Table of Contents

OpenStack hierarchical Quota manager proposal

Introduction

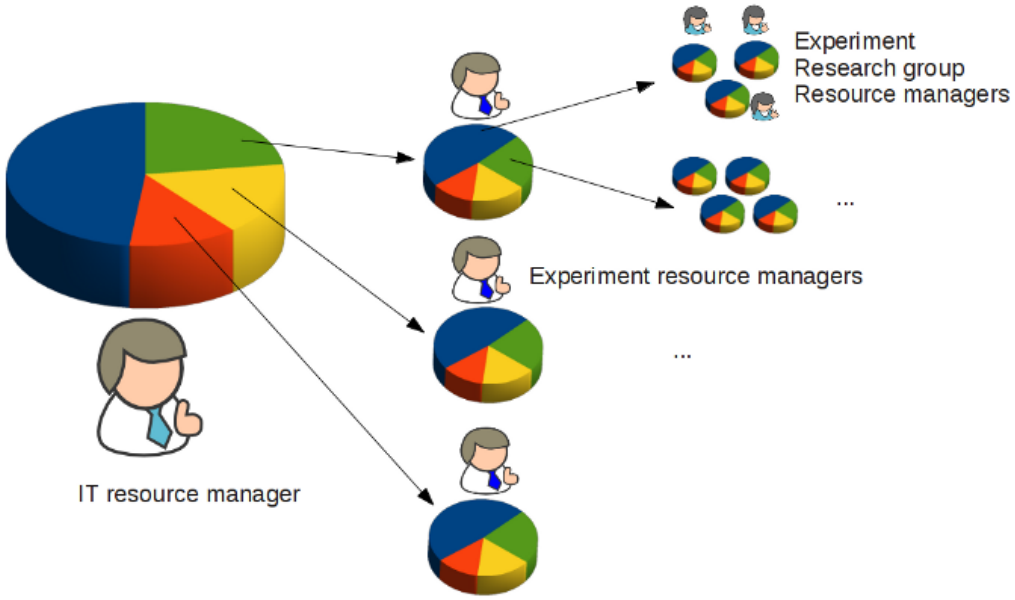
In this document we describe a proposal for a hierarchical quota manager for OpenStack. The idea is to join ideas from recent blue prints including Boson into a single architecture.

This document describes the full picture which we want to have at the end and describes the implementation-phases to accomplish the final goal.

User Stories

The CERN example

The CERN user story is best described by the figure below:



CERN IT has some resources which are managed by OpenStack. Projects (/Tenants) are created for the individual experiments (CMS, ATLAS, ALICE, LHCb, ...), and they get a quota which represents their piece of the cake. A person or a group of people in each of the experiments shall be able to do with these resources what they need to do. As these experiments are huge organizations themselves, the persons or groups of people who got the resources from IT will not use them for themselves but rather re-distribute them inside the experiments, for example to different research groups. The story repeats recursively: Each research group has a resource managers who in turn redistributes "his" bit of the cake to yet another group of people (eg people looking into different aspects of their specific field). As an example there might be a Higgs research group in CMS which has a bunch of sub groups specialized for different decay channels.

The recursion depth is not determined a priori but depends on the internal structure of the experiments. It is not in the hands of IT. Also, experiences from operations in more traditional resource allocation methods (for example batch systems) show that it is important to keep track of the change history. It is necessary to be able

to say who has changed what when and for which reason, even after some time has passed already.

The structure of the hierarchy can be viewed as a tree: The end users are the leaves of the tree while each nod corresponds to a quota value and a name of a user or a user group.

The pattern is not specific to CERN: We believe that large organizations (like HP or Rackspace) will sooner or later face a similar scenario.

More detailed user stories

1.) A service/administrator wants to manage quotas (reading, adding, updating, deleting quota-values) for a resource on per user basis.

Example: CERN Nova service wants to manage quotas for a resource separately for its various users who are working in ALICE, ATLAS and CMS. The quota value for each user can be different.

2.) A service/administrator wants to manage quotas (reading, adding, updating, deleting quota-values) for a resource on per project/tenant basis.

Example: CERN Nova service wants to manage quotas for a resource separately for projects/tenants named ALICE, ATLAS and CMS as quotas for ALICE, ATLAS and CMS can be different.

3.) In general, a service/administrator wants to manage quotas (reading, adding, updating, deleting quota-values) for a resource on per entity basis where entity can be user, user-group, project/tenant or domain.

4.) In a multi-level user hierarchy scenario, a user wants to delegate the quota (which has been allocated to him) to some other user-groups/user who is below him.

5.) In a multi-level project/tenant hierarchy scenario, a project/tenant owner wants to delegate the quota (which has been allocated to it) to some other tenants/users that are below it.

6.) In a multi-level user-group hierarchy scenario, a user-group wants to delegate the quota (which has been allocated to him) to some other user-groups/users who are below him.

7.) In a multi-level domain hierarchy scenario, a domain wants to delegate the quota (which has been allocated to it) to some other domains/project(tenants)/user-groups/users that are below it.

Design

Implementation details

The implementation is proposed to be done in 3 phases. These are

Phases

Phase-1
: Implement Quota Store with History Tracking

Phase-2
: Extend phase-1 to include Quota Management with Usage and History Tracking

Phase-3
: Use Phase-1 and Phase-2 for multi-level user/user-groups/tenants/domain hierarchy quota management with usage and history tracking

Phase-1: Implement Quota Store with History Tracking

It is proposed to have following tables to the keystone database:

Resources Table

Col Name	Type	Description
Id	VARCHAR(64)	Primary Key
Name	VARCHAR(255)	Name of the resource in the form Service-Name.Resource-Name
Child_Data_Identity	TEXT	A set of sets where each set specify the accepted combination of fields in the incoming request. This means that the request to read/add/delete/modify quota values for a partic entity (called child w.r.t. hierarchy linguistics) should have entity (child) data fields in a dictionary format and the keys in the dictionary should exactly to one of the sets mentioned here
		Example: [['user-id','role-id'],['user-id','role-id','project/tenant-id'],['user-id','role-id','doma
Purge_After	LONG	The duration (in seconds) after which the inactive (operationally deleted) reco the quota table should be actually deleted
Created_At	DATETIME	Date and Time of creation of the record in this table
Created_By	TEXT	Dictionary having data related to the entity who created this record
Deleted_At	DATETIME	Date and Time when a request for deleting this record was handled successfull
Deleted_By	TEXT	Dictionary having data related to the entity who deleted this record
Expire_At	DATETIME	Date and Time of expiry of this record in this table

Resources history table

Col Name	Type	Description
resource_id	VARCHAR(64)	Foreign Key referencing to id in Resources table
Updated_At	DATETIME	Date and Time of updation of any field in this record in this table
Updated_By	TEXT	Dictionary having data related to the entity who updated this record
Comment	VARCHAR(255)	comment to keep track of change requests

For certain resources, both user quota as well as project/tenant quota can be applicable. However, for some resources only project/tenant quota will be applicable. Besides, some user can fall directly within a domain (like domain-managers) and a project/tenant concept is not significant to it, at least for the quota calculations.

Hence depending on these conditions the Child_Data_Identity might look like [['user-id','role-id'],['user-id','role-id','project/tenant-id']] , where for a resource "both user quota as well as tenant quota can be applicable"

OR

[['user-id','role-id','tenant-id']], where for a resource "only tenant quota will be applicable"

OR

[['user-id','role-id','domain-id']]. where a user falls directly under a domain and hence "only domain quota will be applicable"

As there will be something like global-quota for a tenant i.e the quota allocated to a tenant by a service for a particular resource hence for those requests the Child_Data_Identity should have ['tenant-id','service-id'] or

['tenant-id','domain-id'] if instead of service it is the domain which gives quota to a project/tenant

Please note that, as user can attain multiple roles, so role-id should always be there when users are in picture.

Possible issues:

Quota Table

Col Name	Type	Description
id	VARCHAR(64)	Primary Key
child_data	TEXT	A dictionary having data related to the entity whose quota record is stored here. It can be a user or user-group or project/tenant or domain. The keys of this dictionary should exactly match with one of the sets in Child_Data_Identity in Resources Table
parent_data	TEXT	A dictionary having data related to the entity who can add/deleted/modify the quota values of the child. Useful in authenticating the request. The parent can be either actual parent as present (or will be present) in keystone or user-group or project/tenant or domain
resource_id	VARCHAR(64)	Foreign Key referencing to id in Resources table
quota_value	LONG	Actual unit less quota value of the child
Created_At	DATETIME	Date and Time of creation of the record in this table
Created_By	TEXT	Dictionary having data related to the entity who created this record
Deleted_At	DATETIME	Date and Time when a request for deleting this record was handled successfully
Deleted_By	TEXT	Dictionary having data related to the entity who deleted this record
Expire_At	DATETIME	Date and Time of expiry of this record in this table

Quota history table

Col Name	Type	Description
Quota_id	VARCHAR(64)	Foreign Key referencing to id in Quota table
Updated_At	DATETIME	Date and Time of updation of any field in this record in this table
Updated_By	TEXT	Dictionary having data related to the entity who updated this record
Comment	VARCHAR(255)	comment to keep track of change requests

---

This topic: AgileInfrastructure > AreaSchedulingAccountingOSHierarchicalQuotaManager
Topic revision: r15 - 22-May-2013 - UlrichSchwickerath