Keystone Identity Service in Openstack

August 2015

Author:
Paweł Pamuła
pawel.pamula@gmail.com

Supervisor:
Marek Denis

CERN Openlab Summer Student Report 2015

Project Specification

CERN runs a large scale OpenStack based cloud on over 3,200 hypervisors to provide computing resources for users on demand. Over 1 million virtual machines have been created during the past 18 months. As part of the project with Rackspace to enable users to be able to use multiple clouds in a transparent fashion, OpenStack tools need to be enhanced to support these new use cases.

The project involves enhancing Keystone Identity Service, which interacts with federated clouds, working with the OpenStack community and CERN administrators to define the right solutions along with implementation and testing in the CERN environment.

Abstract

The aim of this report is to describe and document the configuration steps and development process of the Openlab Summer Student project. The report explains basic ideas of cloud computing. Furthermore, it introduces Openstack as a software for deploying private clouds. Openstack architecture is further explained. The focus is put on Keystone Identity Service and its architecture and role that it serves in the whole Openstack environment.

Moreover, federation as a mechanism for establishing trusts between identity providers and OpenStack clouds is described. Eventually, further details are provided on token revocation mechanism. The last part describes the means for testing and debugging Openstack deployments.

Table of Contents

1 Introduction

1.1 Cloud Computing

Cloud computing is an important technology which enables a ubiquitous access to a shared pool of both hardware and software resources (e.g., networks, servers, storage, services, and applications). The aforementioned resources can be dynamically provisioned, released and redistributed with minimal effort or human interaction. According to the NIST, the Cloud has five essential characteristics.

- On-demand self-service: a user can be provided with computing resources without interaction with each service provider
- Broad network access: resources are accessible over the network and implement standardised protocols of communications
- Resource pooling: the provider serves multiple tenants, who are given access to virtual resources and have no control over their physical location. The user can be granted the ability to control the location at a higher level of abstraction.
- Rapid elasticity: the resources can easily scale outward and inward, according to the users' needs.

Measured service: the resources can be effectively monitored, controlled and measured in terms of performance at different levels of abstraction, appropriate to the type of service (e.g. storage, bandwidth).

1.2 Service models

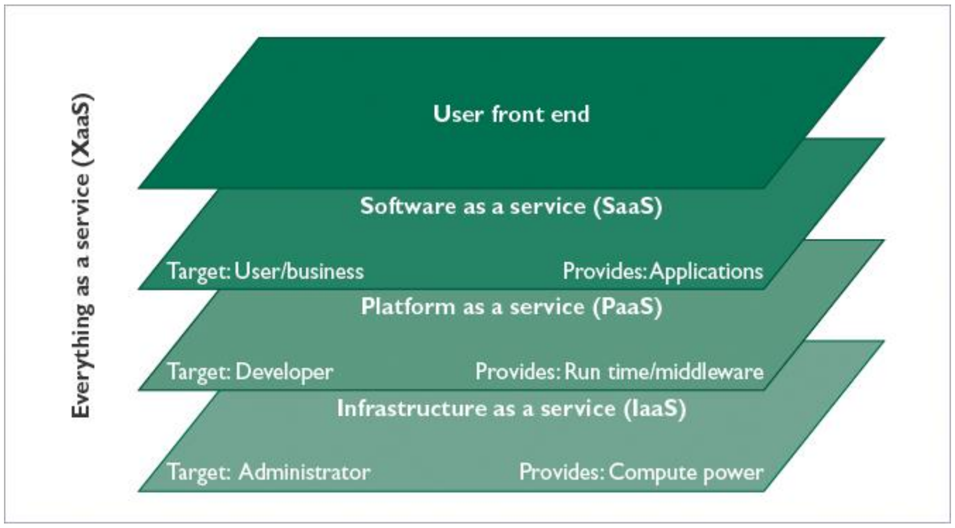
Everything as a service (XaaS)
User front end
Software as a service (SaaS)
Target: User/business
Provides: Applications
Platform as a service (PaaS)
Target: Developer
Provides: Run time/middleware
Infrastructure as a service (IaaS)
Target: Administrator
Provides: Compute power

Figure 1. Service models in cloud computing

Cloud computing architectures can be divided into three main models:

a) Infrastructure as a service (IaaS): the most basic model which concerns itself with providing hardware resources (physical or virtual machines)

b) Platform as a service (PaaS): the model facilitates the deployment of software by an independent developer without the need of managing the underlying hardware. Users are provided with databases, web servers and development tools, to name a few

c) Software as a service (SaaS): in this model, users can access software on remote servers. The provider is in charge of both platform and infrastructure on which the application runs

Openstack is mainly deployed as Infrastructure as a Service, therefore we are primarily interested in this particular model. Below there is an overview of services for which a user is responsible in each model.

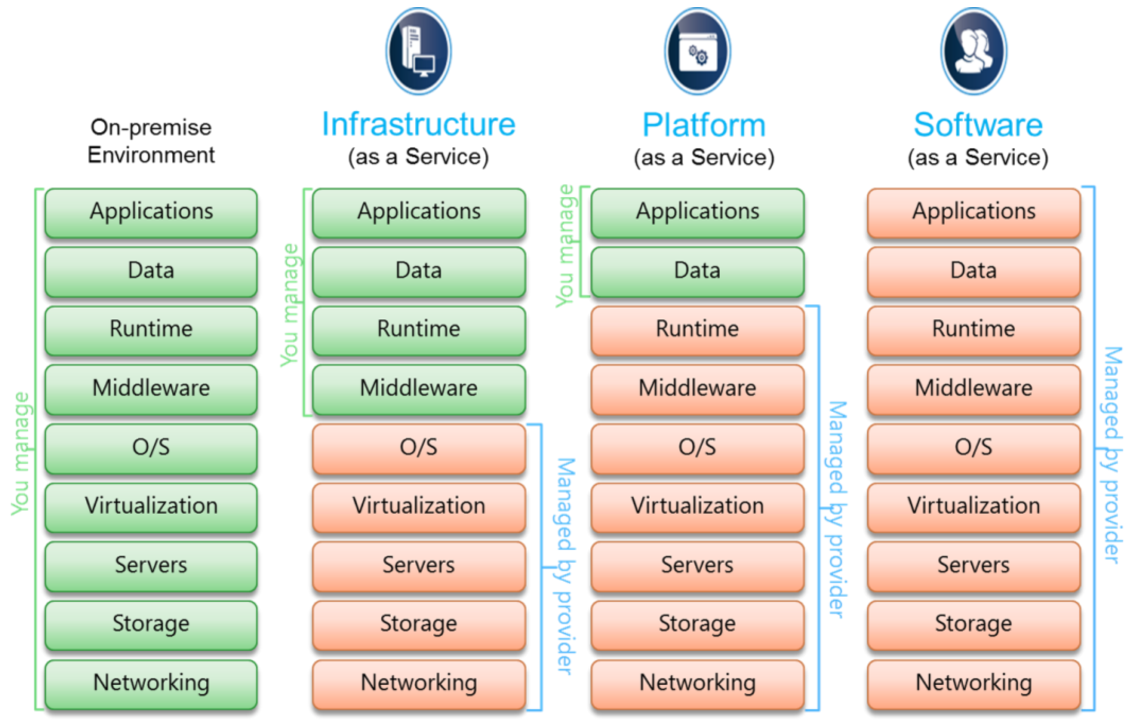

Figure 2. Comparison of different service models

1.3 Deployment models

There are four main deployment models. Private cloud is provisioned for exclusive use by a single organization. On the other hand, public cloud is available to the general public. There also exist community cloud, where the infrastructure is provisioned for use by a community that share the same concerns. Eventually, there is the hybrid cloud model which combines two or more aforementioned models.

2 Introduction to Openstack

2.1 Overview

Openstack is an open-source cloud computing software platform which provides an IaaS service model. It consists of a group of projects that allow managing storage and networking resource through a web-based dashboard, CLI tools, or a REST API.

CERN has been running Openstack in production for managing its private cloud since 2013.

2.2 Components of Openstack

Openstack consists of a number of components. At CERN, there are deployed 7 components:

- Openstack Compute (Nova): allows the user to create and manage virtual servers using the machine images.
- Block Storage (Cinder): provides persistent block storage to running instances.
- Identity Service (Keystone): provides an authentication and authorization service for other Openstack services.
- Image Service (Glance): provides the discovery, registration and delivery services for the disk and server images.
- Dashboard (Horizon): provides a web-based portal to interact with all the underlying services.
- Orchestration (Heat): implements an orchestration engine to launch multiple composite cloud applications.

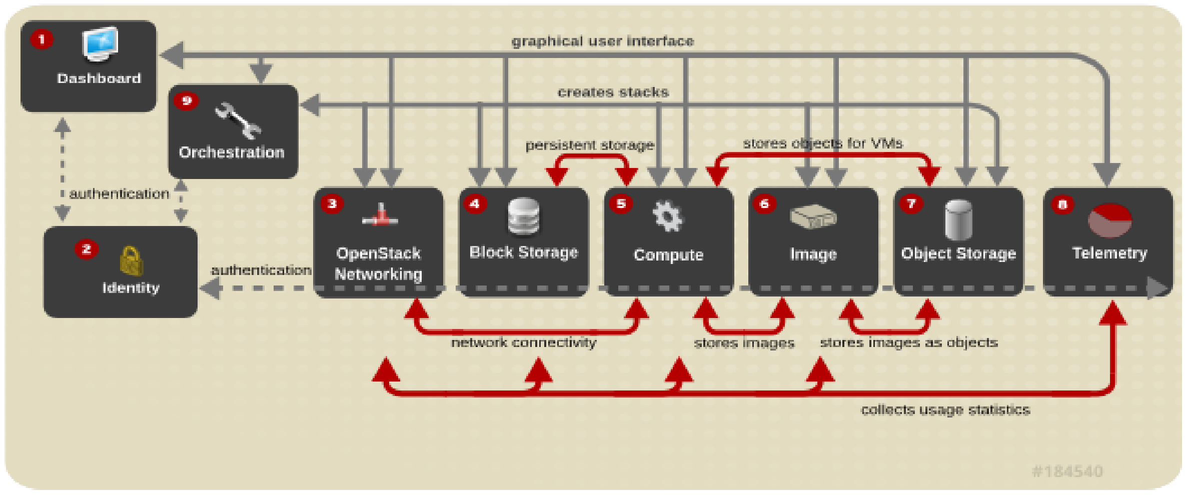

Figure 3 Openstack architecture

2.3 Keystone Identity Service

Keystone Identity module is of a particular interest to this project. Its purpose is to establish the identity of a user based on credentials and manage the actions that he can perform.

Keystone integrates functions for authentication, authorization, registration of tenants and users, granting tokens, and creating polices for users and services. The important thing is that Keystone is used between all Openstack services and provides a catalogue of available services and locations of API endpoints.

2.3.1 Keystone components

- User: digital representation of a person, system, or service that uses Openstack cloud. Users are assigned to a particular tenant, granted a role and associated information as username, password and email
- Tenant (Project): a container to group and isolate resources and users. It has to be specified in order to make requests to services. A tenant may map to a customer, account or organization.
- Role: it includes a set of privileges for performing specific actions that can be assigned to a specific user.
- Token: a short message returned by Keystone and used for accessing services. Each token has a scope which describes which resources are accessible with it.
- Endpoint: an address accessible over a network, described by URL, from where it is possible to make requests to Openstack services.
- Service: provides one or more endpoints through which users can access resources and perform operations. A service is identified by: identification number, name, service type, description and are associated with endpoints

2.3.2 Keystone workflow

- user launches client for the intended service and provides user credentials and an endpoint for the service location
- client makes a request to Keystone if a user has no token
- Keystone responds with an unscoped token if the project was not specified or with a scoped token otherwise. Unscoped token can be used only to list available projects and access to any services will not be granted. Once a user specifies project id, a scoped token can be returned by Keystone
- from the list returned by Keystone, the user chooses the endpoint and makes a request, including a scoped token
- A service returns a token to Keystone or validates it by itself, depending on token architecture

- If token is correctly validated, Keystone returns the project and the associated roles
- If the user is granted with an appropriate role and is authorised, the response is returned to a client

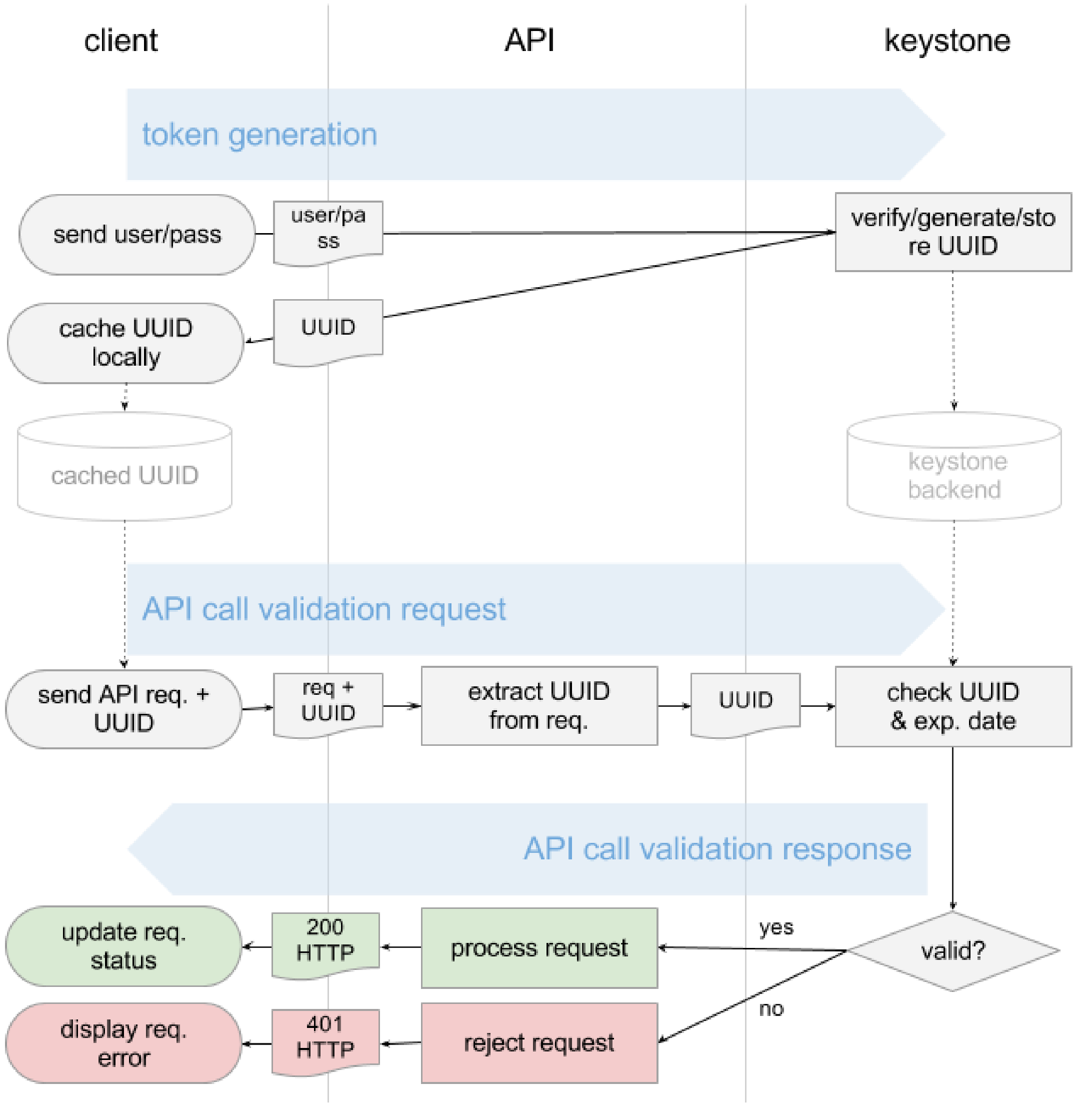

Figure 4 Authentication in Keystone

There are many advantages of this approach, namely this pattern provides a single point of authorization for all Openstack services and users always use the same interface. Moreover, users can access all of the services with a single authentication. It supports various authentication protocols with a common API on top.

2.3.3 UUID Tokens

UUID tokens' payload consists of 32 byte random string, thus it provides only uniqueness and has to be validated with Keystone, which maps a token to an identity. It also requires persistence.

2.3.4 Fernet Tokens

Fernet tokens are non-persistent, which means that they do not need to be persisted to a database. They contain minimal identity information and a dynamic authorization context – a user ID or federation ID, project ID, domain ID, and a creation timestamp. They typically fall within a range of 180 and 240 bytes. Payload is encrypted using a symmetric cipher and is cryptographically signed. In order to configure a deployment to use Fernet tokens, it is necessary to modify the keystone configuration file (/etc/keystone/keystone.conf):

```
[token]

provider = keystone.token.providers.fernet.Provider
```

It is also required to initialize a key repository:

```
> keystone-manage fernet_setup
```

It populates key repository directory with a pair of keys for encrypting and decrypting tokens.

2.3.5 PKI Tokens

Tokens based on public key infrastructure consist of whole validation response from Keystone and additional data as they are supposed to be validated offline. It results in great size (more than 1kB each) and poses many problems with handling revocation events, therefore these tokens are not popular within Openstack deployments.

2.4 Installing Openstack with Devstack

This section will be devoted to preparing environment with Openstack installed on it.

2.4.1 Virtual Machine

Openstack should be installed on a minimal version of the supported distribution. In this project Ubuntu 14.04 LTS has been used.

- create an image with a distribution

```
https://openstack.cern.ch/dashboard/project/images/
```

- launch a virtual machine via Dashboard. It is recommended to select at least m1.medium flavour and boot the machine from prepared image

```
https://openstack.cern.ch/dashboard/project/instances/
```

- connect to the machine via ssh

```
> ssh ubuntu@instance_name.cern.ch
```

- install git

```
> sudo apt-get install git
```

- download and install devstack. The installation script will prompt for various passwords during the first stage of the installation.

```
> git clone https://git.openstack.org/openstack-dev/devstack
> cd devstack; ./stack.sh
```

3 Federation in Openstack

The Keystone Federation is the mechanism that allows establishing trusts between identity providers and Openstack clouds. In result a user can securely access resources with single credential which is authenticaded by a trusted provider. The main advantage is that a user does not have to provision additional identities and re-login every time he wants to access resources.

There are several reasons the federation mechanism was developed and implemented in Keystone. First of all, it is inconvenient to deal with multiple tokens and administrating identities across cloud service providers. Furthermore, it poses security risks to issue new identities as well as use mass centralized storage for storing user credentials. Therefore, federated management techniques have been developed to share identity attributes without centrally storing this information.

Services must agree on a common protocol of exchanging information. There are several technologies that facilitate creating federation including Security Assertion Markup Language (SAML), OAuth, OpenID. We will focus on SAML, which was used to create a federation in our test environment.

There are three main building blocks of an identity federation:

- Identity Provider (IdP): a third party service (in our case testshib.org)that is trusted by the Identity API to authenticate identities.

```
{
        "identity_provider": {
                "description": None,
                "enabled": True,
                "id": "testshib",
                "remote_id": [u'https://idp.testshib.org/idp/shibboleth']
        }
}
```

- Protocol: contain information that dictates which mapping rules to use for an incoming requuest. Identity Providers can support multiple protocols.

```
{
	"protocol": {
		"id": "saml2",
		"mapping_id": "cern_mapping"
	}
}
```

- Mapping: a set of rules to map federation protocol attributes to Identity API objects. It translates remote atttributes to local attributes (Keystone entities). Below there is an example of such mapping, which indicates that anyone from the given set of e-mails authenticated by testshib.org will be mapped to a user with 'local_id' and assigned to a particular group.

```
{
	"mapping": {
		"id": "cern_mapping",
		"rules": [
		u'remote': [{u'type': u'eppn',
		             u'any_one_of': [u'myself@testshib.org']}],
		u'local': [{u'user': {u'id': u'local_id'}},
		           {u'group': {u'id': u'0cca258729c94efca447371e30d62c39'}
		]
	}
}
```

Federated users do not exist in Keystone, they exists on an IdP, who only returns attributes related to an identity. We utilize mappings to establish relationships between Keystone attributes and IdP attributes. In the aforementioned example, a federates user can authenticate with an IdP and mapped to a particular group, which will cause inheriting roles from the group.

3.1 SAML

Security Assertion Markup Language is a widely used authentication technique. The core of the standard are assertions. An assertion is an encrypted XML file that summarises identity and attribute information.

SAML protocol operates in the following way

- the user tries to access an Openstack service (the Service Provider)
- the user receives HTTP 302 code (redirection) and SAML request

- the client forwards the request to the Identity Provider
- in case there is no current session, the user provides necessary credentials
- the IdP returns a SAML assertion with a HTTP success code.
- the assertion can be now redirected to a particular Service Provider
- the user can access protected resource

3.2 Creating Federation in Keystone

Our goal is to set up a Federation in Keystone, where Keystone is the Service Provider (SP). We will assume that devstack is already installed on virtual machine.

First step is to configure Apache to use federation capable authentication method. For this we will use Shibboleth implementation.

3.2.1 Configuring Apache for Shibboleth

- Install libapache2-mod-shib2

```
> apt-get install libapache2-mod-shib2
```

- Edit virtual host configuration

```
> sudo nano /etc/apache2/sites-enabled/keystone.conf
```

- add following lines to <VirtualHost *:5000> section:

```
WSGIScriptAliasMatch
^(/v3/OS-FEDERATION/identity_providers/.*?/protocols/.*?/auth)$
/var/www/keystone/main/$1y


SSLEngine on
SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

- add following sections to the keystone.conf file

```
<Location /Shibboleth.sso>
    SetHandler shib
</Location>


<LocationMatch /v3/OS-FEDERATION/identity_providers/.*?/protocols/saml2/auth>
    ShibRequestSetting requireSession 1
    AuthType shibboleth
    ShibExportAssertion Off
    Require valid-user
</LocationMatch>

<LocationMatch /secure>
    ShibRequestSetting requireSession 1
    AuthType shibboleth
    ShibExportAssertion Off
    Require valid-user
</LocationMatch>
```

- Enable the ssl and shib2 modules

```
> sudo a2enmod ssl
> sudo a2enmod shib2
```

- Restart Apache service

```
> sudo service apache2 restart
```

3.2.2 Configuring shibboleth2.xml

- Generate certificates

```
> sudo shib-keygen -y <number of years>
```

- Download SAML Metadata from the server by going to the following address

```
http://instance_name.cern.ch/Shibboleth.sso/Metadata
```

- Upload the metadata to Identity Provider Service. It will register our Service Provider

```
http://www.testshib.org/register.html
```

- Backup the shibboleth configuration file

```
> sudo cp /etc/shibboleth/shibboleth2.xml
etc/shibboleth/shibboleth2.xml_backup
```

- Generating shibboleth.xml file on testshib.org is not necessary, we can edit the configuration file manually.
- Set entityID parameter in the ApplicationDefaults tag and remove REMOTE_USER parameter. It is important to specify a port in the entityID parameter. Parameter ECP allows to log in not only using web browsers.

```
<ApplicationDefaults entityID="https://instance_name.cern.ch:5000/shibboleth"
ECP="true">
```

- Restart Apache service

```
> sudo service apache2 restart
```

- We can verify the correctness of the installation by proceeding to:

```
https://instance_name.cern.ch:5000/secure/
```

It will result in an error after authentication which is expected.

3.2.3 Configuring Federation in Keystone

Change the authentication method in `/etc/keystone/keystone.conf` files:

```
[auth]
methods = external,password,token,oauth1,saml2
```

3.2.4 Openstack CLI

In order to start using Openstack command line interface, it is mandatory to set up required environmental variables. If we want to have administrator privileges, there is a predefined file with required variables that needs to be sourced:

```
> source devstack/accrc/admin/admin
```

It might be necessary to Identity API version from 2 to 3 in the admin file:

```
> export OS_AUTH_URL="http://instance_name.cern.ch:35357/v3"
> export OS_IDENTITY_API_VERSION=3
> export OS_PROJECT_NAME="demo"
> export OS_PROJECT_DOMAIN_NAME="default"
```

At this points we can access Openstack CLI:

```
> openstack
```

List of available commands:

```
(openstack)> help
```

3.2.5 Creating IdPs, mappings and protocol

We will assume that Openstack CLI is used.

```
> identity provider create testshib
> mapping create --rules mapping.json cern_mapping

> federation protocol create --identity-provider testshib --mapping
cern_mapping saml2

> identity provider set --remote-id https://idp.testshib.org/idp/shibboleth
testshib
```

Mapping.json file has the following structure:

```
[
  {
    "local": [
      {"user": {"id": "user"}},
      {"group": {"id": "0cca258729c94efca447371e30d62c39"}}],
    "remote": [{
        "type": "eppn",
       "any_one_of": ["myself@testshib.org"]
      }]
  }
]
```

In the first step we create an identity provider testshib. Then a mapping cern_mapping is created with the rules from mapping.json file. Now, we can create a federation protocol that will connect an identity provider and mapping rules. In our case it is called saml2 and the name has to be consistent with the name that we included in the keystone.conf file.

Eventually, we can set remote id for the identity provider. At this point, we can finally perform a federated authentication.

3.2.6 Keystone Python client

In order to authenticate yourself with Testshib identity, it is necessary to install python client and federation authentication plugins for Openstack. First of all, we need to install basic keystone client:

```
> pip install python-keystoneclient
```

Then, it is necessary to install Openstack Identity Authentication Library and Federation authentication plugins for Openstack. As they are not available in the repositories, it is recommended to download sourcecode directly from GitHub:

```
> git clone https://github.com/openstack/keystoneauth-saml2.git

> git clone https://github.com/openstack/keystoneauth.git
```

It is possible now to use Python bindings for the authentication process.

```
from keystoneclient_saml2.v3 import saml2
from keystoneclient import session
from keystoneclient.v3 import client
from keystoneauth1.auth.identity.v3 import password
```

In order to authenticate yourself the following commands need to be executed:

```
s = session.Session(verify=False)
plugin = password.Password(**CRED)
access = plugin.get_access(s)
```

CRED variable is a dictionary that consists of values such as authentication URL, user id, password and project id.

The access object contains auth_token among others. In order to perform actions, we need a client, which is accessible with the following commands:

```
endpoint = 'http://instance-name.cern.ch:5000/v3'
token = access.auth_token
client = client.Client(token=token, endpoint=endpoint)
```

At this point, client allows us accessing same services as Openstack CLI.

4 Keystone Architecture

Keystone is organized as a group of services exposed to endpoints. There are 6 services:

- Identity: validates users' credentials
- Resource: provides data about projects
- Assignment: provides data about roles and role assignments between objects from Identity and Resource services
- Token: validates and manages Tokens
- Catalog: provides an endpoint registry

Most of the services have Manager class (core.py) which provides the API functions of the service. Manager class is decorated with @dependency.provider decorator. It assures that only one object of Manager class is created in the system. With @dependency.requires decorator we can safely use APIs of other services. However, it is not called directly but by so called controller (controllers.py) which uses a necessary API.

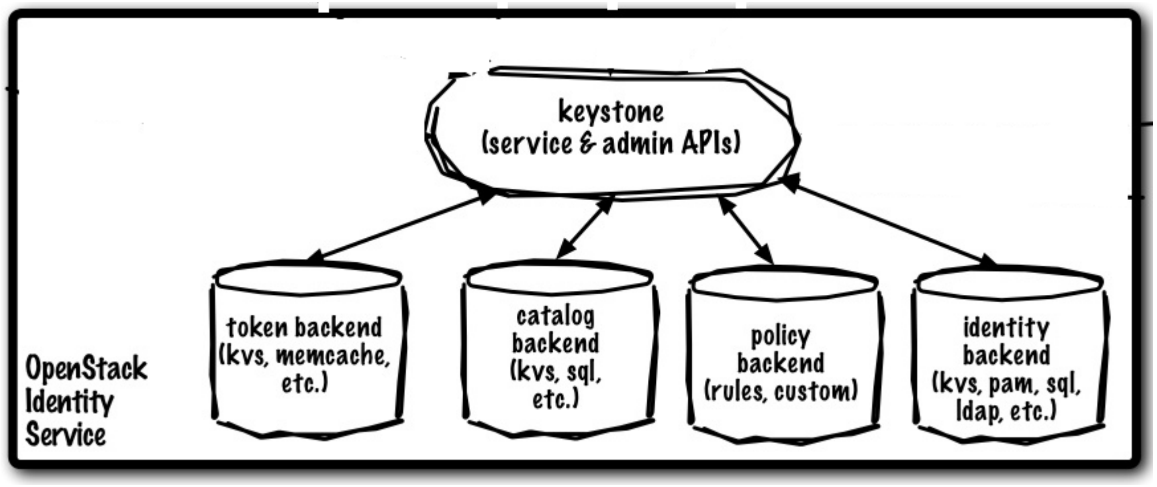

Figure 5 Keystone Architecture

Keystone provides different backends that each service can be configured to use. Different backends are available: key-value storage (KVS), a SQL backend (accessed through SQLAlchemy library; most focus was put on this backend), LDAP backend.

In order to efficiently use multiple backends, there exists the idea of a driver, which, according to the specified configuration in CONF variable, knows how to interact with appropriate backend.

4.1 Token revocation

For every type of token, there need to exists a mechanism that allows us invalidate tokens and prevent users using these particular tokens accessing resources.

As explained in previous chapters, token are issued every time a non-authenticated user tries to access resources.

A new entry is added to a token database. At this point we assume that SQL backend is used.

```
+------------------------------+--------------------+-------+------------------------------+
| id                           | expires            | valid | user_id                      |
+------------------------------+--------------------+-------+------------------------------+
| acc2b54 . . . 5              | 2015-08-27 14:46:02 |     1 | e237927 . . . a              |
+------------------------------+--------------------+-------+------------------------------+
```

4.1.1 Revocation lists

Revocation lists are the most simple mechanism of revoking tokens. When PersistenceManager gets a request for deleting a token:

```
delete_token(self, token_id)
```

it selects the token from the database and sets value in the `valid` column to False and saves the token on the revocation list. The token itself is added to the list, against incoming tokens can be compared for validation purposes.

This approach is straightforward but has its drawbacks. First of all, it does not work with Fernet tokens which are non-persistent and are not present in the database. Second of all, it may cause performance issue when multiple tokens has to be revoked.

4.1.2 Revocation event

Instead of invalidating every single token, the revocation event is created along a specific action, such as deleting a user, project or identity provider.

There is a revocation_event table in SQL database:

```
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| id              | varchar(64) | NO   | PRI | NULL    |       |
| domain_id       | varchar(64) | YES  |     | NULL    |       |
| project_id      | varchar(64) | YES  |     | NULL    |       |
| user_id         | varchar(64) | YES  |     | NULL    |       |
| role_id         | varchar(64) | YES  |     | NULL    |       |
| trust_id        | varchar(64) | YES  |     | NULL    |       |
| consumer_id     | varchar(64) | YES  |     | NULL    |       |
| access_token_id | varchar(64) | YES  |     | NULL    |       |
| issued_before   | datetime    | NO   |     | NULL    |       |
| expires_at      | datetime    | YES  |     | NULL    |       |
| revoked_at      | datetime    | NO   | MUL | NULL    |       |
| audit_id        | varchar(32) | YES  |     | NULL    |       |
| audit_chain_id  | varchar(32) | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
```

If we request an action of deleting specific user and project, the following revocation event will be generated in the database. These events can be further used to check tokens against them and make sure that tokens are valid. It consists of necessary data required for checking tokens' validity.

```
+------------+------------+---------------------+---------------------+
| user_id    | project_Id | issued_before       | revoked_at          |
+------------+------------+---------------------+---------------------+
| 519c2 ... 6 | NULL       | 2015-08-27 14:42:38 | 2015-08-27 14:42:38 |
| NULL         | 70d1 ... c | 2015-08-27 14:55:48 | 2015-08-27 14:55:48 |
+-------------+------------+--------------------+---------------------+
```

4.2 Testing

Openstack source code exists by default in /etc/stack/keystone. In order to perform unit tests, we may use the Tox tool. Tox creates a virtual environment for every run according to the configuration file. The configuration file for this testing tool is in tox.ini. Variable 'envlist' informs us about accessible enviroments. The simplest way to run all the test is:

```
> tox -e py27
```

However, the command runs all tests which may take 20 minutes. In order to run a single test, we need to use regular expressions or activate the virtual enviroment ourselves and run testr tool instead.

```
> tox -epy27 -- test_name_regex
```

or

```
> tox -epy27
keystone.tests.unit.test_auth.AuthBadRequests.test_authenticate_blank_auth
```

We can specify the full path of a test as shown above.

Using testr however, we can run only failing tests which is very useful during development process.

```
> source .tox/py27/bin/activate
> testr failing
> testr run --failing
```

4.3 Debugging

4.3.1 Keystone tests

Usually, debugging in Python boils down to adding:

```
> import pdb; pdb.set_trace()
```

This approach will not work in Keystone unit tests while running the aforementioned command. Instead, it is required to use Oslo Test's debug helper by running tox command with debug flag, for example:

```
> tox -e debug
keystone.tests.unit.test_auth.AuthBadRequests.test_authenticate_blank_auth
```

4.3.2 Live deployments

Debugging live deployments is impossible because of the fact that Keystone is running on Apache and we will run into an error while trying to use pdb. Instead of this, there is a Remote Python Debugger (rpdb) which can be easily installed with:

```
> pip install rpdb
```

and adding the following line in a source file:

```
> import rpdb; rpdb.set_trace()
```

By looking at a log file that pdb is running on localhost, on port 4444.

```
> tail -f /var/log/apache2/keystone.log
```

We can now attempt to connect to the service with netcat and use debugger in the same way as with pdb

```
> nc 127.0.0.1 4444
```

5 Conclusions

The report consists of information on necessary steps that need to be taken to start development with Openstack. It presents how to deploy a testing environment, explains Keystone architecture and describes testing techniques. It is crucial to understand the aforementioned steps in order to start contributing to Keystone Identity Service.

After configuring the environment, the following issues that appeared in Keystone has been solved. In the Federation extension, the actions associated with Identity Providers, such as deletion or update did not cause token invalidation. The problem has been fixed for both persistent and non-persistent tokens.