Chapter 14 :

Understanding Keystone

- Introduction
- Identity management (Keystone)
- How keystone works the way it does
- Basic Rest request and Response
- Token-flow diagram
- Importance of Tokens
- Tenants
- MySQL, keystone-querying table

14. Understanding Keystone

In this chapter we will cover:

1: General & Keystone identity management.
2: Explaining the working of Keystone/ the way it works.
3: Basic concept if Rest request and Response.
4: Token-flow.
5: Significance of Tokens.
6: Description of Tenants.
7: MySQL basics, keystone table and querying table.

Introduction

1. Identity Management (Keystone)

What is general ID management?

Identity management (ID management) is a administrative area that deals with identifying individuals in a system and controlling their access to resources within that system by associating user rights and restrictions with the established identity. Centralized identity management provides a huge advantage when using complex and widely distributed systems.

In case of cloud computing, as companies add more cloud services to their IT environments, the process of managing identities is getting more complex. There are so many moving parts to a public, hybrid, or multicloud deployment that security is a nightmare. The ability to assign centralized identities to all these parts, person, device, or data allows for more control, as well as more flexibility.

The idea of identity management in cloud involves each "actor" in a system, a device, person, database, server, or queue that goes to the mother of all identity servers to validate its credentials and be allowed access. This technique has advantages like:

- The ability to have common identity validation for systems both inside and outside the enterprise, such as those hosted on public clouds.
- The ability to centrally solve problems, such as identifying and neutralizing security problems.

- The ability to spend less on enterprise security by relying on the centralized trust model to deal with identity management across external and internal systems.

Keystone - Identity Management

Every multiuser service needs some mechanism to manage who can access the application and which actions each person can perform. A private cloud is no exception and OpenStack has streamlined these functions into a separate project called Keystone.

Keystone is the project name for OpenStack Identity, a service that provides token, policy, and catalog functions via an OpenStack application programming interface (API). Keystone represents an abstraction layer. It doesn't actually implement any user-management functions; rather, it provides plug-in interfaces so that organizations can leverage their current authentication services or choose from a variety of identity management systems that are on the market. Keystone integrates the OpenStack functions for authentication, policy management, and catalog services, including registering all tenants and users, authenticating users and granting tokens for authorization, creating policies that span all users and services, and managing a catalog of service endpoints. As we discussed earlier, the core object of an identity-management system is the user — a digital representation of a person, system, or service using OpenStack services. Users are often assigned to containers called tenants, which isolate resources and identity objects. A tenant can represent a customer, account, or any organizational unit.

As an identity management system, the main concepts of Keystone are listed below:

- **Authentication**: The process of confirming the identity of a user. To confirm an incoming request, OpenStack Identity validates a set of credentials that the user supplies. Initially, these credentials are a user name and password or a user name and API key. When OpenStack Identity validates user credentials, it issues an authentication token that the user provides in subsequent requests.

- **Credentials:** Data that confirms the identity of the user. For example, user name and password, user name and API key, or an authentication token that the Keystone service provides.

- **Domain**: Domain represents a collection of projects and users that defines administrative boundaries for the management of Keystone entities. A domain, which can represent an individual, company, or operator-owned space, exposes administrative activities directly to system users. Users can be granted the administrator role for a domain. A domain administrator can create projects, users, and groups in a domain and assign roles to users and groups in a domain.

- **Endpoint:** A network-accessible address, usually a URL, through which you can access a service. If you are using an extension for templates, you can create an endpoint template

that represents the templates of all consumable services that are available across the regions.

- **Group:** Group represents a collection of users that are owned by a domain. A group role granted to a domain or project applies to all users in the group. Adding users to, or removing users from, a group respectively grants, or revokes, their role and authentication to the associated domain or project.

- **OpenStackClient:** A command-line interface for several OpenStack services including the Identity API.

- **Project:** A container that groups or isolates resources or identity objects. Depending on the service operator, a project might map to a customer, account, organization, or tenant.

- **Region:** A region represents a general division in an OpenStack deployment. You can associate zero or more sub-regions with a region to make a tree-like structured hierarchy. Although a region does not have a geographical connotation, a deployment can use a geographical name for a region, such as US-East.

- **Role:** A personality with a defined set of user rights and privileges to perform a specific set of operations. The Keystone service issues a token that includes a list of roles to a user. When a user calls a service, that service interprets the set of user roles and determines to which operations or resources each role grants access.

- **Token:** An alpha-numeric text string that enables access to OpenStack APIs and resources. A token may be revoked at any time and is valid for a finite duration.

- **Users**: A digital representation of a person, system, or service that uses OpenStack cloud services. The Identity service validates that incoming requests are made by the user who claims to be making the call. Users have a login and can access resources by using assigned tokens. Users can be directly assigned to a particular project and behave as if they are contained in that project.

- **Tenants:** A tenant can be thought of as a project, group or organization. Whenever we make request to OpenStack services, we must specify a tenant. It can be thought of as a group of users used to isolate access to compute resources.

2. How keystone works?

The core object of an identity-management system is the user — a digital representation of a person, system, or service using OpenStack services. Users are often assigned to containers called tenants, which isolate resources and identity objects. A tenant can represent a customer, account, or any organizational unit.

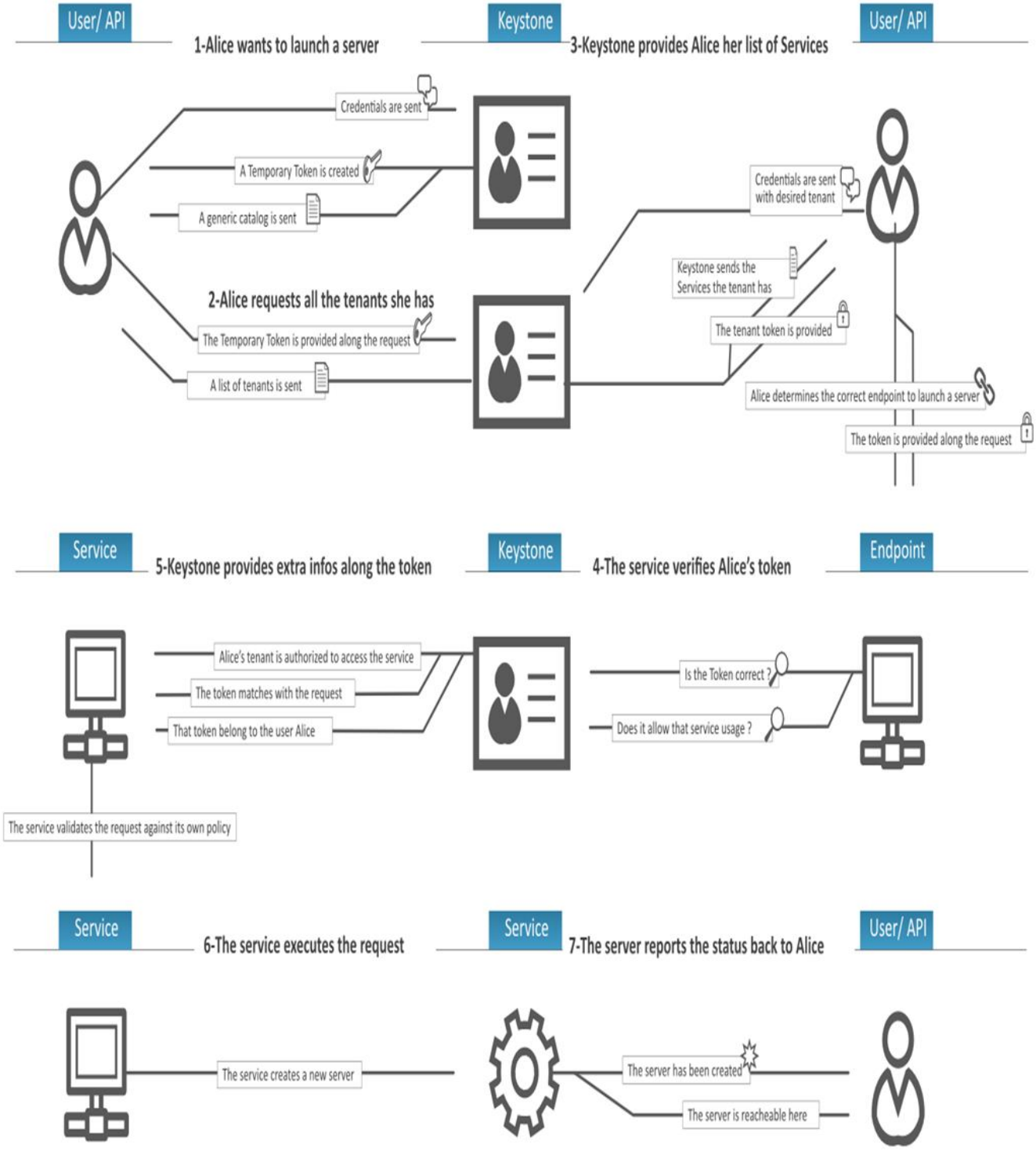
Authentication is the process of establishing who a user is. Keystone confirms that any incoming functional call originates from the user who claims to be making the request. It performs this validation by testing a set of claims which take the form of credentials. The distinguishing feature of credential data is that it should only be accessible to the user who owns the data. It can consist of data only the user knows (user name and password or key), something the user physically possesses (a hardware token), or something the user "is" (biometric data like an iris scan or fingerprint).

After OpenStack Identity has confirmed the user's identity, it provides the user with a token that corroborates that identity and can be used for subsequent resource requests. Each token includes a scope that lists the resources to which it apples. The token is valid only for a finite duration and can be revoked if there is a need to remove a particular user's access.

Security policies are enforced with a rule-based authorization engine. After a user has been authenticated, the next step is to determine the level of authorization. Keystone encapsulates a set of rights and privileges with a notion called a role. The tokens that the identity service issues include a list of roles that the authenticated user can assume. It is then up to the resource service to match the set of user roles with the requested set of resource operations and either grant or deny access.

One additional service of Keystone is the service catalog used for end point discovery. This catalog provides a listing of available services along with their API end points. An end point is a network-accessible address, such as a URL, from which a user can consume a service. All of the OpenStack services, including OpenStack Compute (Nova) and OpenStack Object Storage (Swift) supply end points to Keystone through which users can request resources and perform operations.

Figure 1: Keystone working process (identity management)



3. Structure of basic Rest request and Response:

What is Rest?

REST (REpresentational State Transfer) is an architectural style, and an approach to communications that is often used in the development of Web services. The use of REST is often preferred over the more heavyweight SOAP (Simple Object Access Protocol) style because REST does not leverage as much bandwidth, which makes it a better fit for use over the Internet

REST sits on this stack in a way that makes it easy for humans to understand what's being exchanged while allowing computers to talk to one another efficiently.

```
$ curl google.com //requests to talk to "google.com"
$ curl -X Get google.com
```

For an example:

```
 * Documentation:  https://help.ubuntu.com/
openstack@lab-utsa-fundamentals-n03:~$ ls
openstack@lab-utsa-fundamentals-n03:~$ curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
openstack@lab-utsa-fundamentals-n03:~$ curl -X GET google.com
```

These requests and responses has Restful API. REST can easily handle more complex requests, including multiple parameters. In most cases, you'll just use HTTP GET parameters in the URL.

If you need to pass long parameters, or binary ones, you'd normally use HTTP POST requests, and include the parameters in the POST body.

As a rule, GET requests should be for read-only queries; they should not change the state of the server and its data. For creation, updating, and deleting data, use POST/DELETE/PUT etc requests. (POST can also be used for read-only queries, as noted above, when complex parameters are required.)

Let’s see another way to look into the responses is:

```
$ curl google.com -v
```

- It gives the response back in terms of header & data:

```
openstack@lab-utsa-fundamentals-n03:~$ curl google.com -v
* Rebuilt URL to: google.com/
* Hostname was NOT found in DNS cache
*   Trying 2607:f8b0:4004:809::1007...
* Connected to google.com (2607:f8b0:4004:809::1007) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.35.0
> Host: google.com
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Location: http://www.google.com/
< Content-Type: text/html; charset=UTF-8
< Date: Wed, 07 Oct 2015 07:12:54 GMT
< Expires: Fri, 06 Nov 2015 07:12:54 GMT
< Cache-Control: public, max-age=2592000
* Server gws is not blacklisted
< Server: gws
< Content-Length: 219
< X-XSS-Protection: 1; mode=block
< X-Frame-Options: SAMEORIGIN
<
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

So now this response gets back with two things 1) header & 2) body (data response). Here the content type means what is there in the body, it returns that to us. Here its text.d

Now lets’ try this..

```
$ curl www.google.com
```

It gives us the response results that runs here without any GUI.

Same like this Opnestack services work with restful API that can be handled with

- Curl
- Python code
- wget
- apget etc

The reason why it works with the Restful API is that, it helps the developer to interact and understand the workflow and services. So for the openstack when we send the request it should be in the form such that it should contains: "curl –GET"

- username
- password
- URL
- Type of the service (eg keystone)

In response we get the following: these responses are required to access openstack services.

- End points: (it contains service catalog)
- session ID
- & Token

So now have everything to authenticate us as a user, we make another request say to "Glance" service. It will now verify and give the list of the images.

Lest see an example:

```
      $ curl –d @auth.json –H “content-type: application/json”
http://66.150.45.132:5000/v2.0/token

      # it gives the token back as

            “token” :{
                      “audit_ids” : [
                              “MnQvPsq0sf5Us2Ls7K7”
```

So now we run this, it gives the image list like this:

```
                  # keystone endpoint-create \
              --service-id=the_service_id_above \
            --publicurl=http://controller:5000/v2.0 \
           --internalurl=http://controller:5000/v2.0 \
             --adminurl=http://controller:35357/v2.0
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| adminurl    | http://controller:35357/v2.0     |
| id          | 11f9c625a3b94a3f8e66bf4e5de2679f |
| internalurl | http://controller:5000/v2.0      |
| publicurl   | http://controller:5000/v2.0      |
| region      | regionOne                        |
| service_id  | 15c11a23667e427e91bc31335b45f4bd |
+-------------+----------------------------------+
```

4. Token flow diagram

Token Based Authentication

Token based authentication is one of the most prominent features over the web nowadays. With almost every web company using API, tokens are the best way to handle authentication for multiple users. Any major API or web application that one has come across has most likely used tokens. Applications like Facebook, Twitter, Google+, GitHub etc. uses tokens.

Earlier, authentication was done using Server based traditional method. Since HTTP protocol is stateless, meaning if we authenticate user with a username and password, then on next request, our application won’t know who we are. We then need to authenticate again. We therefore need to store user logged in information on the server. Few major problems arose due to this: [2]

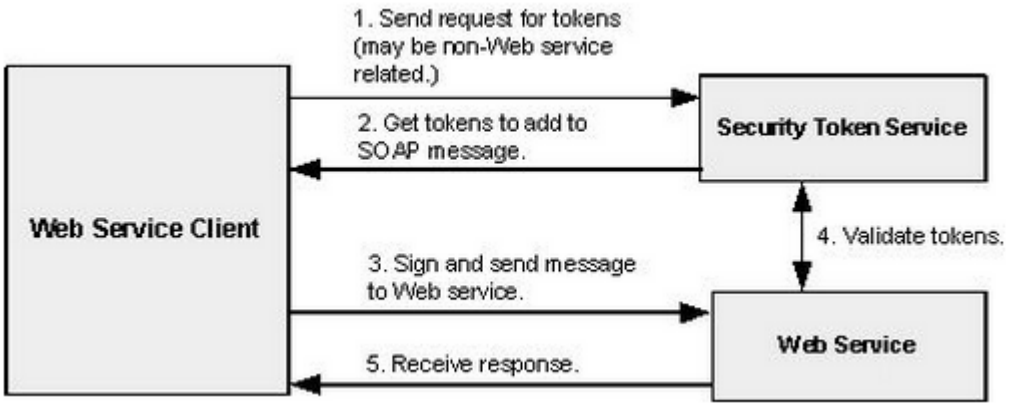
- **Sessions:** Every time user is authenticated, server will need to create a record somewhere on our server. This is usually done in memory and when there are many users authenticating, overhead of server increases
- **Scalability**: Since sessions are stored in memory, this provides problems with scalability. As cloud providers start replicating servers to handle application load, having virtual information in session memory will limit the ability to scale.
- **Cross Site Request Forgery:** Users are susceptible to CSRF attacks since they can already be authenticated with say a banking site and this could be taken advantage of when visiting other sites.

How Token Works

Token based authentication is stateless i.e. no information about users on server or session is being stored, thus taking care of many problems. No Session information meaning application can scale and add more machines as necessary without worrying about where a user is logged in.

Although this implementation can vary, the gist of it is as follows:

- User Requests Access with Username / Password
- Application validates credentials
- Application provides a signed token to the client
- Client stores that token and sends it along with every request
- Server verifies token and responds with data



Sample Token Flow Diagram [1]

Every single request will be requiring a token. Token should be sent in HTTP header so that we keep with the idea of Stateless HTTP requests. After authentication of our information, we

can do many things with token. We can create permission based token and pass this along to a third-party application etc.

5. Importance of Tokens/ Key benefits

- **Stateless and Scalable**
  Tokens stored on client side. Completely stateless and ready to be scaled. Our load balancers are able to pass a user along to any of our servers since there is no state or session information anywhere. Token itself holds the data for the respective user thus eliminating the problem that if we wish to keep session information on a user logged in, this would require to keep sending that user to same server they are already logged in, therefore some users would be forced to same server resulting in bringing heavy traffic. [2]

- **Security**
  The token is sent on every request and since there is no cookie being sent, this helps in preventing against CSRF attacks. Even if your specific implementation stores the token within a cookie on the client side, the cookie is merely a storage mechanism instead of an authentication one. There is no session based information to manipulate since we don't have a session. The token also expires after a certain amount of time, thus a user will be required to login again. This helps us stay secure. There is also the concept of token revocation that allows us to invalidate a specific token and even a group of tokens based on the same authorization grant.

- **Extensibility**
  Tokens will allow us to build applications that share permissions with another. For example, we have linked random social accounts to our major ones like Facebook or Twitter. When we login to Twitter through a service (let's say Buffer), we are allowing Buffer to post to our Twitter stream. We could even build our own API and hand out special permission tokens if our users wanted to give access to their data to another application. [2]

Tokens in Keystone

So where does OpenStack begin and end? The users will say that it's their cloud GUI or CLI. From the architectural perspective, however, OpenStack middleware ends on its API endpoints: nova-api, glance-api, and so on they all expose their APIs over http. You can connect to these APIs using different clients. Clients can be either certified by a cloud vendor and deployed on its infrastructure (for example, a Horizon instance) or installed anywhere else (a python-nova client installed on any laptop, pointing to a remote nova-api). This boundary implies one important condition. Since the clients can reside anywhere, they cannot be trusted. Any request coming from them to any OpenStack API endpoint needs to be authenticated before it can be processed further.

So what do you have to do in this case? Does that mean you should supply each single API request with a username and password? Not so easy. Or maybe store them in some environment variables? Insecure. The answer to this is tokens.

Tokens—what they are and why you need them

Tokens have one significant plus: they are temporary and short lived, which means it is safer to cache them on clients than username/password pairs.

In general, a token is a piece of data given to a user by Keystone upon providing a valid username/password combination. As said above, what is closely related to a token is its expiration date (which typically is hours or even minutes). The user client can cache the token and inject it into an OpenStack API request. The OpenStack API endpoints take the token out of user requests and validate it against the Keystone authentication backend, thereby confirming the legitimacy of the call.

6. Tenants

Concept of Tenant:

As discussed above, a tenant can be thought of as a project, group or organization. In fact, there have been repeated attempts to make the name be either tenant or project, but as a result of an inability to stick to just one term they both mean the same thing. As far as the API is concerned a project is a tenant. So if you log into horizon you will see a drop down for your projects. Each project corresponds to a tenant id. Your tokens are associated with a specific tenant id as well. So you may need several tokens for a user if you intend to work on several tenants the user is attached to. Whenever we make request to OpenStack services, we must specify a tenant.

Now, say you add a user to the tenant id of admin. Does that user get admin privileges? The answer is no. That's where roles come into play. While the user in the admin tenant may have access to admin virtual machines and quotas for spinning up virtual machines that user wouldn't be able to do things like query keystone for a user list. But if you add an admin role to that user, they will be endowed with the ACL rights to act as an admin in the keystone API, and other APIs. So think of a tenant as a sort of resource group, and roles as an ACL set.

The following figure shows the mapping between tenant, user, role and permissions:

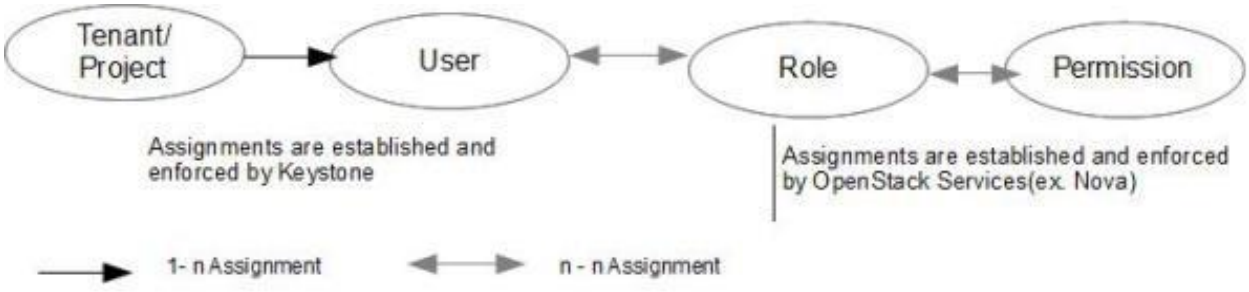


Figure 1- Mapping between tenant, user, role and permissions

From the above figure, a tenant can be perceived as **a group of zero or more users. In nova, a tenant owns virtual machines. In swift, a tenant owns containers. Users can be associated with more than one tenant. Each tenant and user pairing can have a role associated with it.** Every resources (user, virtual machine, network, object in swift, container) in OpenStack is tied to a tenant. All users of the tenant may (or may not) access these resources based on the role they are assigned to.

In order to see tenant list of the system, the command is:

```
$ keystone –os-username=tenant user-list
```

To view all the users in the system, the command is:

```
$ keystone –os-username=admin user-list
```

To view the available roles in the system, the command is:

```
$ keystone –os-username=admin role-list
```

7. MySQL with Keystone

MySQL is an open-source database system provided in the format of the client-server model. For typical Linux installations, MySQL is installed as a service through the MySQL daemon, mysqld. By default, the daemon listens on the port 3306, as defined by the /etc/services file. Setting up MySQL initially will also, by default, listen on the localhost permitting only the host running the daemon to connect to the any active databases.

For MySQL's use in connection with OpenStack's Keystone project is an optional driver for the Identity driver for users and groups. The two options for the "domain-specific" options are LDAP (directory structured) or SQL (database structured). In the presentation of this document, the Keystone implementation will be using the SQL driver and the LDAP will be ignored for now.

MySQL CLI

In order to interact with MySQL, first, one must ensure that the mysqld daemon is actively running on the system. The command in which to determine whether it is running varies from one operating system to the next. For this example, we are using a default installation of Ubuntu 14.04.2 LTS. To check, we can use the service command:

```
openstack@lab-utsa-fundamentals-n10:~$ sudo service mysql status
      * /usr/bin/mysqladmin Ver 9.0 Distrib 5.5.44-MariaDB, for debian-linux-
gnu on x86_64
      Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.
      Server version 5.5.44-MariaDB-1ubuntu0.14.04.1
      Protocol version 10
      Connection Localhost via UNIX socket
      UNIX socket /var/run/mysqld/mysqld.sock
      Uptime: 3 days 22 hours 24 min 34 sec
      Threads: 1 Questions: 2720 Slow queries: 0 Opens: 130 Flush tables: 2
Open tables: 71
```

NOTE: It is worth noting that opposed to running regular MySQL, in this instance, we are running MariaDB. MariaDB is an open-source fork of MySQL and can be considered a drop-in for MySQL as it tends to support all the same functionality and then some.

Now that we know that the service is actively running, we may now attempt to connect to it using the client program, mysql. To do so, we must pass an authenticated user and password that was set up during the database's creation / configuration. For our setup, the username will be root and the password will be notmysql. So to connect, we issue:

```
openstack@lab-utsa-fundamentals-n10:~$ mysql -uroot -pnotmysql
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 124
Server version: 5.5.44-MariaDB-1ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

And to see which databases are currently available to us, we may issue the show databases; command.

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| keystone           |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.00 sec)

MariaDB [(none)]>
```

As we can see, we have four databases made available to us. Both of the _schema databases are used for metadata and information relating to the actual relational schema of the other databases contained within. The mysql database concerns configurations and other options for the MySQL daemon. For our concerns, we are obviously most interested in the keystone database. To set this as our current database, we can execute:

```
MariaDB [(none)]> use keystone;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [keystone]>
```

NOTE: If you are already aware of the database you wish to use when starting the mysql client program, you can specify the database name at the end of the command. For example, using the command, mysql -uroot -pnotmysql keystone, would start the MySQL client already using keystone as its active database.

Now, here we may wish to view the tables present in the database. To do so, we can use show tables; as so:

MariaDB [keystone]> show tables;

```
+------------------------+
| Tables_in_keystone     |
+------------------------+
| access_token           |
| assignment             |
| consumer               |
| credential             |
| domain                 |
| endpoint               |
| endpoint_group         |
| federation_protocol    |
| group                  |
```

```
| id_mapping             |
| identity_provider      |
| idp_remote_ids         |
| mapping                |
| migrate_version        |
| policy                 |
| policy_association     |
| project                |
| project_endpoint       |
| project_endpoint_group |
| region                 |
| request_token          |
| revocation_event       |
| role                   |
| sensitive_config       |
| service                |
| service_provider       |
| token                  |
| trust                  |
| trust_role             |
| user                   |
| user_group_membership  |
| whitelisted_config     |
+------------------------+
32 rows in set (0.00 sec)
MariaDB [keystone]>
```

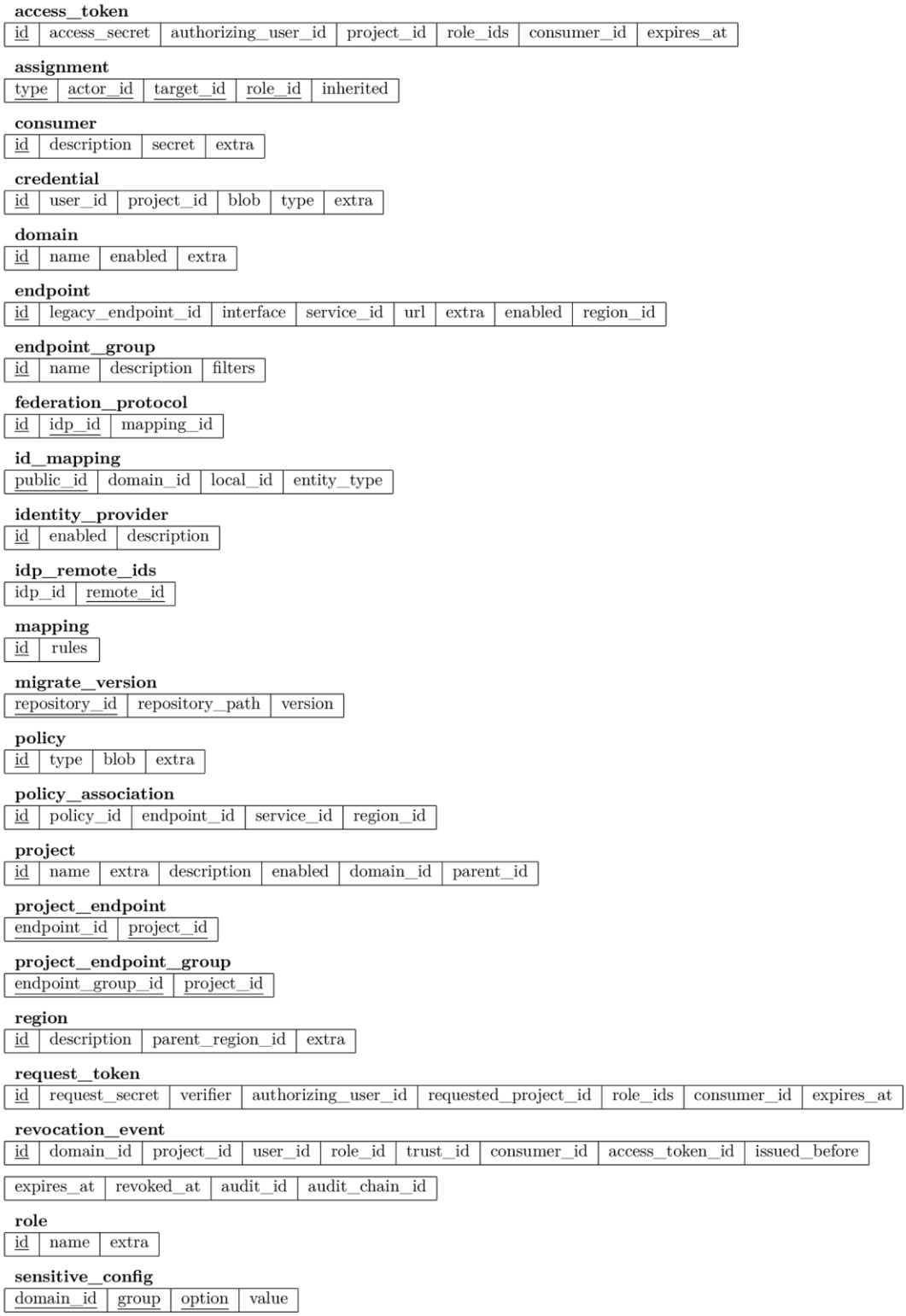
One final command that may be of use is using the describe command on a table as a means of describing the relation's schema.

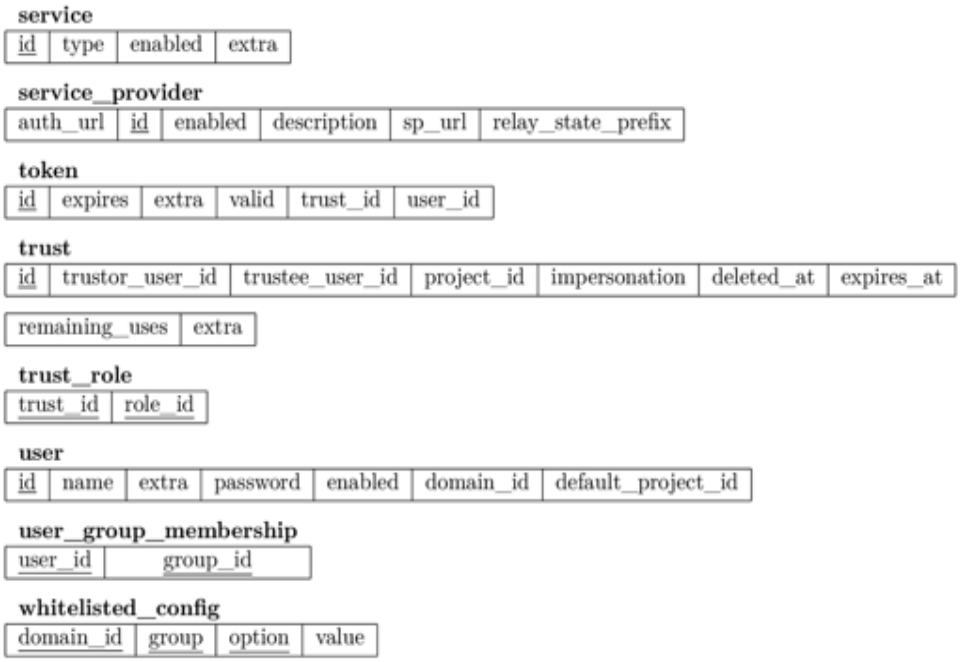
```
MariaDB [keystone]> describe access_token;
+---------------------+-------------+------+-----+---------+-------+
| Field               | Type        | Null | Key | Default | Extra |
+---------------------+-------------+------+-----+---------+-------+
| id                  | varchar(64) | NO   | PRI | NULL    |       |
| access_secret       | varchar(64) | NO   |     | NULL    |       |
| authorizing_user_id | varchar(64) | NO   | MUL | NULL    |       |
| project_id          | varchar(64) | NO   |     | NULL    |       |
| role_ids            | text        | NO   |     | NULL    |       |
| consumer_id         | varchar(64) | NO   | MUL | NULL    |       |
| expires_at          | varchar(64) | YES  |     | NULL    |       |
+---------------------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

MariaDB [keystone]>
```

From this point on, most other commands are standard SQL / MySQL commands. It is important to note that the CLI supports multiline commands and therefore requires that you end each command with a ';' character so that it knows when to execute your command. For example, SELECT * from user will not execute unless you add the ';' to make SELECT * from user; before pressing return. Finally, to exit the current CLI, one only needs to type exit or quit.

Keystone Database Table Schema





References

1. http://docs.openstack.org/developer/keystone/
2. http://www.ibm.com/developerworks/cloud/library/cl-openstack-keystone/
3. https://developer.atlassian.com/display/CROWDDEV/JSON+Requests+and+Responses
4. http://docs.openstack.org/juno/install-guide/install/apt/content/keystone-concepts.html
5. http://docs.openstack.org/developer/keystone/architecture.html
6. http://rest.elkstein.org/2008/02/more-complex-rest-requests.html
7. http://www-01.ibm.com/support/knowledgecenter/SS3JSW_5.2.0/com.ibm.help.web_services.doc/SI_ScrtyImplmntn_WS.html
8. https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication
9. https://www.mirantis.com/blog/understanding-openstack-authentication-keystone-pki/
10. http://www.computerweekly.com/news/1519999/Cloud-computing-identity-management-presents-unique-challenges
11. https://www.youtube.com/watch?v=nHCxcN3pgQw
12. http://docs.openstack.org/admin-guide-cloud/identity_concepts.html
13. http://utsa.cloudtrain.me/fundamentals/slides/keystone.pdf

14. http://blog.flux7.com/blogs/openstack/tutorial-what-is-keystone-and-how-to-install-keystone-in-openstack
15. http://docs.openstack.org/user-guide-admin/manage_projects_users_and_roles.html
16. http://docs.openstack.org/icehouse/install-guide/install/apt/content/keystone-users.html
17. http://stackoverflow.com/questions/19004503/the-relationship-between-endpoints-regions-etc-in-keystone-openstack
18. https://prosuncsedu.wordpress.com/2014/02/08/authentication-and-authorization-model-in-openstack/
19. http://docs.openstack.org/developer/keystone/