Adding federated access to OpenStack

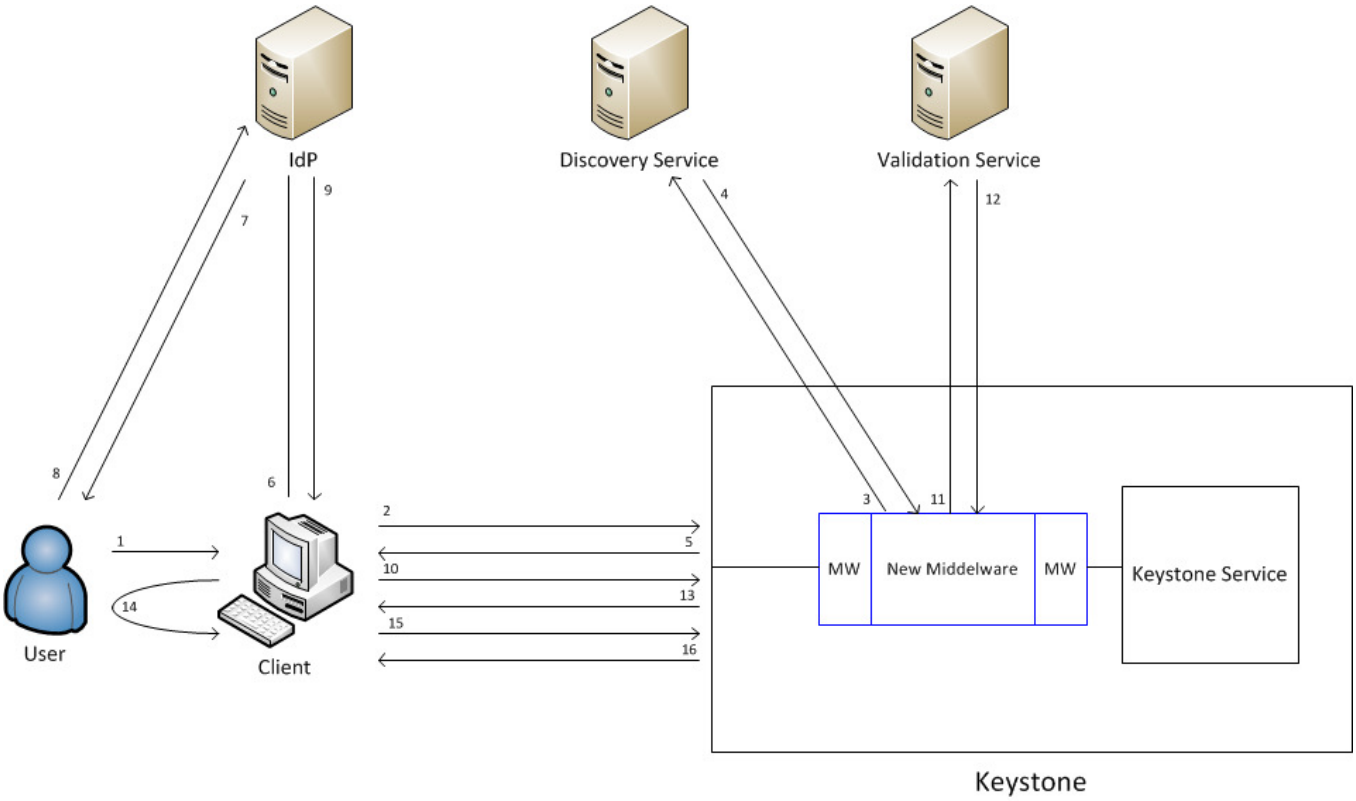
Versioning

Version	Date	Comment	Author
0.1	09/07/2012	Initial creation of the document.	Damien Germonville, Yann Fouillat
0.2	10/07/2012	Comments	David
0.3	13/07/2012	Comments and updates	Damien, Yann
0.4	16 July 2012	Significant edits on data flows	David
0.5	18/07/2012	Workflow updated	Damien, Yann
0.6	18 July 2012	Minor edits	David
0.7	18/07/2012	Review previous version, add a comment	Damien, Yann
0.8	19 July	Corrected inconsistencies	David
0.9	27 July 2012	Correct minor inconsistencies	David
0.91	31/07/2012	Minor update	Yann, Damien
1.0	31 July 2012	First public release	David

Introduction

In order to bring federated access to OpenStack, we will create a new middleware component responsible for the different federation services we need: Discovery and Credential Validation. This middleware component will be inserted in the Keystone pipeline. It will catch and process the incoming requests and the responses from and to the client, and will talk to the different federation services. This new design requires a rework of the clients if they want to use these features. As the different OpenStack clients (Swift, Nova etc.) have different designs, it will be easier to create a library [1] that can be used by them to simplify the implementation.

Workflow



It is assumed that either the client is configured with the address of keystone, or the user enters this along with his login credentials

1. The user types in the realm[1] he wants to use and optionally his tenant friendly name. (Note. a client function is available for obtaining the set of available realms [1]).
2. The wanted realm is sent to Keystone.
3. The discovery middleware component asks the discovery service for the endpoint of the Identity Provider the client should be redirected to in order to authenticate. Note that every different IdP scheme (Shibboleth, Liberty Alliance, OpenID, Oath etc.) may have a different discovery mechanism, so these components (discovery middleware and service) need to be replaceable. However the API to the discovery middleware should be fixed[2].
4. The discovery service replies to the discovery middleware component telling it the endpoint of the IdP the user has to use.
5. The endpoint of the IdP to use is sent back to the client, along with a correctly formatted IdP request message for the list of required attributes (the request blob). Note that in the simple

[1] Realm is the term the user knows for his Identity Provider domain. It will typically be a domain name. For example, the user might know the realm kent.ac.uk, whereas his IdP might be located at https://idp4.kent.ac.uk/simplesaml/module.php

[2] In the first proof of concept implementation, there won't be a discovery service, only a discovery middleware component that will have one fixed inbuilt value that it will always return – that of our local Shibboleth IdP.

case the list of attributes will be * (meaning All). Note that the format of the IdP request message (the request blob) will vary according to the IdP scheme in use[3].

6. The client opens up the user's default browser in a new window and passes it an Https get request to the IdP's endpoint, containing the "request blob" (Secure Get on login URL). Note that by instantiating the user's browser, the client code is independent of the authentication mechanism actually used by the IdP, since the authentication exchange will be between the user's browser and the IdP.
7. The login page is sent directly to the user's default browser.
8. The user logs in to the IdP via his browser window.
9. Upon successful authentication, the IdP sends the "response blob" (e.g. SAML authn and attribute assertions) to the client's browser in the form of a redirect message to the Keystone service (e.g. Form POST). Note that the metadata for Keystone tells the IdP to redirect it to the alternative SSL port on the local host, so the browser does this. The client polls this port until it is able to pick up the response message, then it displays a message to user indicating that they can now close the browser.
10. The client sends the response blob to Keystone.
11. The validation middleware sends the response blob to the credential validation service.
12. The credential validation service validates the IdP's assertions and returns the set of user attributes to the validation middleware. The middleware looks up the user in the Keystone database and is returned his set of tenants. Note that every user should be uniquely identified by one attribute, which is configured into the middleware [2]. If the user exists and the tenant ID was provided and it belongs to the user, then go to 16, otherwise
13. The validation middleware creates new tenants for each one that the user is entitled to, but does not yet have (according to the middleware configuration file [2]) and then sends the "unscoped" authentication token to the client along with the list of tenants the user has access to.
14. The user is asked to choose a tenant.
15. The chosen tenant ID is sent to Keystone along with the "unscoped" authentication token.
16. Keystone sends back the "scoped" authentication token associated with this tenant along with the set of services where it can be used.
17. The client can now use this scoped authentication token as before.

The services

- Discovery: This service is responsible for finding the endpoint of the right Identity Provider for the realm of the user requesting authentication. The client sends the user's realm which is forwarded to the discovery service by the corresponding middleware. The discovery service looks up the realm, then returns the endpoint of the realm's Identity Provider to the middleware, which returns it to the client, along with the set of attributes that the Keystone

[3] In the first proof of concept implementation this will be a SAML authn and attribute request signed by the discovery middleware component of the Keystone service.

service requires, in a correctly formatted IdP request message. Note. The middleware configuration details [2] say which attributes are needed to access which Keystone service. If keystone supports multiple services, then the union of all the attributes may be sent to the IdP.

- Credential Validation: This service is responsible for validating the assertions sent by the IdP (via the client). Once the IdP's assertions have been validated, the middleware taking care of the credential validation service will generate an "unscoped" token and send it back to the client.

References

[1] "Open Stack Client Connection API". Yann, Damien and David. V1.11. July 31, 2012

[2] "Open Stack Federated Middleware Services". Matteo, Kristy and David. V0.93 July 27, 2012