache::purge_configs** フラグはデフォルトで有効になっています。このフラグにより、Puppet は **mod_auth_mellon** RPM に属するファイルを削除することを指示します。Puppet は、**keycloak-httpd-client-install** が生成する設定ファイルも削除します。Puppet が **mellon** ファイルを制御するまでは、**apache::purge_configs** フラグを無効にします。



注記

apache::purge_configs フラグを無効にすると、コントローラーノードが脆弱性に対して開きます。Puppet が **mellon** の管理機能を追加したら、再度有効にします。

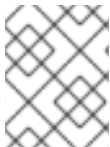
apache::purge_configs フラグを上書きするには、オーバーライドが含まれる Puppet ファイルを作成し、**overcloud_deploy.sh** スクリプトを実行する際に使用する Puppet ファイルの一覧にオーバーライドファイルを追加します。

1. **fed_deployment/puppet_override_apache.yaml** 環境ファイルを作成して、以下の内容を追加します。

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

2. **overcloud_deploy.sh** スクリプトの最後の環境ファイルとして **puppet_override_apache.yaml** を追加します。

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



注記

上記の手順は、**configure-federation** スクリプトで実行できます (`$./configure-federation puppet-override-apache`)。

4.14. IDENTITY サービス (KEYSTONE) でのフェデレーションの設定

Keystone ドメインには追加の設定が必要です。ただし、keystone Puppet モジュールが有効な場合は、この追加の設定ステップを実施することができます。

- Puppet YAML ファイルで、以下を追加します。

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

`/etc/keystone/keystone.conf` で以下の値を設定してフェデレーションを有効にします。

auth:methods

許可される認証方法の一覧。デフォルトでは、一覧は `['external', 'password', 'token', 'oauth1']` です。**mapped** メソッドを使用して SAML を有効にする必要があります。さらに、**external** メソッドも除外する必要があります。値を **password,token,oauth1,mapped** に設定します。

federation:trusted_dashboard

信頼済みダッシュボードホストの一覧。トークンを返すためのシングルサインオン要求を受け入れる前に、元のホストはこの一覧のメンバーである必要があります。異なる値に、この設定オプションを複数回使用することができます。これを Web ベースの SSO フローを使用するように設定する必要があります。このデプロイメントでは、値は

`https://$FED_KEYSTONE_HOST/dashboard/auth/websso/` です。Red Hat OpenStack Platform director は keystone と horizon の両方が同じホストに配置されているため、ホストは `$FED_KEYSTONE_HOST` になります。horizon を keystone とは異なるホスト上で実行する場合は、適宜調整する必要があります。

federation:sso_callback_template

Single Sign-On コールバックハンドラーとして使用される HTML ファイルへの絶対パス。このページでは、POST リクエストでトークンをフォームエンコードすることにより、ユーザーを Identity Service から信頼できるダッシュボードホストにリダイレクトします。デフォルト値は、ほとんどのデプロイメントで十分です。

federation:remote_id_attribute

アイデンティティプロバイダーのエンティティ ID の取得に使用される値。**mod_auth_mellon** については、**Mellon_IDP** を使用します。Mellon IDP ディレクティブを使用して、この値を mellon 設定ファイルに設定します。

- 以下の内容で `fed_deployment/puppet_override_keystone.yaml` ファイルを作成します。

```
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
```

- **overcloud_deploy.sh** スクリプトの最後に作成した環境ファイルを追加します。

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



注記

上記の手順は、**configure-federation** スクリプトで実行できます (`$./configure-federation puppet-override-keystone`)。

4.15. MELLON 設定アーカイブのデプロイ

- Object Storage (swift) アーティファクトを使用して、各コントローラーノードに mellon 設定ファイルをインストールします。

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



注記

上記の手順は、`configure-federation` スクリプトで実行できます (``./configure-federation deploy-mellon-configuration``)。

4.16. オーバークラウドの再デプロイ

- Puppet YAML 設定ファイルおよび Object Storage アーティファクトから変更を適用するには、`deploy` コマンドを実行します。

```
./overcloud_deploy.sh
```

重要: Puppet を再度実行してコントローラーノードに追加の変更を加えると、**overcloud_deploy.sh** スクリプトにより以前の設定が上書きされる可能性があります。この手順の後には、Puppet 設定を適用しないでください。これにより、オーバークラウドのコントローラーノード上の設定ファイルに対する手動編集が失われるのを防ぐことができます。

4.17. 各コントローラーで IDENTITY サービス (KEYSTONE) にプロキシの永続性を使用する

`mod_auth_mellon` がセッションを確立すると、複数のサーバーでその状態情報を共有することはできません。SAML で使用されるリダイレクト数が多いため、同じサーバーがすべてのトランザクションを処理する必要があります。そのため、HAProxy を、毎回各クライアントの要求を同じサーバーに転送するように設定する必要があります。

HAProxy が同じサーバーにクライアントをバインドする方法は 2 つあります。

Affinity

アプリケーション層下のレイヤーからの情報を使用して、クライアント要求を1つのサーバーに固定する場合は、アフィニティーを使用します。

永続性

アプリケーション層の情報がクライアントを1つのサーバーセッションにバインドする場合は、永続性を使用します。永続性はアフィニティーよりもはるかに正確です。永続を実装するには、以下の手順を使用します。

HAProxy クッキーディレクティブは、永続化のために Cookie およびそのパラメーターに名前を付けます。HAProxy サーバーのディレクティブには、クッキーの値をサーバー名に設定する `cookie` オプションがあります。受け取った要求にバックエンドサーバーを識別する `cookie` がない場合、HAProxy はその設定済みの分散アルゴリズムに基づいてサーバーを選択します。

手順

1. `/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg` 設定ファイルの `keystone_public` ブロックで永続性を有効にするには、以下の行を追加します。

```
cookie SERVERID insert indirect nocache
```

この設定は、**SERVERID** が永続 Cookie の名前です。

2. 各 **server** 行を編集し、追加オプションとして `cookie <server-name>` を追加します。

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

4.18. フェデレーションリソースの作成

アイデンティティプロバイダー (IdP) が使用する Identity サービス (keystone) のターゲット、ユーザー、およびグループを作成します。

手順

1. `source` コマンドでアンダークラウド上の **overcloudrc** ファイルを `stack` ユーザーとして読み込み、以下のコマンドを実行します。

```
$ 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_
```



注記

上記の手順は、**configure-federation** スクリプトで実行できます (`$./configure-federation create-federated-resources`)。

4.19. RED HAT OPENSTACK PLATFORM でのアイデンティティプロバイダーの作成

IdP は Identity サービス (keystone) に登録する必要があります。これにより、SAML アサーションの **entityID** と Identity サービスの IdP の名前間にバインディングが作成されます。

手順

1. IdP メタデータにある RH-SSO IdP の **entityID** を見つけます。IdP メタデータは、`/var/lib/config-data/puppet-generated/keystone/etc/httpd/saml2/v3_keycloak_$FED_RHSSO_REALM_idp_metadata.xml` ファイルに保存されます。また、IdP メタデータは、`fed_deployment/var/lib/config-data/puppet-generated/keystone/etc/httpd/saml2/v3_keycloak_$FED_RHSSO_REALM_idp_metadata.xml` ファイルにも確認することができます。
2. EntityID 属性の値に注意してください。これは、**<EntityDescriptor>** 要素内の IdP メタデータファイルにあります。この値には **\$FED_IDP_ENTITY_ID** 変数を割り当てます。
3. 変数 **\$FED_OPENSTACK_IDP_NAME** に割り当てられた IdP **rhsso** に名前を付けます。

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



注記

上記の手順は、**configure-federation** スクリプトで実行できます (`$./configure-federation openstack-create-idp`)。

4.20. マッピングファイルの作成および KEYSTONE にアップロード

Keystone は、IdP の SAML アサーションを keystone が理解できる形式に一致させるマッピングを実行します。このマッピングは keystone のマッピングエンジンによって実行され、IdP にバインドされる一連のマッピングに基づいています。

1. 以下の例で使用しているマッピングは、以下で説明します (導入部分で説明)。

```
[
  {
    "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"]
  }
}
]

```

このマッピングファイルには、1つのルールのみが含まれます。ルールは、**local** と **remote** の2つの部分に分類されます。マッピングエンジンは、1つが一致するまでルールの一覧を反復処理し、実行します。ルールは、ルールの **remote** 部分の **すべての** 条件が一致する場合にのみ一致とみなされます。この例では、**remote** 条件は以下を指定します。

1. アサーションには **MELLON_NAME_ID** という値が含まれている必要があります。
2. アサーションには **MELLON_groups** という名前の値が含まれ、グループ一覧の1つ以上のグループは **openstack-users** である必要があります。

ルールが一致する場合は、以下を実行します。

1. keystone **user** ユーザー名には、**MELLON_NAME_ID** からの値が割り当てられます。
2. ユーザーは、**federated_domain** ドメインの keystone グループ **federated_users** に割り当てられます。

要約すると、IdP がユーザーの認証に成功し、IdP がそのユーザーが **openstack-users** グループに所属していることをアサートすると、keystone は keystone の **federated_users** グループにバインドされている権限で OpenStack にアクセスできます。

4.20.1. マッピングを作成する

1. keystone でマッピングを作成するには、マッピングルールが含まれるファイルを作成してから keystone にアップロードし、参照名を提供します。**fed_deployment** ディレクトリー (例: **fed_deployment/mapping_\${FED_OPENSTACK_IDP_NAME}_saml2.json**) にマッピングファイルを作成し、名前 **\$FED_OPENSTACK_MAPPING_NAME** をマッピングルールに割り当てます。以下に例を示します。

```

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

```

注記

configure-federation スクリプトを使用して、上記の手順を2つの手順として実行することができます。

```

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

```

- **create-mapping** - マッピングファイルを作成します。
- **openstack-create-mapping** - ファイルのアップロードを実行します。

4.21. KEYSTONE フェデレーションプロトコルの作成

1. Keystone は、**Mapped** プロトコルを使用して IdP をマッピングにバインドします。このバインディングを確立するには、以下を実行します。

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



注記

上記の手順は、**configure-federation** スクリプトで実行できます (`$./configure-federation openstack-create-protocol`)。

4.22. KEYSTONE 設定を完全に修飾します。

1. 各コントローラーノードで `/var/lib/config-data/puppet-generated/keystone/etc/httpd/conf.d/10-keystone_wsgi_main.conf` を編集して、**VirtualHost** 内の **ServerName** ディレクティブに HTTPS スキーム、パブリックホスト名、およびパブリックポートが含まれていることを確認します。**UseCanonicalName** ディレクティブも有効にする必要があります。以下に例を示します。

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



注記

\$FED_ 変数をデプロイメントに固有の値に置き換えてください。

4.23. HORIZON がフェデレーションを使用するように設定する

1. 各コントローラーノードで `/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/local_settings` を編集して、以下の設定値が設定されていることを確認します。

```
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")),
)
```



注記

\$FED_ 変数をデプロイメントに固有の値に置き換えてください。

4.24. X-FORWARDED-PROTO HTTP ヘッダーを使用するように設定する

1. 各コントローラーノードで `/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/local_settings` を編集して、行のコメントを解除します。

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



注記

設定の変更を有効にするには、コンテナを再起動する必要があります。

第5章 トラブルシューティング

5.1. KEYSTONE マッピングルールのテスト

マッピングルールが予想通りに機能することを確認することが推奨されます。**keystone-manage** コマンドラインツールを使用すると、ファイルからも読み取られるアサーションデータに対して一連のマッピングルール(ファイルから読み込み)を実行できます。以下に例を示します。

1. **mapping_rules.json** ファイルには以下の内容が含まれます。

```
[
  {
    "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. **assertion_data.txt** ファイルには以下の内容が含まれます。

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

3. その場合は、以下のコマンドを実行します。

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

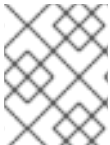
4. このマップされた結果が表示されるはずです。

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

```

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

```



注記

--engine-debug コマンドライン引数を含めることもできます。これにより、マッピングルールが評価される方法を説明する診断情報が出力されます。

5.2. KEYSTONE で受信されるアサート値を確認する

keystone が使用する マッピングされた アサーション値は環境変数として渡されます。これらの環境変数のダンプを取得するには、以下を実行します。

1. 以下の内容で、以下のテストスクリプトを **/var/www/cgi-bin/keystone/test** に作成します。

```

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. **WSGIScriptAlias** ディレクティブを一時的に変更して、**/var/lib/config-data/puppet-generated/keystone/etc/httpd/conf.d/10-keystone_wsgi_main.conf** ファイルを編集し、**test** スクリプトを実行するように設定します。

```

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

```

3. コンテナを再起動します。

```
podman restart keystone
```

4. ログインを試行し、スクリプトがダンプする情報を確認します。終了したら、**WSGIScriptAlias** ディレクティブを復元し、HTTPD サービスを再び再起動します。

5.3. SP と IDP の間で交換される SAML メッセージを確認します。

SAMLTracer Firefox アドオンは、SP と IdP の間で交換される SAML メッセージをキャプチャーし、表示するのに役立つツールです。

1. URL <https://addons.mozilla.org/en-US/firefox/addon/saml-tracer/> から **SAMLTracer** をインストールします。
2. Firefox メニューから **SAMLTracer** を有効にします。すべてのブラウザー要求が表示される **SAMLTracer** のポップアップウィンドウが表示されます。要求が SAML メッセージとして検出される場合は、特別な **SAML** アイコンが要求に追加されます。
3. Firefox ブラウザーから SSO ログインを開始します。
4. **SAMLTracer** ウィンドウで最初の **SAML** メッセージを見つけ、これをクリックします。ウィンドウで **SAML** タブを使用して、デコードされた SAML メッセージを表示します (ツールはメッセージの本文の暗号化されたコンテンツを復号化できません。暗号化されたコンテンツを表示する必要がある場合は、メタデータの暗号化を無効にする必要があります)。最初の SAML メッセージは、SP によって IdP に送信される **AuthnRequest** である必要があります。2 番目の SAML メッセージは、IdP によって送信されるアサーション応答である必要があります。SAML HTTP-Redirect プロファイルが使用されているため、アサーション応答は POST でラップされます。**SAML** タブをクリックして、アサーションの内容を表示します。

第6章 CONFIGURE-FEDERATION ファイル

```
#!/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=${1:-$overcloudrc_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
```

第7章 FED_VARIABLES ファイル

```
# 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"
```