Red Hat OpenStack Platform 15

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

Last Updated: 2022-09-15

Red Hat OpenStack Platform 15 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

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

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

第1章 概要

本ガイドでは、認証サービスに Red Hat Single Sign-On(RH-SSO)サーバーを使用して、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方法について説明します。

1.1. 運用ゴール

本書に従うことで、OpenStack デプロイメントの認証サービスは RH-SSO とフェデレーションされ、以下の特徴があります。

- フェデレーションは、SAML(Security Assertion Markup Language)を基にします。
- ID プロバイダー(IdP)は RH-SSO で、Red Hat OpenStack Platform デプロイメント外部にあります。
- RH-SSO IdP は、フェデレーションされたユーザーバッキングストアとして Red Hat Identity Management(IdM)を使用します。その結果、ユーザーおよびグループは IdM で管理され、RH-SSO は IdM に保存されているユーザーおよびグループ情報を参照します。
- IdM ユーザーは、IdM グループ **openstack-users** に追加されると、OpenStack へのアクセスが許可されます。
- OpenStack Keystone には **federated_users** という名前のグループがあります。**federated_users** グループのメンバーには **Member** ロールが割り当てられ、プロジェクトへのアクセス権限を付与します。
- フェデレーションされた認証プロセス時に、IdM グループの **openstack-users** のメンバーは OpenStack Group **federated_users** にマッピングされます。その結果、OpenStack にアクセスするために IdM ユーザーは **openstack-users** グループのメンバーである必要があります。ユーザーが IdM グループ **openstack-users** のメンバーでない場合は、認証に失敗します。

1.2. 想定条件

本書では、デプロイメントに関して、以下を前提としています。

- RH-SSO サーバーが存在し、サーバーに管理者権限を持っているか、RH-SSO 管理者がレルムを作成しており、そのレルムで管理者権限が付与されている。フェデレーションされた IdP は定義により外部にあるので、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。
- 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、ユーザーフェデレーションバッキングストアとして IdM を使用します。
- OpenStack デプロイメントは Red Hat OpenStack Platform director をベースにしています。
- Red Hat OpenStack Platform director オーバークラウドのインストールには、高可用性(HA)機能が使用されます。
- Red Hat OpenStack Platform director のオーバークラウドのみがフェデレーションを有効にし、アンダークラウドはフェデレーションされません。
- TLS による暗号化は、**すべて** の外部通信に使用されます。
- 全ノードには、完全修飾ドメイン名(FQDN)があります。

- 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。
- Pacemaker は、オーバークラウドサービスの一部（HAProxy を含む）の管理に使用されます。
- Red Hat OpenStack Platform director でオーバークラウドをデプロイしている。
- アンダークラウドおよびオーバークラウドノードに SSH 接続できる。
- 「 Keystone Federation Configuration Guide 」で説明されている例が続きます。
- **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作業します。
- **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操作します。

1.3. 前提条件

- RH-SSO サーバーが設定され、Red Hat OpenStack Platform director オーバークラウド外にある。
- IdM デプロイメントは、Red Hat OpenStack Platform director のオーバークラウドの外部にあります。
- Red Hat OpenStack Platform director でオーバークラウドをデプロイしている。

MOD_AUTH_MELLON を再インストールします。

mod_auth_mellon がコントローラーノードにインストールされていた場合は（コントローラーノードをインスタンス化するために使用されるベースイメージに含まれているため）、再インストールする必要があります。これは、Puppet が Apache モジュールを管理する方法が原因で、Puppet Apache クラスは Puppet の制御下では実行されない Apache 設定ファイルをすべて削除します。これらのファイルが削除されても Apache が起動しず、不明な Mellon ファイルに関するエラーが発生することに注意してください。この書き込み時に、**mod_auth_mellon** は Puppet の制御外に留まります。Puppet が Apache 設定ファイルを削除しないようにする方法は、「Puppet で管理対象外の HTTPD ファイルの削除の防止」 を参照してください。

以下のように、Puppet が **mod_auth_mellon** RPM に属するファイルが削除されているかどうかを確認するには、クエリーを実行して'mod_auth_mellon' パッケージを検証します。

```
$ rpm -qV mod_auth_mellon
missing   c /var/lib/config-data/puppet-generated/keystone/etc/httpd/conf.d/auth_mellon.conf
missing   c /var/lib/config-data/puppet-generated/keystone/etc/httpd/conf.modules.d/10-auth_mellon.conf
```

RPM がこれらの設定ファイルが存在しないことが示唆される場合、Puppet は削除されました。その後、ファイルを復元できます。

```
$ sudo dnf reinstall mod_auth_mellon
```

詳細は、BZ#1434875 および BZ#1497718 を参照してください。

1.4. OPENSTACK ノードへのアクセス

1. **root** ユーザーとして、OpenStack デプロイメントをホストするノードに SSH 接続します。以下に例を示します。

```
$ ssh root@xxx
```

2. アンダークラウドノードに SSH 接続します。

```
$ ssh undercloud-0
```

3. **stack** ユーザーになります。

```
$ su - stack
```

4. source コマンドでオーバークラウド設定を読み込み、必要な OpenStack 環境変数を有効にします。

```
$ source overcloudrc
```

注記

現在、Red Hat OpenStack Platform director は Keystone v2 API を使用するように Keystone を設定しますが、Keystone v3 API を使用します。本ガイドの後半に、**overcloudrc.v3** ファイルを作成します。これ以降は、**overcloudrc** ファイルの v3 バージョンを使用する必要があります。詳細は、「Keystone バージョン 3 API の使用」 を参照してください。

overcloudrc を読み込んだ場合は、**openstack** コマンドラインツールを使用してコマンドを実行できます。これは、オーバークラウドに対して動作します（現在、アンダークラウドノードにログインしている場合でも）。オーバークラウドノードの 1 つに直接アクセスする必要がある場合には、**heat-admin** ユーザーとして SSH を行うことができます。以下に例を示します。

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

1.5. 高可用性について

高可用性の詳細は、『 High Availability Deployment and Usage 』ガイドを参照してください。

- 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の IP アドレスはオーバークラウドプライベートであるため、外部には表示されません。これは、コントローラーノードで実行されているサービスが HAProxy バックエンドサーバーであるためです。コントローラーノードのセットには、1 つの公開されている IP アドレスがあります。これは HAProxy のフロントエンドです。パブリック IP アドレスのサービスに到達すると、HAProxy は要求を処理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、リソースが機能しなくなっ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可能性があります。また、Pacemaker は、リソースを正常に起動および停止する方法も認識します。

1.5.1. HAProxy の概要

HAProxy は、Pacemaker と同様に、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のみを転送するためです。すべてのコントローラーノードで実行中の HAProxy のコピーがあります。

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。このアプローチは、競合の発生を防ぐことができ、HAProxy の複数のコピーが複数のバックエンド間での要求の分散を調整できます。Pacemaker が HAProxy が失敗したことを検出すると、フロントエンドの IP アドレスが別の HAProxy インスタンスに再割り当てされ、制御される HAProxy インスタンスになります。高可用性を確保する場合には、高可用性と考えることができます。Pacemaker により予約される HAProxy インスタンスは実行中ですが、Pacemaker では、アクティブな HAProxy インスタンスへの接続のみがルーティングされるようにネットワークを設定しているので、受信接続は表示されません。

1.5.2. Pacemaker サービスの管理

Pacemaker が管理するサービスは、コントローラーノードの **systemctl** で管理することはできません。代わりに Pacemaker **pcs** command を使用します（例： **sudo pcs resource restart haproxy-clone** ）。Pacemaker status コマンドを使用してリソース名を確認できます(**sudo pcs status**)。これにより、以下のような結果が出力されます。

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

1.5.3. 設定スクリプトの使用

本ガイドの手順の多くは複雑なコマンドの実行を必要とします。そのため、そのタスクをより簡単に（また反復性を確保できるように）するために、すべてのコマンドが **configure-federation** と呼ばれるマスターシェルスクリプトに収集されました。各ステップは、**設定するステップの名前を渡すことで実行できます**。使用できるコマンドの一覧は、help オプション（**-h** または **--help**）を使用して確認できます。

注記

スクリプトについては、以下の場所にあります。 *6章configure-federation ファイル*

configure-federation スクリプトの実行時に、変数置換後のコマンドを正確に把握しておくと便利です。

- **-n** はドライランモードです。何も変更しません。代わりに、正確な操作が stdout に書き込まれます。
- **-v** は冗長モードです。実際の操作は、実行前の標準出力に書き込まれます。これはロギングに役立ちます。

1.5.4. サイト固有の値

本ガイドで使用されている特定の値はサイト固有なので、本書に直接これらのサイト固有の値を含めることが混乱し、これらのステップを複製しようとする人にとってエラーの原因になっている可能性があります。これに対応するため、本ガイドで参照されるサイト固有の値は変数形式です。変数名はドル記号(**$**)で始まり、all-caps で **FED_** のプレフィックスが付けられます。たとえば、RH-SSO サーバーへのアクセスに使用する URL は **$FED_RHSSO_URL**になります。

注記

変数ファイルは以下の場所にあります。 *7章fed_variables ファイル*

サイト固有の値は常に、set **-federation** スクリプトで使用される **$FED_** Site 固有の値を検索して特定できます。**fed_variables** に収集されます。このファイルは、実際のデプロイメントに合わせて編集する必要があります。

1.6. プロキシーまたは SSL ターミネーターの使用

サーバーがプロキシーの背後にある場合、その環境はクライアントがサーバーのパブリックアイデンティティーとして表示される環境とは異なります。バックエンドサーバーのホスト名が異なる場合があり、別のポートをリッスンするか、またはプロキシーのフロントエンドでクライアントが認識する内容とは異なるプロトコルを使用します。多くの Web アプリケーションでは、これは大きな問題ではありません。通常、ほとんどの問題は、サーバーが自己参照可能な URL を生成する必要がある場合に発生します（クライアントが同じサーバーの異なる URL にリダイレクトするためなど）。サーバーが生成する URL は、クライアントが表示するパブリックアドレスおよびポートと一致する必要があります。

認証プロトコルは、特にホスト、ポート、プロトコル（HTTP/HTTPS など）は特定のポートおよびセキュアなトランスポートで特定のサーバーでターゲットである可能性があるためです。プロキシーは、バックエンド内のパブリックでないサーバーにディスパッチする前にプロキシーによって公開されるフロントエンドで受信される要求を変換するため、プロキシーはこの重要な情報に干渉する可能性があります。同様に、パブリック以外のバックエンドサーバーからの応答は、プロキシーのパブリックフロントエンドからの応答がかのように表示されるように調整する必要がある場合もあります。

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、プロトコル情報に信頼されており、高可用性プロキシー(HAProxy)の背後で SAML を設定しているため、これらの問題に対応する必要があります。そうでないと、設定が失敗する可能性が高くなります（通常は暗号化方法ではありません）。

1.6.1. ホスト名およびポートに関する考慮事項

ホストおよびポートの詳細は、複数のコンテキストで使用されます。

- クライアントによって使用される URL のホストおよびポート。
- HTTP リクエストに挿入されるホスト HTTP ヘッダー（クライアント URL ホストから派生）
- クライアントが接続するフロントエンドプロキシーのホスト名。これは、プロキシーがリッスンする IP アドレスの FQDN です。
- クライアント要求を実際に処理するバックエンドサーバーのホストおよびポート。
- クライアント要求を実際に処理するサーバーの仮想ホストおよびポート。

これらの各値がどのように使用されるかを理解することが重要です。そうでないと、不正なホストおよびポートが使用されるリスクがあり、その結果、トランザクションに関与する参加者を検証できないため、認証プロトコルが失敗することがあります。

まずは、要求を処理するバックエンドサーバーを考慮することができます。これは、ホストとポートが評価され、ほとんどの問題が発生する場所です。

バックエンドサーバーは以下を認識する必要があります。

- 要求の URL（ホストとポートを含む）。
- 独自のホストおよびポート。

Apache は、命名のホストをサポートします。これにより、1 台のサーバーで複数のドメインをホストできます。たとえば、**example.com** で実行しているサーバーは、**example.com** と example - 2.com の両方に対して要求を処理する場合があり、これらは仮想ホスト名になります。Apache の仮想ホストは、server 設定ブロック内に設定されます。以下に例を示します。

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

Apache がリクエストを受信すると、**HOST** HTTP ヘッダーからホスト情報を収集し、仮想ホストのコレクションでホストを **ServerName** に一致しようとします。

ServerName ディレクティブは、サーバーがそれ自体を識別するために使用する要求スキーム、ホスト名、およびポートを定義します。**ServerName** ディレクティブの動作は、**UseCanonicalName** ディレクティブにより変更されます。**UseCanonicalName** が有効になっている場合、Apache は **ServerName** ディレクティブで指定されたホスト名とポートを使用して、サーバーの正規名を構築します。この名前はすべての自己参照 URL で使用され、CGI の **SERVER_NAME** および **SERVER_PORT** の値に使用されます。**UseCanonicalName** が **Off** の場合、Apache はクライアントが提供するホスト名とポート（ある場合）を使用して自己参照可能な URL を形成します。

ServerName にポートが指定されていない場合、サーバーは受信要求からのポートを使用します。信頼性と予測性を最適化するには、**ServerName** ディレクティブを使用して明示的なホスト名およびポートを指定する必要があります。**ServerName** が指定されていない場合、サーバーはまずオペレーティングシステムにシステムのホスト名について要求してホストの推測を試みます。これが失敗した場合は、システムで存在する IP アドレスの逆引き参照を実行します。その結果、サーバーがプロキシーの背後にあると誤ったホスト情報が生成されるため、**ServerName** ディレクティブの使用が必須になります。

Apache ServerName ドキュメントでは、サーバーがプロキシーの背後にある際に、**Server** name ディレクティブでスキーム、ホスト、およびポートを完全に指定する必要があります。

> サーバーは、リバースプロキシー、ロードバランサー、SSL オフロードアプライアンスなどの SSL を処理するデバイスの背後で実行される場合があります。この場合は、https:// スキームと、クライアントが ServerName ディレクティブで接続するポート番号を指定して、サーバーが正しい自己参照 URL を生成するようにします。

プロキシーが有効な場合、それらは **X-Forwarded-*** HTTP ヘッダーを使用して、要求が転送されたことを要求を処理するエンティティーや、元の値が転送前を認識できるようにします。Red Hat OpenStack Platform director の HAProxy 設定は、以下の設定を使用して、フロント接続で SSL/TLS を使用しているかどうかに基づいて、**X-Forwarded-Proto** HTTP ヘッダーを設定します。

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

さらに、Apache はこのヘッダーを解釈しないため、責任は別のコンポーネントに分類され、適切に処理されます。要求を処理するバックエンドサーバーの前に HAProxy が SSL を終了する状況では、Apache は拡張モジュール（**mellon**など）が要求のプロトコルスキームを要求する際に **X-Forwarded-Proto** HTTP ヘッダーが HTTPS に設定されているため、X-Forwarded-Proto HTTP ヘッダーが HTTPS

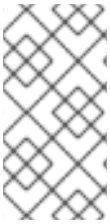
に設定されていることは無関係です。このため、**ServerName** ディレクティブに **scheme::///host:port** が含まれ、**UseCanonicalName** が有効になっている場合は、**mod_auth_mellon** などの Apache 拡張モジュールはプロキシーの背後で適切に機能しません。

Apache がプロキシーの背後でホストされる Web アプリケーションに関しては、転送されたヘッダーを処理する Web アプリケーションの（または Web アプリケーションフレームワーク）が責任となります。そのため、アプリケーションは、転送されたリクエストのプロトコルスキームを Apache 拡張モジュールとは異なる方法で処理します。Dashboard(horizon)は Django Web アプリケーションであるため、**X-Forwarded-Proto** ヘッダーを処理する責任があります。この問題は、認証中に horizon **が** 使用する元のクエリーパラメーターで発生します。Horizon は、認証を実行するために呼び出す keystone URL に元のクエリーパラメーターを追加します。**元の リソース にリダイレクトするために、horizon により元のパラメーターが使用されます。**

horizon が生成する元のパラメーターは、ファクト horizon が HTTPS を有効にして実行していても、https ではなく、スキームとして誤って指定する場合があります。これは、horizon が関数 build_absolute_uri（） を呼び出して元のパラメーターを形成するために発生します。build_absolute_url（） は最終的に Django によって実装されます。Django は、特別な設定ディレクティブを使用して X-Forwarded-Proto を処理するように強制できます。これは、Django secure-proxy-ssl-header ドキュメントで説明されています。

/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/local_settings でこの行をコメント解除して、この設定を有効にすることができます。

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



注記

Django はヘッダーの前に 「HTTP_」 を追加し、ハイフンをアンダースコアに変換することに注意してください。

コメント解除後、Origin パラメーターは HTTPS スキームを正しく使用します。ただし、ServerName ディレクティブに HTTPS スキームが含まれている場合でも、Django 呼び出し build_absolute_url（） は HTTPS スキームを使用しません。Django の場合は、SECURE_PROXY_SSL_HEADER オーバーライドを使用する必要があります。ServerName ディレクティブでスキームを指定すると動作しません。Apache 拡張モジュールと Web アプリケーションが転送された要求の要求スキームを異なる方法で処理することに注意してください。これには、ServerName と X-Forwarded-Proto HTTP ヘッダー技術の両方を使用する必要があります。

第2章 RED HAT IDENTITY MANAGEMENT の設定

以下の例では、IdM は OpenStack Red Hat OpenStack Platform director デプロイメントに外部に配置されており、すべてのユーザーおよびグループ情報のソースです。RH-SSO は、IdM をユーザーフェデレーション として使用するよう設定され、IdM に対して LDAP 検索を実行してユーザーおよびグループ情報を取得します。

2.1. RH-SSO の IDM サービスアカウントの作成

IdM は匿名バインドを許可しますが、セキュリティー上の理由から、情報があるものもあります。RH-SSO ユーザーフェデレーションでは、匿名バインド時にこの情報を受け入れるには必須となるものがあります。そのため、RH-SSO は必要な情報を正常にクエリーするのに十分な権限で IdM LDAP サーバーにバインドする必要があります。その結果、IdM で RH-SSO の専用のサービスアカウントを作成する必要があります。IdM は、これを実行するためのコマンドをネイティブで提供しませんが、ldapmodify コマンドを使用できます。以下に例を示します。

```
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
```

注記

configure-federation スクリプトを使用して上記の手順を実行できます。

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

2.2. テストユーザーの作成

IdM にテストユーザーアカウントも必要です。既存のユーザーを使用するか、新しいユーザーを作成することができます。本ガイドの例では、jdoe の uid で "John Doe" を使用しています。IdM で jdoe ユーザーを作成できます。

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

ユーザーにパスワードを割り当てます。

```
$ ipa passwd jdoe
```

2.3. OPENSTACK ユーザー用の IDM グループの作成

IdM で openstack-users グループを作成します。

1. **openstack-users グループがまだ存在していないことを確認してください。**

```
$ 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. **test 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 RH-SSO の設定

RH-SSO インストールプロセスは本ガイドの対象範囲外です。Red Hat OpenStack Platform director デプロイメントとは別に配置されているノードに RH-SSO がすでにインストールされていることを前提とします。

- RH-SSO URL は、$FED_RHSSO_URL 変数で識別されます。
- 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。本ガイドでは、サイト固有の変数 $FED_RHSSO_REALM を使用して、使用されている RH-SSO レルムを特定します。
- RH-SSO レルムは、事前に作成することができます（RH-SSO が IT グループで管理される場合と同様）、keycloak-httpd-client-install ツールは RH-SSO サーバーに管理者権限がある場合には作成することができます。

3.1. RH-SSO レルムの設定

RH-SSO レルムが利用可能になったら、RH-SSO Web コンソールを使用して、IdM に対するユーザーフェデレーション用にそのレルムを設定します。

1. 左上隅のドロップダウンリストから $FED_RHSSO_REALM を選択します。
2. 左側の Configure パネルから User Federation を選択します。
3. User Federation パネルの右上隅にある Add provider ... ドロップダウンリストから、ldap を選択します。
4. 以下のフィールドにこれらの値を入力し、$FED_ サイト固有の変数に置き換えてください。

プロパティー	値
コンソール表示名	Red Hat IDM

プロパティー	値
編集モード	READ_ONLY
登録の同期	Off
Vendor	Red Hat Directory Server
ユーザ名 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
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=rhsso,cn=sysaccounts,cn=etc,$FED_IPA_BASE_DN
バインド認証情報	$FED_IPA_RHSSO_SERVICE_PASSWD

5. **Test connection and Test authentication ボタンを使用して、ユーザーフェデレーションが機能していることを確認します。**

6. **User Federation パネルの下部にある Save をクリックして、新しいユーザーフェデレーションプロバイダーを保存します。**

7. **作成した Red Hat IDM ユーザーフェデレーションページの上部にある Mappers タブをクリックします。**

8. **ユーザーのグループ情報を取得するマッパーを作成します。つまり、ユーザーのグループメンバーシップが SAML アサーションで返されます。後でグループメンバーシップを使用して OpenStack で認証を行います。**

9. **Mappers ページの右上にある Create ボタンをクリックします。**

10. **Add user fe 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 という名前を指定します。以下のフィールドにこれらの値を入力し、$FED_ サイト固有の変数に置き換えてください。**

プロパティー	値
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

11. **Save をクリックします。**

3.2. SAML アサーションのユーザー属性の追加

SAML アサーションは、ユーザーにバインドされるプロパティー（ユーザーメタデータなど）に keystone に送信することができます（例： ユーザーメタデータ）。これらは SAML の 属性 と呼ばれます。アサーションで必要な属性を返すように RH-SSO を設定する必要があります。keystone が SAML アサーションを受け取ると、keystone がこれらの属性をユーザーのメタデータにマッピングし、keystone が処理できるようにします。IdP 属性を keystone データにマッピングするプロセスは Federated Mapping と呼ばれ、本ガイドで後ほど説明します（「マッピングファイルの作成および Keystone にアップロード」を参照）。

RH-SSO は、返された属性 プロトコルマッピング を追加するプロセスを呼び出します。プロトコルマッピングは、RH-SSO クライアントのプロパティーです（例： RH-SSO レルムに追加されたサービスプロバイダー(SP)など）。特定の属性を SAML に追加するプロセスは、同様のプロセスに従います。

RH-SSO 管理 Web コンソールで、以下を実行します。

1. 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. 左側の Configure パネルから Clients を選択します。

3. keycloak-httpd-client-install によって設定された SP クライアントを選択します。これは SAML EntityId によって識別されます。

4. クライアントパネルの上部にあるタブの水平リストから Mappers タブを選択します。

5. 右上の Mappers パネルには、Create および Add Builtin の 2 つのボタンがあります。このボタンのいずれかを使用して、プロトコルマッパーをクライアントに追加します。

必要な属性を追加できますが、この演習では、ユーザーが所属するグループの一覧のみが必要です（グループメンバーシップはユーザーの承認方法であるため）。

3.3. アサーションへのグループ情報の追加

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 type ドロップダウンリストから Group list を選択します。

3. Name フィールドに Group List を名前として入力します。

4. Group attribute name フィールドに SAML 属性の名前として groups を入力します。

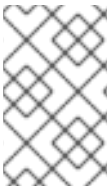
注記

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 セクションにある名前を検索する場合は、検索する SAML 属性名です。アサーションに渡すために RH-SSO に属性を追加するたびに、SAML 属性名を指定する必要があります。この名前は定義されている RH-SSO プロトコルマッパーです。

5. **SAML Attribute NameFormat フィールドで 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. **パネルの下部にある Save をクリックします。**



注記

keycloak-httpd-client-install は、実行時にグループマッパーを追加します。

第4章 OPENSTACK でのフェデレーションの設定

4.1. IP アドレスと FQDN 設定の特定

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。したがって、オーバークラウドの外部パブリック IP アドレスを確認する必要があります。オーバークラウドの IP アドレスが必要な点に注意してください。これは、オーバークラウドの個別ノード（controller- 0、controller -1 など）に割り当てられる IP アドレスと同じです。

IP アドレスは個別のノードではなく高可用性クラスターに割り当てられるので、オーバークラウドの外部パブリック IP アドレスが必要です。Pacemaker と HAProxy は連携して、単一 IP アドレスの外観を提供します。この IP アドレスは、クラスター内の任意のノードの個別 IP アドレスとは全く異なります。そのため、OpenStack サービスの IP アドレスについて適切な方法は、サービスがどのノードで実行されているかを検討するのではありませんが、クラスターがそのサービスに対してアドバタイズする効果的な IP アドレスを考慮するのではなく（VIP など）、OpenStack サービスの IP アドレスについて適切な方法を検討することができます。

4.1.1. IP アドレスの取得

正しい IP アドレスを決定するには、DNS を使用する代わりに名前を割り当てる必要があります。これには 2 つの方法があります。

1. Red Hat OpenStack Platform director は、全 OpenStack サービスに共通のパブリック IP アドレスを 1 つ使用し、それらのサービスをポート番号で 1 つのパブリック IP アドレスで

分離します。OpenStack クラスター内の 1 つのサービスのパブリック IP アドレスを把握している場合は、それらすべてを確認できます（サービスのポート番号も通知されません）。アンダークラウドの ~stack ホームディレクトリーにある overcloudrc ファイルの Keystone URL を確認することができます。以下に例を示します。

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

これにより、パブリックの keystone IP アドレスが 10.0.0.101 で、keystone がポート 13000 で利用可能であることが分かります。拡張により、その他の OpenStack サービスはすべて、固有のポート番号を持つ 10.0.0.101 の IP アドレスでも利用できます。

2.

ただし、IP アドレスとポート番号の情報を判断するためのより正確な方法は、各オーバークラウドノードにある HAProxy 設定ファイル(/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg)を調べることです。haproxy.cfg ファイルは、オーバークラウドの各コントローラーノード上の 同一 コピーです。これは、HAProxy に障害が発生した場合に、Pacemaker がクラスターに対して HAProxy を実行する責任を 1 つ割り当てるため、HAProxy の失敗 Pacemaker が別のオーバークラウドコントローラーを再割り当てしてしまうためです。どのコントローラーノードが 現在 HAProxy を実行しているかに関係なく、同じ動作状態でなければなりません。したがって、haproxy.cfg ファイルは同一でなければなりません。

a.

haproxy.cfg ファイルを検証するには、クラスターのコントローラーノードのいずれかに対して SSH を実行し、/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg を確認します。前述のように、どのコントローラーノードを選択しても問題ありません。

b.

haproxy.cfg ファイルはセクションに分割され、各セクションには listen ステートメントで始まり、その後にサービス名が続きます。サービスセクションの直後には bind ステートメントです。これらは フロントエンド IP アドレスであり、一部がパブリックで、その他はクラスターの内部になります。サーバー の行は、サービスが実際に実行されているバック IP アドレスです。クラスター内の各コントローラーノードに 1 つの サーバー 行が必要です。

c.

セクションの複数の バインド エントリーからサービスのパブリック IP アドレスとポートを確認するには、以下を実行します。

Red Hat OpenStack Platform director は、最初の バインド エントリーとしてパブリック IP アドレスを配置します。さらに、パブリック IP アドレスは TLS をサポートする必要があるため、バインド エントリーには ssl キーワードがあります。IP アドレスは、overstackrc ファイルにある OS_AUTH_URL に設定された IP アドレスにも一致する必要があります。たとえば、haproxy.cfg の keystone_public セクションの例を以下に示します。

```
listen keystone_public
```

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

d. **最初の bind 行には ssl キーワードがあり、IP アドレスは overstackrc ファイルにある OS_AUTH_URL と一致します。これにより、keystone がポート 13000 の IP アドレス 10.0.0.101 で公開されているようにすることができます。**

e. **2 つ目の bind 行はクラスターの内部であり、クラスターで実行されている他の OpenStack サービスで使用されます（TLS は公開されないため、TLS を使用しないことに注意してください）。**

f. **モード http 設定は、使用中のプロトコルが HTTP であることを示しています。これにより、HAProxy は他のタスクにおいて HTTP ヘッダーを検査できます。**

g. **X-Forwarded-Proto 行：**

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

これらの設定には特別な注意が必要であり、「ホスト変数の設定とホスト名」 で詳細を説明します。HTTP ヘッダー X-Forwarded-Proto がバックエンドサーバーによって設定され、表示されることを保証します。多くの場合、バックエンドサーバーはクライアントが HTTPS を使用しているかどうかを知っておく必要があります。ただし、HAProxy は TLS を終了するため、バックエンドサーバーは TLS 以外として接続を確認できます。X-Forwarded-Proto HTTP ヘッダーは、バックエンドサーバーが、要求が到達したプロトコルではなく、クライアントが実際に使用されたプロトコルを特定できるようにするメカニズムです。クライアントがプロトコルが HTTPS である悪意のある偽装を可能にするため、クライアントは X-Forwarded-Proto HTTP ヘッダーを送信できないことが 必須です。X-Forwarded-Proto HTTP ヘッダーはクライアントから受信時にプロキシーによって削除できます。または、プロキシーは強制的に設定するため、クライアントによる悪意のある使用を軽減することができます。このため、X-Forwarded-Proto は常に https または http のいずれかに設定されます。

HTTP ヘッダー X-Forwarded-For は、クライアントを追跡するために使用されます。これにより、バックエンドサーバーはプロキシーとして表示されるのではなく、要求

側のクライアントを識別できます。このオプションにより 、HTTP ヘッダーの X-Forwarded-For が要求に挿入されます。

```
option forwardfor
```

転送された proto、direct、ServerName などの詳細は、「ホスト変数の設定とホスト名」 を参照してください。

h. 以下の行では、パブリック IP アドレスで HTTPS のみが使用されていることを確認します。

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

この設定は、公開 IP アドレス（例： 10.0.0.101）で要求を受信し、HTTPS ではなかったかどうかを特定し、301 リダイレクト を実行し、スキームを HTTPS に設定します。

i. HTTP サーバー（Apache など）はリダイレクト目的で自己参照 URL を生成することがよくあります。このリダイレクト場所は正しいプロトコルを指定する必要がありますが、サーバーが TLS ターミネーターの背後にある場合は、リダイレクト URL は HTTPS ではなく HTTP である必要があります。この行は、HTTP スキームを使用する応答に Location ヘッダーが表示されるかどうかを特定し、HTTPS スキームを使用するように書き換えます。

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

4.1.2. ホスト変数の設定とホスト名

使用する IP アドレスとポートを確認する必要があります。以下の例では、IP アドレスは 10.0.0.101 で、ポートは 13000 です。

1. この値は overcloudrc で確認することができます。

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

2. また、haproxy.cfg ファイルの keystone_public セクションで、以下を実行します。

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

3. また、IP アドレスに FQDN を指定する必要があります。この例では、overcloud.localdomain を使用します。DNS が使用されていないため、IP アドレスは /etc/hosts ファイルに配置する必要があります。

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

注記

Red Hat OpenStack Platform director にはオーバークラウドノードの ホスト ファイルがすでに設定されているはずですが、参加する外部ホストにホストエントリーを追加する必要があります。

4. $FED_KEYSTONE_HOST および $FED_KEYSTONE_HTTPS_PORT は fed_variables ファイルで設定する必要があります。上記の例の値の使用：

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

Mellon は keystone をホストする Apache サーバーで実行しているため、Mellon host:port および keystone host:port の値は一致します。

注記

コントローラーノードのいずれ かでホスト名 を実行する場合は、controller-0.localdomain が使用される可能性がありますが、これはパブリック名ではなく 内部クラスター 名であることに注意してください。代わりに、パブリック IP アドレス を使用する必要があります。

4.2. UNDERCLOUD-0 への HELPER ファイルのインストール

1. configure-federation ファイルおよび 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。このファイルは、「設定スクリプトの使用」 の一部として作成します。

4.3. デプロイメント変数の設定

1. 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。各サイト固有のフェデレーション変数の前に FED_ が付けられ、変数として使用する場合は $

FED_ などの $ 変数構文を使用します。fed_variables のすべての FED_ 変数は必ず値が指定されていることを確認してください。

4.4. HELPER ファイルを UNDERCLOUD-0 から CONTROLLER-0 へコピー

1. **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。以下に例を示します。**

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

注記

configure-federation スクリプトを使用して上記の手順を実行できます： $ ./configure-federation copy-helper-to-controller

4.5. アンダークラウドでの作業環境の初期化

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 になります。以下に例を示します。**

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

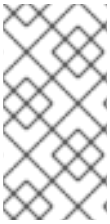
注記

configure-federation スクリプトを使用して、上記の手順を実行できます： $ ./configure-federation initialize

4.6. CONTROLLER-0 の作業環境を初期化します。

1. **アンダークラウドノードから、heat-admin ユーザーとし て controller-0 ノードに SSH 接続して、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 になります。以下に例を示します。**

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



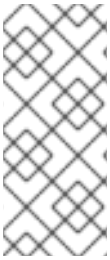
注記

configure-federation スクリプトを使用して上記の手順を実行できます。controller-0 ノードから： $ ./configure-federation initialize

4.7. 各コントローラーノードへの MOD_AUTH_MELLON のインストール

1. **アンダークラウドノードから、heat-admin ユーザーとして controller-n ノードに SSH 接続し、mod_auth_mellon をインストールします。以下に例を示します。**

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



注記

mod_auth_mellon がすでにコントローラーノードにインストールされている場合は、再度再インストールする必要がある場合があります。詳細は、mod_auth_mellon の注記を再インストール します。

注記

configure-federation スクリプトを使用して上記の手順を実行できます($ ./configure-federation install-mod-auth-mellon)。

4.8. KEYSTONE バージョン 3 API の使用

openstack コマンドラインクライアントを使用してオーバークラウドを管理する前に、特定のパラメーターを設定する必要があります。通常、シェルセッションで rc ファイルを取得します。これにより、必要な環境変数が設定されます。*Red Hat OpenStack Platform director は、この目的のために undercloud-0 ノードに* stack *ユーザーのホームディレクトリーに overcloudrc ファイルを作成します。デフォルトでは、overcloudrc ファイルは keystone API の v2 バージョンを使用するように設定されていますが、フェデレーションでは v3 keystone API を使用する必要があります。したがって、v3 keystone API を使用する新しい* rc *ファイルを作成する必要があります。

1. ***以下に例を示します。***

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

2. *以下の内容を overcloudrc.v3 に書き込みます。*

```
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
```

注記

configure-federation スクリプトを使用して上記の手順を実行できます($ ./configure-federation create-v3-rcfile)。

3. *これ以降は、overcloudrc.v3 ファイルを使用するオーバークラウドと連携します。*

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

4.9. 各コントローラーへの RH-SSO FQDN の追加

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 に接続するように設定されます。

1. *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）全コントローラーノードの /etc/hosts ファイルに FQDN を手動で追加する必要があります。*

```
$ 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
```

4.10. コントローラーノードへの MELLON のインストールおよび設定

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。keycloak-httpd-client-install ツールは、mellon が実行されるノードで実行する必要があります。この例では、mellon が Keystone を保護するオーバークラウドコントローラーで実行します。

これは高可用性デプロイメントであるため、それぞれ同一コピーを実行するオーバークラウドのコントローラーノードが複数ある点に注意してください。そのため、mellon 設定は各コントローラーノードに複製する必要があります。これは、controller-0 に mellon をインストールして設定し、次にアーカイブ（tar ファイルなど）に作成した keycloak-httpd-client-install ツールの全設定ファイルを収集し、swift がアーカイブを各コントローラーにコピーし、そこでファイルをアーカイブ解除するようにすることで、これを行います。

1. *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/rhsso/protocols/mapped/websso" \
 -l "/v3/OS-FEDERATION/identity_providers/rhsso/protocols/mapped/auth"
```

注記

configure-federation スクリプトを使用して、上記の手順を実行できます：
$ ./configure-federation client-install

2. *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 Certificiates
[Step  5] Build Mellon httpd config file
[Step  6] Build Mellon SP metadata file
[Step  7] Query realms from Keycloak server
[Step  8] Create realm on Keycloak server
[Step  9] Query realm clients from Keycloak server
```

```
[Step 10] Get new initial access token
[Step 11] Creating new client using registration service
[Step 12] Enable saml.force.post.binding
[Step 13] Add group attribute mapper to client
[Step 14] Add Redirect URIs to client
[Step 15] Retrieve IdP metadata from Keycloak server
[Step 16] Completed Successfully
```

4.11. MELLON 設定の編集

追加の **mellon** 設定がデプロイメントに必要です。**IdP-assertion-to-Keystone** マッピングフェーズでグループの一覧を使用すると、**keystone** マッピングエンジンはリストが特定の形式（セミコロン(;)で区切られた **1** つの値）にあることを想定しています。そのため、属性に複数の値を受信する場合は、複数の属性をセミコロンで区切られた項目で **1** つの値に統合するように **mellon** を設定する必要があります。この **mellon** ディレクティブは、以下に対応します。

```
MellonMergeEnvVars On ";"
```

1. デプロイメントでこの設定を設定するには、以下を実行します。

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

2. **< Location /v3>** ブロック を見つけ、その行に行を追加します。以下に例を示します。

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

4.12. 生成された設定ファイルのアーカイブの作成

mellon 設定はすべてのコントローラーノードで複製する必要があるため、各コントローラーノードに同じファイルの内容をインストールできるようにするファイルのアーカイブを作成します。アーカイブは **~heat-admin/fed_deployment** サブディレクトリーに保存されます。

1. 圧縮された **tar** アーカイブを作成します。

```
$ mkdir fed_deployment
$ tar -cvzf 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上記の手順を実行できます($ ./configure-federation create-sp-archive)。

4.13. MELLON 設定アーカイブの取得

1. *後続の手順でデータの一部にアクセスする必要があるため、undercloud-0 ノードで作成してファイルを取得し、ファイルを展開します（例： RH-SSO IdP の entityID ）。*

```
$ 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
```

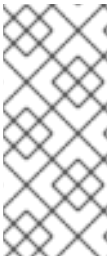
注記

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）。

4.14. PUPPET で管理対象外の HTTPD ファイルの削除の防止

デフォルトでは、Puppet Apache モジュールは管理していない Apache 設定ディレクトリー内のファイルをパージします。これは、Apache が Puppet で実施した設定以外の方法で動作しなくなるため、妥当な措置と見なされます。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 apache::purge_configs フラグを有効にすると（これはデフォルトで）、mod_auth_mellon RPM のインストール時に mod_auth_mellon RPM に属するファイルを Puppet から削除します。また、実行時に keycloak-httpd-client-install によって生成された設定ファイルも削除されます。mellon ファイルが Puppet コントロール下にあるまで、apache::purge_configs フラグを無効にする必要があります。

また、前の実行で mod_auth_mellon 設定ファイルがすでに削除されているかどうかを確認する必要がある場合があります。詳細は、「Reinstall mod_auth_mellon 」を参照してください。



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かれます。Puppet が mellon の管理のサポートを追加する際には、再度有効にしないでください。

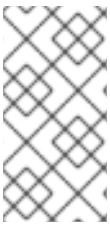
apache::purge_configs フラグを上書きするには、オーバーライドが含まれる Puppet ファイルを作成し、overcloud_deploy.sh の実行時に使用する Puppet ファイルの一覧にオーバーライドファイルを追加します。

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

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

2. overcloud_deploy.sh スクリプトの最後付近にファイルを追加します。これは、last -e 引数である必要があります。以下に例を示します。

```
-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）。

4.15. フェデレーションのための KEYSTONE の設定

本ガイドでは、追加の設定が必要な keystone ドメインを使用しています。有効な場合には、keystone Puppet モジュールはこの追加の設定ステップを実施することができます。

1. Puppet YAML ファイルの 1 つに、以下を追加します。

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

フェデレーションを有効にするには、/etc/keystone/keystone.conf で追加の値を設定する必要があ

ります。

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

これらの設定とその推奨値の説明：

- *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。マッピングされた方法を使用して SAML を有効にする必要があります。したがって、この値は external,password,token,oauth1, mapped でなければなりません。*
- *fe:trusted_dashboard: 信頼された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。この設定オプションは、複数の値に対して繰り返すことができます。これは、web ベースの SSO フローを使用するために設定する必要があります。このデプロイメントの場合には、値 https://$FED_KEYSTONE_HOST/dashboard/auth/websso/host は $FED_KEYSTONE_HOST であることに注意してください。これは、Red Hat OpenStack Platform director が同じホスト上に keystone と horizon の両方を同じホスト上に共存するので、このホストが $FED_KEYSTONE_FED_KEYSTONE_HOST/dashboard/auth/websso/HOST であることに注意してください。horizon が keystone とは別のホストで実行している場合は、それに応じて調整する必要があります。*
- *fe:sso_callback_template: シングルサインオン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。このページは、POST 要求でトークンをエンコードすることで、keystone からのユーザーを信頼済みのダッシュボードホストにリダイレクトすることが想定されます。Keystone のデフォルト値は、ほとんどのデプロイメントでは十分です(/etc/keystone/sso_callback_template.html)。*
- *federation:remote_id_attribute: ID プロバイダーのエンティティー ID を取得するために使用される値。mod_auth_mellon には MELLON_IDP を使用します。これは、MellonIdP IDP ディレクティブを使用して mellon 設定ファイルに設定されることに注意してください。*

1. *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
```

2. *overcloud_deploy.sh スクリプトの最後に、作成したファイルを追加します。これは、last -e 引数である必要があります。以下に例を示します。*

```
-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）。

4.16. MELLON 設定アーカイブのデプロイ

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。以下に例を示します。

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

注記

configure-federation スクリプトを使用して上記の手順を実行できます($ ./configure-federation deploy-mellon-configuration)。

4.17. オーバークラウドの再デプロイ

先の手順で、Puppet YAML 設定ファイルおよび swift アーティファクトに変更を加えました。これらの変更は、以下のコマンドを使用して適用できるようになりました。

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

注記

後のステップでは、オーバークラウドのコントローラーノードに他の設定変更が行われます。overcloud_deploy.sh スクリプトを使用して Puppet を再実行すると、これらの変更の一部を上書きする可能性があります。オーバークラウドコントローラーノードの設定ファイルに加えられた手動編集が失われないように、この時点で Puppet 設定を適用しないでください。

4.18. 各コントローラー上の KEYSTONE のプロキシー永続性の使用

高可用性を確保すると、複数のバックエンドサーバーの 1 つでリクエストのフィールドが表示されることが期待されます。SAML で使用されるリダイレクトの数と、これらのリダイレクトのそれぞれのリダイレクトには状態情報が必要になるため、同じサーバーがすべてのトランザクションを処理することが重要です。さらに、mod_auth_mellon によりセッションが確立されます。現在、mod_auth_mellon は複数のサーバーで状態情報を共有できないため、常にクライアントから同じサーバーに要求を送信するように HAProxy を設定する必要があります。

HAProxy は、アフィニティーまたは永続性のいずれかを使用してクライアントを同じサーバーにバインドできます。この記事では、HAProxy Sticky Sessions で役に立つ背景情報を提供します。

永続性 と アフィニティー の違いは、アプリケーションレイヤーの下のレイヤーからの情報がクライアント要求を 1 つのサーバーに固定する場合に使用されることです。永続性は、アプリケーションレイヤー情報がクライアントを 1 つのサーバーのスティッキーセッションにバインドする場合に使用されます。アフィニティーに対する永続性の主な利点は、より正確である点です。

永続性は Cookie を使用して実装されます。HAProxy cookie ディレクティブは、永続性に使用されるクッキーに名前を付け、その使用を制御するパラメーターと共にこれを使用します。HAProxy サーバー ディレクティブには Cookie オプションがあり、これはサーバー名に設定する必要があります。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。HAProxy は、Cookie が応答で選択されたサーバーの名前に設定されていることを確認します。受信要求にバックエンドサーバーを識別する cookie がある場合、HAProxy は要求を処理するサーバーを自動的に選択します。

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
```

この設定には、SERVERID が永続クッキーの名前になります。

2. 次に、各 サーバー の行を編集し、Cookie <server-name> を追加オプションとし て追加する必要があります。以下に例を示します。

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

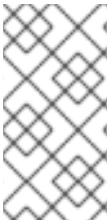
明確にするために、server ディレクティブの他の部分は省略されています。

4.19. FEDERATED リソースの作成

「keystone グループの作成」セクションに記載のフェデレーション例に従い、keystone フェデレーションドキュメントのロールを割り当てると、紹介する場合があります。

1. （overcloudrc.v3 ファイルを読み込んだ後）アンダークラウドノードで、stack ユーザーとして以下の手順を実施します。

```
$ 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）。

4.20. OPENSTACK でのアイデンティティープロバイダーの作成

IdP を 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 keystone の IdP の名前の間にバインディングが作成されます。

RH-SSO IdP の entityID を特定する必要があります。この値は、keycloak-httpd-client-install の実行時に取得された IdP メタデータにあります。IdP メタデータは、/var/lib/config-data/puppet-generated/keystone/etc/httpd/saml2/v3_keycloak_$FED_RHSSO_REALM_idp_metadata.xml ファイルに保存されます。先のステップでは、mellon 設定アーカイブを取得して fed_deployment 作業エリアに抽出しました。これにより、fed_deployment/var/lib/config-data/puppet-generated/keystone/etc/httpd/saml2/v3_keycloak_$FED_RHSSO_REALM_idp_metadata.xml に IdP メタ データを確認できます。IdP メタデータファイルでは、entityID 属性を持つ <EntityDescriptor > 要素があります。entityID 属性の値が必要です。たとえば、本書では $FED_IDP_ENTITY_ID 変数に保存されていることを仮定します。変数 $FED_OPENSTACK_IDP_NAME に割り当てられた IdP rhsso の名前を付けることができます。以下に例を示します。

```
$ 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）。

4.21. 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。*

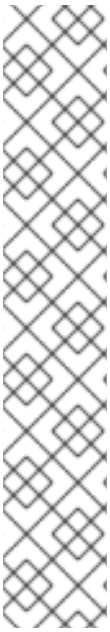
2. *ユーザーは Default ドメインの keystone グループ federated_users に割り当てられます。*

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。

4.21.1. 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。

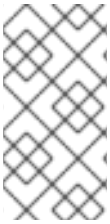
```
$ ./configure-federation create-mapping
$ ./configure-federation openstack-create-mapping
```

- *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。*

4.22. 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）。

4.23. 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。

4.24. 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")),
)
```

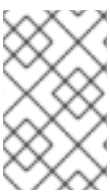
注記

$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。

4.25. 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、コンテナー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。

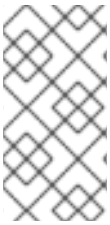
```
MELLON_NAME_ID: 'G-90eb44bc-06dc-4a90-aa6e-fb2aa5d5b0de
MELLON_groups: openstack-users;ipausers
```

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. **httpd を再起動します。**

```
systemctl restart httpd
```

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
```

```
            $cmd
            status=$?
            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://$FED_IPA_HOST"
    local dir_mgr_dn="cn=Directory Manager"
    local service_name="rhsso"
    local service_dn="uid=$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/rhsso/protocols/mapped/websso" \
    -l "/v3/OS-FEDERATION/identity_providers/rhsso/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 patttern 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 entitID 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
```

```
        (>&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
    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
    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="rhsso_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="rhsso-passwd"

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

# RH-SSO server FQDN
FED_RHSSO_FQDN="jdennis-rhsso-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="rhsso"

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