Learning Keystone

Email: wu.wenxiang@99cloud.net

Email: maodouzi@gmail.com

Sina Weibo: @wu_wenxiang

概述

Keystone是OpenStack的组件之一，用于为OpenStack所有组件提供统一的认证服务。

安装

安装基本系统

参见《Learning DevStack》中的“安装基本系统”篇

安装Keystone

1. 安装步骤：
    1. 系统更新

       ```
       sudo apt-get update
       sudo apt-get dist-upgrade -y
       ```

    2. 安装依赖的软件包

       ```
       sudo apt-get install build-essential python-dev \
           python-setuptools python-pip python-ldap curl \
           libxml2-dev libxslt-dev mysql-server mysql-client \
           python-mysqldb -y
       ```

    3. 在mysql数据库里创建keystone用户

       ```
       mysql -u root -p
       create database keystone;
       grant all on keystone.* to 'keystone'@'%' identified \
           by 'openstack';
       grant all on keystone.* to 'keystone'@'localhost' \
           identified by 'openstack';
       quit;
       ```

    4. 安装keystone

```
cd ~
git clone https://github.com/openstack/keystone.git
cd ~/keystone
sudo pip install -r tools/pip-requires
sudo pip install -r tools/test-requires
sudo python setup.py develop
```

5. 安装keystone client

```
cd ~
git clone https://github.com/openstack/\
    python-keystoneclient.git
cd ~/python-keystoneclient
sudo pip install -r tools/pip-requires
sudo pip install -r tools/test-requires
sudo python setup.py develop
```

6. 配置keystone

```
sudo mkdir -p /etc/keystone
sudo cp ~/keystone/etc/* /etc/keystone/
sudo mkdir -p /var/log/keystone
sudo touch /var/log/keystone/keystone.log
sudo cp /etc/keystone/keystone.conf.sample \
    /etc/keystone/keystone.conf
```

7. 修改/etc/keystone/keystone.conf

    1. 在[default]里添加：

    ```
    admin_token = 231
    log_dir = /opt/stack/keystone/
    log_file = keystone.log
    ```

    2. 在[sql]里添加：

    ```
    connection = mysql://root:231@localhost/\
        keystone?charset=utf8
    ```

    3. 在[token]里添加：

    ```
    driver = keystone.token.backends.sql.Token
    ```

    4. 在[catalog]里添加：

    ```
    driver = keystone.catalog.backends.sql.Catalog
    ```

    5. 在[signing]里添加：

    ```
    token_format = UUID
    ```

8. 配置SSL

```
sudo mkdir -p /etc/keystone/ssl
sudo cp -r ~/keystone/examples/pki/* /etc/keystone/ssl/
cd /etc/keystone/ssl
sudo sudo ./gen_pki.sh
```

9. 数据库创建

```
sudo keystone-manage db_sync
```

2. 启动

```
sudo keystone-all -d --debug -v
```

3. 初始化

```
keystone --token 231 --endpoint http://localhost:35357/v2.0 \
    tenant-create --name=admin # 后面的命令会省略token和endpoint
keystone user-create --name=admin --pass=231 \
    --tenant-id=<admin tenant id>
keystone role-create --name=admin
keystone user-role-add \
    --user_id=<admin-user-id> \
    --tenant_id=<admin-tenant-id> \
    --role_id=<admin-role-id>
keystone service-create     --name=keystone --type=identity
keystone service-create     --name=swift --type=object-store
keystone endpoint-create \
    --region=RegionOne \
    --service-id=<keystone-service-id> \
    --publicurl=http://localhost:5000/v2.0/ \
    --adminurl=http://localhost:35357/v2.0/ \
    --internalurl=http://localhost:5000/v2.0/
# 或者--admincurl=http://localhost:$(admin_port)s/v2.0/
keystone endpoint-create \
    --region=RegionOne \
    --service-id=<swift-service-id> \
    --publicurl=http://localhost:8080/v1/AUTH_$(tenant_id)s \
    --adminurl=http://localhost:8080 \
    --internalurl=http://localhost:8080/v1/AUTH_$(tenant_id)s
```

4. 查看命令

```
keystone --os-username=admin --os-password=231 \
    --os-auth-url=http://localhost:35357/v2.0 \
    --os-tenant-name=demo user-list
# 同样还有tenant-list, role-list, endpoint-list, service-list
```

5. 测试命令

```
curl -d '{"auth": {"tenantName": "admin", "passwordCredentials":\
    {"username": "admin", "password": "231"}}}' -H \
    "Content-type:application/json" http://localhost:35357\
    /v2.0/tokens | python -mjson.tool
```

源代码分析

Keystone Client

1. 代码结构

```
keystoneclient/
keystoneclient/access.py # AccessInfo类的实现，用于封装验证请求的信息
keystoneclient/base.py # Manage类，完成CLI Manage操作的API的基类。
keystoneclient/client.py # class HTTPClient(httplib2.Http)
keystoneclient/common
keystoneclient/common/cms.py # SSL相关，CMS是加密消息语法的意思
keystoneclient/contrib
keystoneclient/contrib/bootstrap
keystoneclient/contrib/bootstrap/shell.py # bootstrap模式，直接从
    #user/tenant/role的name参数，调用user-role-add。
keystoneclient/exceptions.py # 定义异常类
keystoneclient/generic
keystoneclient/generic/client.py # 通用的client，通过client对象调用
keystoneclient/generic/shell.py # 通用的shell，CLI接口
keystoneclient/middleware
keystoneclient/middleware/auth_token.py # 用于验证token是否正确
keystoneclient/middleware/test.py
keystoneclient/openstack
keystoneclient/openstack/common
keystoneclient/openstack/common/cfg.py # config文件和CLI解析
keystoneclient/openstack/common/iniparser.py # ini文件解析
keystoneclient/openstack/common/jsonutils.py # JSON相关
keystoneclient/openstack/common/setup.py
keystoneclient/openstack/common/timeutils.py
keystoneclient/service_catalog.py
    # 从Keystone的反馈里取到token和endpoint
keystoneclient/shell.py # CLI入口，详见setup.py
keystoneclient/utils.py
keystoneclient/v2_0 # Common接口的具体实现
keystoneclient/v2_0/client.py
keystoneclient/v2_0/ec2.py
keystoneclient/v2_0/endpoints.py
keystoneclient/v2_0/roles.py
keystoneclient/v2_0/services.py
keystoneclient/v2_0/shell.py
keystoneclient/v2_0/tenants.py
keystoneclient/v2_0/tokens.py
keystoneclient/v2_0/users.py
keystoneclient/v3 #类同v2_0
```

2. keystoneclient.v2_0.client：

   主要是一个class Client(client.HTTPClient)

   先看用法：

第一种用法：获取token+验证token+执行服务

```
from keystoneclient.v2_0 import client
keystone = client.Client(username=USER,
                         password=PASS,
                         tenant_name=TENANT_NAME,
                         auth_url=KEYSTONE_URL)
keystone.tenants.list()
...
user = keystone.users.get(USER_ID)
user.delete()
```

第二种用法：复用token => new_client

```
from keystoneclient.v2_0 import client
keystone = client.Client(username=USER,
                         password=PASS,
                         tenant_name=TENANT_NAME,
                         auth_url=KEYSTONE_URL)
auth_ref = keystone.auth_ref
# pickle or whatever you like here
new_client = client.Client(auth_ref=auth_ref)
```

第三种用法：用admin token

```
from keystoneclient.v2_0 import client
admin_client = client.Client(
    token='12345secret7890',
    endpoint='http://localhost:35357/v2.0')
keystone.tenants.list()
```

在初始化函数里，Client类定义了users，tenants，roles等Manager实例，并完成了获取验证token/endpoint的工作。

接下来，在每一个Manager类的实例中，会通过这个token，向keystone请求服务。Manager类是委托了HttpClient里的get/post等方法。ManagerWithFind类是Manager的子类，添加了Find方法。Resource类则用于解析返回的HTML Reply信息。

3. users模块的结构，其余tenant/role等等类同。

模块：keystoneclient.v2_0.users

类：UserManager(base.ManagerWithFind) => Manager，Manager的初始化函数：

```
def __init__(self, api):
    self.api = api
```

这里的api，就是client实例，在Client类的初始化函数中：

```
def __init__(self, **kwargs):
    """ Initialize a new client for the Keystone v2.0 API. """
    super(Client, self).__init__(**kwargs)
    self.endpoints = endpoints.EndpointManager(self)
    self.roles = roles.RoleManager(self)
    self.services = services.ServiceManager(self)
```

```
self.tenants = tenants.TenantManager(self)
self.tokens = tokens.TokenManager(self)
self.users = users.UserManager(self)
```

再来看kc.users.list() => _list()

```
query = "?" + urllib.urlencode(params)
return self._list("/users%s" % query, "users")
```

kc.users._list()

```
resp, body = self.api.get(url)
```

可以看到最终是由client api向Keystone发起了HTTP请求，并将收到的response返回给Manager类的方法们。

4. user-list工作流分析
    1. 在命令行输入keystone user-list
    2. keystoneclient/shell.py收到请求，执行args.func(self.cs, args)，就会调用到keystoneclient/v2_0/shell.py里的do_user_list方法。其中self.cs是keystoneclient.v2_0.client.Client类的实例，args是解析后的命令行参数。

        ```
        {debug=False, func=<MagicMock id='42654928'>, help=False,  \
        insecure=False, os_auth_url='http://127.0.0.1:5000/v2.0/', \
        os_cacert=None, os_cert='', os_endpoint='', \
        os_identity_api_version='', os_key='', os_password='password\
        ', os_region_name='', os_tenant_id='tenant_id', \
        os_tenant_name='tenant_name', os_token='', os_username=\
        'username'}
        ```

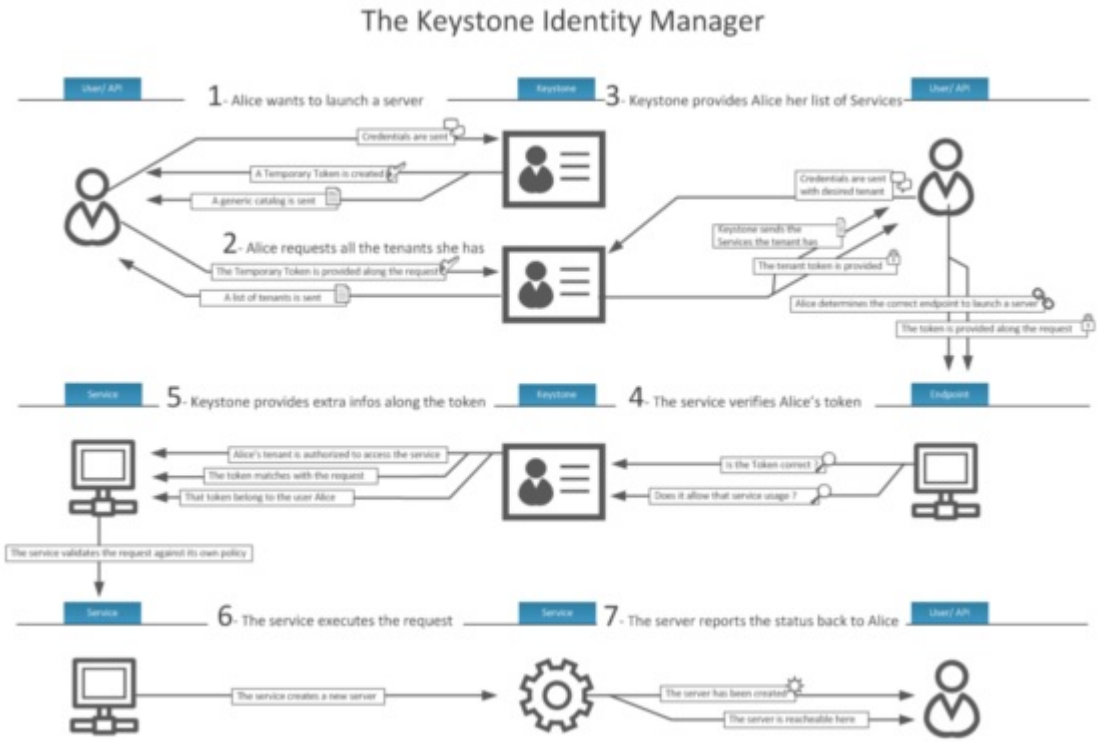
        其中func就是do_user_list方法，我这里是贴了testcase里的输出，testcase里用MagicMock伪造了do_user_list方法。
    3. do_user_list方法只有两行：

        ```
        users = kc.users.list(tenant_id=args.tenant_id)
        utils.print_list(users, ['id', 'name', 'enabled', 'email'])
        ```

        通过keystoneclient.v2_0.client.Client类的实例调用API，然后打印。
    4. 关于Client类和UserManager类的分析见上两节。

Keystone

1. 原理图



原图见官方文档。

用户请求服务（启动一台虚拟机）的步骤如下：

1. 用户向Keystone服务器请求token，Keystone返回token和service catalog
2. 用户拿着token，按service catalog上的endpoint向Keystone请求tenant列表，Keystone验证一下token，通过就返回tenant列表。
3. 用户带着tenant参数向keystone请求服务，keystone返回对应tenant的token和endpoints
4. 用户拿着tenant token，向需要的服务(Nova)对应的endpoint发起请求，对应的服务器(Nova)收到请求后向keystone要求验证：token是否正确，权限是否满足?
5. Keystone告诉服务器(Nova)：用户的tenant有权限请求服务(启动虚拟机)，token正确，token属于用户的tenant。
6. 服务器(Nova)执行服务。
7. 服务器向用户返回。

2. 逻辑结构

Keystone服务器在两个端口监听服务，一个是Admin端口（默认是35357），另一个是用户端口（默认是5000）。

Keystone大致可分为四个模块：

- Token：用来生成和管理token
- Catalog：用来管理service和endpoint
- Identity：用来管理tenant/user/role和验证(用户名密码或者密钥)
- Policy：用来管理policy(访问权限)

这四个模块都有统一的模型接口core.py，然后通过driver(backends)来实现模型接口。

- Token Driver: kvs/memcache/sql
- Catalog Driver: kvs/sql/templated
- Identity Driver: kvs/sql/ldap/pam
- Policy Driver: rules

Keystone基于Paste Deploy部署（通过ini配置文件来管理url和application，不熟悉Paste Deploy的同学要好好研究下ini文件）。

Keystone基于eventlet和greenlet来提供协程，使得REST API支持的请求并发数超越Apache，直追Nginx。

3. 代码结构

主要代码列举如下：

```
.
./catalog
./catalog/backends # catalog driver list
./catalog/backends/kvs.py
./catalog/backends/sql.py
./catalog/backends/templated.py
./catalog/core.py # catalog抽象接口
./clean.py # 做输入的验证和空格处理工作
./cli.py
./common
./common/bufferedhttp.py
./common/cms.py
./common/controller.py
./common/kvs.py
./common/ldap
./common/ldap/core.py
./common/ldap/fakeldap.py
./common/logging.py
./common/manager.py
./common/models.py
./common/openssl.py
./common/policy.py
./common/serializer.py
./common/sql
./common/sql/core.py
./common/sql/legacy.py
./common/sql/migrate_repo
./common/sql/migrate_repo/manage.py
./common/sql/migrate_repo/migrate.cfg
./common/sql/migrate_repo/README
./common/sql/migrate_repo/versions
./common/sql/migrate_repo/versions/002_sqlite_downgrade.sql
./common/sql/migrate_repo/versions/002_sqlite_upgrade.sql
./common/sql/migrate_repo/versions/002_token_id_hash.py
./common/sql/migrate_repo/versions/003_sqlite_downgrade.sql
./common/sql/migrate_repo/versions/003_sqlite_upgrade.sql
./common/sql/migrate_repo/versions/003_token_valid.py
./common/sql/migrate_repo/versions/004_undo_token_id_hash.py
./common/sql/migrate_repo/versions/005_set_utf8_character_set.py
./common/sql/migration.py
./common/sql/nova.py
./common/sql/util.py
./common/systemd.py
./common/utils.py
./common/wsgi.py
./config.py # keystone/openstack/cfg，包括common_cli和log两部分
```

```
./contrib
./contrib/admin_crud
./contrib/admin_crud/core.py
./contrib/ec2
./contrib/ec2/backends
./contrib/ec2/backends/kvs.py
./contrib/ec2/backends/sql.py
./contrib/ec2/core.py
./contrib/s3
./contrib/s3/core.py
./contrib/stats
./contrib/stats/backends
./contrib/stats/backends/kvs.py
./contrib/stats/core.py
./contrib/user_crud
./contrib/user_crud/core.py # CRUD操作抽象接口
./exception.py
./identity
./identity/backends # identity驱动
./identity/backends/kvs.py
./identity/backends/ldap
./identity/backends/ldap/core.py
./identity/backends/pam.py
./identity/backends/sql.py
./identity/core.py
./locale
./locale/keystone.pot
./middleware
./middleware/auth_token.py
./middleware/core.py
./middleware/ec2_token.py
./middleware/s3_token.py
./middleware/swift_auth.py
./openstack
./openstack/common
./openstack/common/cfg.py
./openstack/common/importutils.py
./openstack/common/iniparser.py
./openstack/common/jsonutils.py
./openstack/common/README
./openstack/common/setup.py
./openstack/common/timeutils.py
./policy
./policy/backends # policy驱动
./policy/backends/rules.py
./policy/core.py
./service.py
./test.py
./token
./token/backends # token驱动
./token/backends/kvs.py
./token/backends/memcache.py
./token/backends/sql.py
./token/core.py
```

CRUD是指在做计算处理时的增加、查询（重新得到数据）、更新和删除几个单词的首字母简写。主要被用在描述软件系统中数据库或者持久层的基本操作功能。

4. 代码入口：bin/keystone-all & bin/keystone-manager

   - Config => keystone/config.py
   - keystone-manage

     与keystone service交互来初始化和更新keystone的数据。用于REST API不能完成的任务，如数据的导入导出，schema的迁移。

     ```
     db_sync: Sync the database.
     export_legacy_catalog: Export the service catalog from a \
         legacy database.
     import_legacy: Import a legacy database.
     import_nova_auth: Import a dump of nova auth data \
         into keystone.
     pki_setup: Initialize the certificates used to sign tokens.
     ```

   - keystone-all

     used to run the keystone services

5. 收到请求(user-list)之后的workflow：/etc/keystone/keystone.conf

   - 两个服务端口，各有三个入口

     ```
     [composite:main]
     use = egg:Paste#urlmap
     /v2.0 = public_api
     /v3 = api_v3
     / = public_version_api

     [composite:admin]
     use = egg:Paste#urlmap
     /v2.0 = admin_api
     /v3 = api_v3
     / = admin_version_api
     ```

   - 服务启动：bin/keystone-all

     ```
     servers = []
     servers.append(create_server(CONF.config_file[0],
                                  'admin',
                                  CONF.bind_host,
                                  int(CONF.admin_port)))
     servers.append(create_server(CONF.config_file[0],
                                  'main',
                                  CONF.bind_host,
                                  int(CONF.public_port)))
     serve(*servers)
     ```

   - 请求public version: curl http://localhost:5000

     1. [pipeline:public_version_api]

```
pipeline = stats_monitoring url_normalize xml_body \
    public_version_service
```

2. [app:public_version_service]

```
paste.app_factory = keystone.service:\
    public_version_app_factory
```

return PublicVersionRouter() 返回的是一个PublicVersionRouter对象，=>ComposingRouter=>Router Router是一个WSGI中间件，把incoming的请求map到App(Controller)。 初始化Router时，需要一个mapper对象，并生成一个self._router对象:

```
mapper = routes.Mapper()
version_controller = VersionController('public')
mapper.connect('/', controller=version_controller, \
    action='get_versions')
self._router = routes.middleware.RoutesMiddleware(\
    self._dispatch, self.map)
```

Call这个Router时，会return这个self._router 在request map到一个router之后，会把信息写到req.environ里面，然后由self._router来调用_dispatch，从环境中取到controller，并返回。

```
match = req.environ['wsgiorg.routing_args'][1]
app = match['controller']
return app
```

ComposingRouter，多一个routers变量，在init时，Add routes to given mapper。 还没有实现。

在这里，/ => VersionController('public') => action='get_versions'

3. [filter:xml_body]

```
paste.filter_factory = \
    keystone.middleware:XmlBodyMiddleware.factory
```

keystone.middleware.core.XmlBodyMiddleware Base wsgi.Middleware, Base Application。 wsgi.Middleware，在__call__里面调用初始化时候给它的app。只实现两个方法：process_request和process_response。都是针对request的。

```
response = self.process_request(request)
if response:
    return response
response = request.get_response(self.application)
    return self.process_response(request, response)
```

这个process_request用于Transform the request from XML to JSON。process_response则反之。

4. [filter:url_normalize]

```
paste.filter_factory = \
    keystone.middleware:NormalizingFilter.factory
```

只有process_request，1. 去掉path最后的/，2. 如果path没有，就写/

5. [filter:stats_monitoring]

```
paste.filter_factory = \
    keystone.contrib.stats:StatsMiddleware.factory
```

Monitors various request/response attribute statistics.

- 请求Admin API操作(user-list)

1. [pipeline:admin_api]

```
pipeline = stats_monitoring url_normalize token_auth \
    admin_token_auth xml_body json_body debug \
    stats_reporting ec2_extension s3_extension \
    crud_extension admin_service
```

2. [filter:token_auth]

```
paste.filter_factory = \
    keystone.middleware:TokenAuthMiddleware.factory
```

process_request：把token从request里取出来，放到context里，再放进environ里去。

3. [filter:admin_token_auth]

```
paste.filter_factory = \
    keystone.middleware:AdminTokenAuthMiddleware.factory
```

把token从request里取出来，判断是不是admin token，然后把context['is_admin']放到context里，再放进environ里去。

4. [filter:crud_extension]

```
paste.filter_factory = \
    keystone.contrib.admin_crud:CrudExtension.factory
```

Based ExtensionRouter, Based Router ExtensionRouter: A router that allows extensions to supplement or overwrite routes.

需要实现add_routes方法。

```
def add_routes(self, mapper):
    tenant_controller = identity.TenantController()
    user_controller = identity.UserController()
    role_controller = identity.RoleController()
    service_controller = catalog.ServiceController()
    endpoint_controller = catalog.EndpointController()
```

```
    # 然后一堆map
```

UserController:

```
def get_users(self, context):
    self.assert_admin(context)  #做校验，在get_users里，
        # 如果注释掉这一句，那么user-list用错误的token也可以。
```

return {'users': self.identity_api.list_users(context)} # sql查询去了。

5. [app:admin_service]

```
paste.app_factory = keystone.service:admin_app_factory
```

```
return AdminRouter()
```

/tokens/* & /certificates/* => TokenController()

/tokens => Authentication，用于验证用户名，密码，返回token。

/extensions/* => AdminExtensionsController()

- 请求public API操作

1. pipeline:public_api]

```
pipeline = stats_monitoring url_normalize token_auth \
    admin_token_auth xml_body json_body debug \
    ec2_extension user_crud_extension public_service
```

2. [filter:user_crud_extension]

```
paste.filter_factory = \
    keystone.contrib.user_crud:CrudExtension.factory
```

3. [app:public_service]

```
paste.app_factory = keystone.service:public_app_factory
```

return PublicRouter()