Keystone Documentation

Release 26.0.1.dev9

OpenStack

Dec 12, 2025

CONTENTS

1 Keystone Installation Tutorial **1**

1.1 Keystone Installation Tutorial for openSUSE and SUSE Linux Enterprise 1

1.1.1 Abstract 1

1.1.2 Contents 1

Identity service overview 1

Install and configure 2

Create a domain, projects, users, and roles 6

Verify operation 8

Create OpenStack client environment scripts 10

1.2 Keystone Installation Tutorial for Red Hat Enterprise Linux and CentOS 11

1.2.1 Abstract 11

1.2.2 Contents 12

Identity service overview 12

Install and configure 13

Create a domain, projects, users, and roles 16

Verify operation 18

Create OpenStack client environment scripts 19

1.3 Keystone Installation Tutorial for Ubuntu 21

1.3.1 Abstract 21

1.3.2 Contents 22

Identity service overview 22

Install and configure 22

Create a domain, projects, users, and roles 25

Verify operation 28

Create OpenStack client environment scripts 29

2 Getting Started **33**

2.1 Keystone Architecture 33

2.1.1 Services 33

Identity 33

Resource 34

Assignment 34

Token 35

Catalog 35

2.1.2 Application Construction 35

2.1.3 Service Backends 37

Templated Backend 38

2.1.4 Data Model 38

2.1.5 Approach to CRUD 39

2.1.6 Approach to Authorization (Policy) . . . 39
Rules . . . 39
Capability RBAC . . . 40
2.1.7 Approach to Authentication . . . 40
What is Scope? . . . 42
2.2 Mapping of policy target to API . . . 42
2.3 Getting Involved . . . 47
2.3.1 How to Join the Community . . . 47
#openstack-keystone on OFTC IRC Network . . . 47
Keystone on Launchpad . . . 47
OpenStack Blog . . . 47
Twitter . . . 48

3 Code Documentation **49**
3.1 keystone . . . 49
3.1.1 keystone package . . . 49
Subpackages . . . 49
Submodules . . . 362
Module contents . . . 378

4 Indices and tables **379**

5 Contributor Documentation **381**
5.1 So You Want to Contribute . . . 381
5.1.1 Communication . . . 381
5.1.2 Contacting the Core Team . . . 381
5.1.3 New Feature Planning . . . 381
5.1.4 Task Tracking . . . 381
5.1.5 Reporting a Bug . . . 382
5.1.6 Getting Your Patch Merged . . . 382
Project Team Lead Duties . . . 382
5.2 How Can I Help? . . . 382
5.2.1 The Meaning of Low Hanging Fruit . . . 383
5.3 Setting up Keystone . . . 383
5.3.1 Prerequisites . . . 383
5.3.2 Installing from source . . . 384
5.3.3 Development environment . . . 384
5.3.4 Deploying configuration files . . . 385
Configuring Keystone with a sample file . . . 385
5.3.5 Bootstrapping a test deployment . . . 385
5.3.6 Verifying keystone is set up . . . 385
5.3.7 Database setup . . . 386
5.3.8 Initializing Keystone . . . 386
Initial Sample Data . . . 386
5.3.9 Interacting with Keystone . . . 387
5.4 Identity API v2.0 and v3 History . . . 387
5.4.1 Specifications . . . 387
5.4.2 History . . . 387
5.4.3 How do I migrate from v2.0 to v3? . . . 387
I am a deployer . . . 387
I have a Python client . . . 388
I have a non-Python client . . . 388

5.4.4 Why do I see deployments with Keystone running on two ports? . . . 390
5.4.5 HTTP/1.1 Chunked Encoding . . . 390
5.5 Proposing Features . . . 390
5.5.1 RFE Bug Reports . . . 390
5.5.2 Specifications . . . 391
5.6 Working with Release Notes . . . 391
5.6.1 Release Notes for Bugs . . . 392
5.6.2 Release Notes for Features . . . 392
5.7 Testing Keystone . . . 393
5.7.1 Running Tests . . . 393
Interactive debugging . . . 394
5.7.2 Building the Documentation . . . 394
5.7.3 Tests Structure . . . 394
5.7.4 Testing Schema Migrations . . . 395
5.7.5 LDAP Tests . . . 395
5.7.6 Work in progress Tests . . . 395
5.7.7 API & Scenario Tests . . . 396
Writing new API & Scenario Tests . . . 396
5.8 Developing `doctor` checks . . . 399
5.9 Making an API Change . . . 400
5.9.1 Prerequisites . . . 400
5.9.2 Proposing a change . . . 400
Create . . . 400
Agreement . . . 401
5.9.3 Implementing a change . . . 401
Architectural Recapitulation . . . 401
Changing the SQL Model and Driver . . . 401
Changing the Manager . . . 402
Changing the API Interface . . . 402
5.9.4 Conclusion . . . 403
5.10 Authentication Plugins . . . 403
5.10.1 How to Implement an Authentication Plugin . . . 404
5.11 Database Migrations . . . 404
5.11.1 Writing your own migrations . . . 405
5.12 Identity entity ID management for domain-specific backends . . . 406
5.13 Learning Architecture Internals . . . 406
5.13.1 Caching Layer . . . 406
5.13.2 Filtering responsibilities between API resources and drivers . . . 407
5.13.3 Entity list truncation by drivers . . . 408
5.14 Keystone for Other Services . . . 408
5.14.1 Glossary . . . 408
5.14.2 Domains . . . 409
5.14.3 The default domain . . . 409
5.14.4 Authorization Scopes . . . 410
System Scope . . . 410
Domain Scope . . . 410
Project Scope . . . 411
Unscoped . . . 411
5.14.5 Why are authorization scopes important? . . . 411
Flexibility for exposing your work . . . 411
Less custom code . . . 412

Reusable default roles . . . 412
5.14.6 How do I incorporate authorization scopes into a service? . . . 414
Ruthless Testing . . . 414
Auditing the API . . . 415
Setting scope types . . . 415
Rewriting check string . . . 416
Communication . . . 416
5.14.7 Auth Token middleware . . . 416
Service tokens . . . 416
5.14.8 Picking the version . . . 416
5.14.9 Hierarchical Multitenancy . . . 417
5.15 Developing Keystone Drivers . . . 417
5.15.1 In/Out of Tree . . . 417
5.15.2 How To Make a Driver . . . 417
5.15.3 Identity Driver Configuration . . . 418
Defining configuration options . . . 418
Allowing domain configuration per API . . . 418
5.15.4 Driver Interface Changes . . . 419
Removing Methods . . . 419
Adding Methods . . . 419
Updating Methods . . . 419
5.16 Service Catalog Overview . . . 419
5.16.1 An example service catalog . . . 420
5.16.2 Services . . . 420
5.16.3 Endpoints . . . 421
5.17 Technical Vision for Keystone . . . 422
5.17.1 Mission Statement . . . 422
5.17.2 Vision for OpenStack . . . 422
Self-service . . . 422
Application Control . . . 422
Interoperability . . . 422
Bidirectional Compatibility . . . 422
Partitioning . . . 423
Basic Physical Data Center Management . . . 423
Plays Well With Others . . . 423
Customizable Integration . . . 423
Graphical User Interface . . . 423
Secure by Design . . . 423
5.18 Programming Exercises for Interns and New Contributors . . . 424
5.18.1 Add a Parameter to an API . . . 424
5.18.2 Write an External Driver . . . 424
5.18.3 Write an Auth Plugin . . . 425

6 User Documentation **427**
6.1 Supported clients . . . 427
6.1.1 Authenticating with a Password via CLI . . . 427
6.2 Application Credentials . . . 428
6.2.1 Managing Application Credentials . . . 428
6.2.2 Access Rules . . . 432
6.2.3 Using Application Credentials . . . 434
6.2.4 Rotating Application Credentials . . . 434

6.2.5 Frequently Asked Questions . . . 435
Why is the application credential owned by the user rather than the project? . . . 435
6.3 Trusts . . . 435
6.4 API Discovery with JSON Home . . . 436
6.4.1 What is JSON Home? . . . 436
6.4.2 Requesting JSON Home Documents . . . 436
6.5 API Examples using Curl . . . 437
6.5.1 v3 API Examples Using Curl . . . 437
GET / . . . 437
Tokens . . . 439
Domains . . . 450
Projects . . . 451
GET /v3/services . . . 453
GET /v3/endpoints . . . 453
Users . . . 454
PUT /v3/projects/{project_id}/groups/{group_id}/roles/{role_id} . . . 456
POST /v3/OS-TRUST/trusts . . . 457
6.6 Multi-Factor Authentication . . . 458
6.6.1 Configuring MFA . . . 458
6.6.2 Using MFA . . . 458
Single step . . . 458
Multi-Step . . . 459

7 CLI Documentation **461**
7.1 keystone-manage . . . 461
7.1.1 Keystone Management Utility . . . 461
SYNOPSIS . . . 461
DESCRIPTION . . . 461
USAGE . . . 462
OPTIONS . . . 463
FILES . . . 464
SEE ALSO . . . 464
SOURCE . . . 464
7.2 keystone-status . . . 464
7.2.1 Keystone Status Utility . . . 464
SYNOPSIS . . . 465
DESCRIPTION . . . 465
USAGE . . . 465
OPTIONS . . . 466
FILES . . . 466
SEE ALSO . . . 466
SOURCE . . . 466

8 Administrator Guides **467**
8.1 Getting Started . . . 467
8.1.1 Identity concepts . . . 467
User management . . . 468
Service management . . . 471
Groups . . . 472
8.1.2 Configuring Keystone . . . 472
Identity sources . . . 472
8.1.3 Bootstrapping Identity . . . 474

Using the CLI . . . 475
Using a shared secret . . . 476
8.1.4 Manage projects, users, and roles . . . 476
Projects . . . 477
Users . . . 479
Roles and role assignments . . . 480
8.1.5 Create and manage services and service users . . . 483
Service Catalog . . . 483
Service users . . . 487
8.2 Keystone Configuration . . . 488
8.2.1 Troubleshoot the Identity service . . . 489
8.2.2 Logging . . . 489
8.2.3 Domain-specific configuration . . . 489
Enable drivers for domain-specific configuration files . . . 489
Enable drivers for storing configuration options in SQL database . . . 490
Public ID Generators . . . 492
Migrate domain-specific configuration files to the SQL database . . . 493
8.2.4 Integrate Identity with LDAP . . . 493
Identity LDAP server set up . . . 493
Integrate Identity back end with LDAP . . . 496
Secure the OpenStack Identity service connection to an LDAP back end . . . 500
8.2.5 Caching layer . . . 501
Caching for tokens and tokens validation . . . 503
Caching for non-token resources . . . 503
Cache invalidation . . . 504
Configure the Memcached back end example . . . 504
Verbose cache logging . . . 504
8.2.6 Security compliance and PCI-DSS . . . 505
Setting an account lockout threshold . . . 505
Disabling inactive users . . . 505
Force users to change password upon first use . . . 506
Configuring password expiration . . . 506
Configuring password strength requirements . . . 506
Requiring a unique password history . . . 507
Prevent Self-Service Password Changes . . . 508
8.2.7 Performance and scaling . . . 508
Keystone configuration options that affect performance . . . 508
Keystonemiddleware configuration options that affect performance . . . 508
8.2.8 URL safe naming of projects and domains . . . 509
8.2.9 Limiting list return size . . . 509
8.2.10 Endpoint Filtering . . . 510
8.2.11 Endpoint Policy . . . 510
8.3 Keystone Operations . . . 510
8.3.1 Upgrading Keystone . . . 510
Before you begin . . . 511
Upgrading with downtime . . . 511
Upgrading with minimal downtime . . . 511
Upgrading without downtime . . . 512
8.3.2 Case-Insensitivity in keystone . . . 514
Resources in keystone . . . 514
Backends . . . 516

8.3.3 Managing trusts . . . 516
Removing Expired Trusts . . . 516
8.4 All about keystone tokens . . . 517
8.4.1 Keystone tokens . . . 517
Authorization scopes . . . 517
Token providers . . . 518
8.4.2 Fernet - Frequently Asked Questions . . . 521
What is a fernet token? . . . 521
What is a fernet key? . . . 521
What are the different types of keys? . . . 521
So, how does a staged key help me and why do I care about it? . . . 522
Where do I put my key repository? . . . 522
What is the recommended way to rotate and distribute keys? . . . 523
Do fernet tokens still expire? . . . 523
Why should I choose fernet tokens over UUID tokens? . . . 523
Why should I choose fernet tokens over PKI or PKIZ tokens? . . . 523
Should I rotate and distribute keys from the same keystone node every rotation? . . . 523
How do I add new keystone nodes to a deployment? . . . 524
How should I approach key distribution? . . . 524
How long should I keep my keys around? . . . 525
Is a fernet token still a bearer token? . . . 525
What if I need to revoke all my tokens? . . . 525
What can an attacker do if they compromise a fernet key in my deployment? . . . 525
I rotated keys and now tokens are invalidating early, what did I do? . . . 525
8.4.3 JWS key rotation . . . 527
Initial setup . . . 528
Continued operations . . . 528
8.4.4 Token provider . . . 529
8.5 Default Roles . . . 529
8.5.1 Primer . . . 529
8.5.2 Roles Definitions . . . 530
Reader . . . 530
Member . . . 531
Manager . . . 531
Admin . . . 532
Service . . . 532
8.5.3 System Personas . . . 532
System Administrators . . . 533
System Members & System Readers . . . 534
8.5.4 Domain Personas . . . 534
Domain Administrators . . . 535
Domain Managers . . . 535
Domain Members & Domain Readers . . . 536
8.5.5 Project Personas . . . 536
Project Administrators . . . 537
Project Members & Project Readers . . . 537
8.5.6 Writing Policies . . . 538
8.6 Advanced Keystone Features . . . 538
8.6.1 Unified Limits . . . 538
What is a limit? . . . 539
Limits and usage . . . 540

Enforcement models . . . 541
8.6.2 Resource Options . . . 550
User Options . . . 550
Role Options . . . 553
Project Options . . . 554
Domain Options . . . 554
8.6.3 Credential Encryption . . . 555
Configuring credential encryption . . . 555
How credential encryption works . . . 555
Encryption key management . . . 556
8.6.4 Health Check . . . 556
8.6.5 Keystone Event Notifications . . . 556
Common Notification Structure . . . 557
Auditing with CADF . . . 557
Basic Notifications . . . 564
Recommendations for consumers . . . 565
Opting out of certain notifications . . . 565
8.7 Authentication Mechanisms . . . 565
8.7.1 Multi-Factor Authentication . . . 565
Configuring MFA . . . 565
Using MFA . . . 566
Supported multi-factor authentication methods . . . 567
8.7.2 Time-based One-time Password (TOTP) . . . 567
Configuring TOTP . . . 567
Authenticate with TOTP . . . 568
8.7.3 Federated Identity . . . 569
Introduction to Keystone Federation . . . 569
Configuring Keystone for Federation . . . 580
Mapping Combinations . . . 603
8.7.4 Using external authentication with Keystone . . . 621
Configuration . . . 622
Using HTTPD authentication . . . 622
8.7.5 Configuring Keystone for Tokenless Authorization . . . 623
Definitions . . . 623
Keystone Configuration . . . 626
Setup Mapping . . . 628
SSL Terminator Configuration . . . 630
Setup `auth_token` middleware . . . 632
8.7.6 OAuth1 1.0a . . . 632
8.8 OAuth2.0 Client Credentials Grant Flow . . . 633
8.8.1 Overview . . . 633
8.8.2 Guide . . . 633
Enable application credentials authentication . . . 633
Try to access the Keystone APIs . . . 634
8.9 Configure HTTPS in Identity Service . . . 635

9 Keystone Configuration Options **637**
9.1 API Configuration options . . . 637
9.1.1 Configuration . . . 637
DEFAULT . . . 637
application_credential . . . 648

assignment . . . 649
auth . . . 649
cache . . . 651
catalog . . . 659
cors . . . 660
credential . . . 661
database . . . 662
domain_config . . . 666
endpoint_filter . . . 667
endpoint_policy . . . 667
federation . . . 668
fernet_receipts . . . 670
fernet_tokens . . . 671
healthcheck . . . 671
identity . . . 673
identity_mapping . . . 676
jwt_tokens . . . 677
ldap . . . 678
oauth1 . . . 688
oauth2 . . . 689
oslo_messaging_amqp . . . 689
oslo_messaging_kafka . . . 698
oslo_messaging_notifications . . . 702
oslo_messaging_rabbit . . . 703
oslo_middleware . . . 711
oslo_policy . . . 712
policy . . . 714
profiler . . . 715
profiler_jaeger . . . 718
profiler_otlp . . . 718
receipt . . . 718
resource . . . 720
revoke . . . 722
role . . . 723
saml . . . 723
security_compliance . . . 727
shadow_users . . . 729
token . . . 730
tokenless_auth . . . 732
totp . . . 733
trust . . . 733
unified_limit . . . 734
wsgi . . . 735
9.1.2 Domain-specific Identity drivers . . . 735
9.2 Policy configuration . . . 735
9.2.1 Configuration . . . 735
keystone . . . 736

CHAPTER

ONE

KEYSTONE INSTALLATION TUTORIAL

The OpenStack system consists of several key services that are separately installed. These services work together depending on your cloud needs and include the Compute, Identity, Networking, Image, Block Storage, Object Storage, Telemetry, Orchestration, and Database services. You can install any of these projects separately and configure them stand-alone or as connected entities.

This section describes how to install and configure the OpenStack Identity service, code-named keystone, on the controller node. For scalability purposes, this configuration deploys Fernet tokens and the Apache HTTP server to handle requests.

1.1 Keystone Installation Tutorial for openSUSE and SUSE Linux Enterprise

1.1.1 Abstract

This guide will show you how to install OpenStack by using packages on openSUSE Leap 42.2 and SUSE Linux Enterprise Server 12 - for both SP1 and SP2 - through the Open Build Service Cloud repository.

Explanations of configuration options and sample configuration files are included.

Warning

This guide is a work-in-progress and is subject to updates frequently. Pre-release packages have been used for testing, and some instructions may not work with final versions. Please help us make this guide better by reporting any errors you encounter.

1.1.2 Contents

Identity service overview

The OpenStack Identity service provides a single point of integration for managing authentication, authorization, and a catalog of services.

The Identity service is typically the first service a user interacts with. Once authenticated, an end user can use their identity to access other OpenStack services. Likewise, other OpenStack services leverage the Identity service to ensure users are who they say they are and discover where other services are within the deployment. The Identity service can also integrate with some external user management systems (such as LDAP).

Users and services can locate other services by using the service catalog, which is managed by the Identity service. As the name implies, a service catalog is a collection of available services in an OpenStack deployment. Each service can have one or many endpoints and each endpoint can be one of three types: admin, internal, or public. In a production environment, different endpoint types might reside on separate networks exposed to different types of users for security reasons. For instance, the public API network might be visible from the Internet so customers can manage their clouds. The admin API network might be restricted to operators within the organization that manages cloud infrastructure. The internal API network might be restricted to the hosts that contain OpenStack services. Also, OpenStack supports multiple regions for scalability. For simplicity, this guide uses the management network for all endpoint types and the default `RegionOne` region. Together, regions, services, and endpoints created within the Identity service comprise the service catalog for a deployment. Each OpenStack service in your deployment needs a service entry with corresponding endpoints stored in the Identity service. This can all be done after the Identity service has been installed and configured.

The Identity service contains these components:

Server
: A centralized server provides authentication and authorization services using a RESTful interface.

Drivers
: Drivers or a service back end are integrated to the centralized server. They are used for accessing identity information in repositories external to OpenStack, and may already exist in the infrastructure where OpenStack is deployed (for example, SQL databases or LDAP servers).

Modules
: Middleware modules run in the address space of the OpenStack component that is using the Identity service. These modules intercept service requests, extract user credentials, and send them to the centralized server for authorization. The integration between the middleware modules and OpenStack components uses the Python Web Server Gateway Interface.

Install and configure

This section describes how to install and configure the OpenStack Identity service, code-named keystone, on the controller node. For scalability purposes, this configuration deploys Fernet tokens and the Apache HTTP server to handle requests.

> **Note**
>
> Ensure that you have completed the prerequisite installation steps in the Openstack Install Guide before proceeding.

Prerequisites

Before you install and configure the Identity service, you must create a database.

> **Note**
>
> Before you begin, ensure you have the most recent version of `python-pyasn1` installed.

1. Use the database access client to connect to the database server as the `root` user:

```
$ mysql -u root -p
```

2. Create the `keystone` database:

```
MariaDB [(none)]> CREATE DATABASE keystone;
```

3. Grant proper access to the `keystone` database:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@
↪'localhost' \
IDENTIFIED BY 'KEYSTONE_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
IDENTIFIED BY 'KEYSTONE_DBPASS';
```

   Replace `KEYSTONE_DBPASS` with a suitable password.

4. Exit the database access client.

Install and configure components

Note

Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

Note

Starting with the Newton release, SUSE OpenStack packages are shipping with the upstream default configuration files. For example `/etc/keystone/keystone.conf`, with customizations in `/etc/keystone/keystone.conf.d/010-keystone.conf`. While the following instructions modify the default configuration file, adding a new file in `/etc/keystone/keystone.conf.d` achieves the same result.

1. Run the following command to install the packages:

```
# zypper install openstack-keystone apache2 apache2-mod_wsgi
```

2. Edit the `/etc/keystone/keystone.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

```
[database]
# ...
connection = mysql+pymysql://keystone:KEYSTONE_DBPASS@controller/
↪keystone
```

     Replace `KEYSTONE_DBPASS` with the password you chose for the database.

> **Note**
>
> Comment out or remove any other `connection` options in the `[database]` section.

- In the `[token]` section, configure the Fernet token provider:

```
[token]
# ...
provider = fernet
```

3. Populate the Identity service database:

```
# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

4. Initialize Fernet key repositories:

> **Note**
>
> The `--keystone-user` and `--keystone-group` flags are used to specify the operating systems user/group that will be used to run keystone. These are provided to allow running keystone under another operating system user/group. In the example below, we call the user & group `keystone`.

```
# keystone-manage fernet_setup --keystone-user keystone --keystone-group␣
↪keystone
# keystone-manage credential_setup --keystone-user keystone --keystone-
↪group keystone
```

5. Bootstrap the Identity service:

> **Note**
>
> Before the Queens release, keystone needed to be run on two separate ports to accommodate the Identity v2 API which ran a separate admin-only service commonly on port 35357. With the removal of the v2 API, keystone can be run on the same port for all interfaces.

```
# keystone-manage bootstrap --bootstrap-password ADMIN_PASS \
  --bootstrap-admin-url http://controller:5000/v3/ \
  --bootstrap-internal-url http://controller:5000/v3/ \
  --bootstrap-public-url http://controller:5000/v3/ \
  --bootstrap-region-id RegionOne
```

Replace `ADMIN_PASS` with a suitable password for an administrative user.

Configure the Apache HTTP server

1. Edit the `/etc/sysconfig/apache2` file and configure the `APACHE_SERVERNAME` option to reference the controller node:

```
APACHE_SERVERNAME="controller"
```

   The `APACHE_SERVERNAME` entry will need to be added if it does not already exist.

2. Create the `/etc/apache2/conf.d/wsgi-keystone.conf` file with the following content:

```
Listen 5000

<VirtualHost *:5000>
    WSGIDaemonProcess keystone-public processes=5 threads=1 user=keystone␣
↪group=keystone display-name=%{GROUP}
    WSGIProcessGroup keystone-public
    WSGIScriptAlias / /usr/bin/keystone-wsgi-public
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    ErrorLogFormat "%{cu}t %M"
    ErrorLog /var/log/apache2/keystone.log
    CustomLog /var/log/apache2/keystone_access.log combined

    <Directory /usr/bin>
        Require all granted
    </Directory>
</VirtualHost>
```

3. Recursively change the ownership of the `/etc/keystone` directory:

```
# chown -R keystone:keystone /etc/keystone
```

SSL

A secure deployment should have the web server configured to use SSL or running behind an SSL terminator.

Finalize the installation

1. Start the Apache HTTP service and configure it to start when the system boots:

```
# systemctl enable apache2.service
# systemctl start apache2.service
```

2. Configure the administrative account by setting the proper environmental variables:

```
$ export OS_USERNAME=admin
$ export OS_PASSWORD=ADMIN_PASS
$ export OS_PROJECT_NAME=admin
```

(continues on next page)

(continued from previous page)

```
$ export OS_USER_DOMAIN_NAME=Default
$ export OS_PROJECT_DOMAIN_NAME=Default
$ export OS_AUTH_URL=http://controller:5000/v3
$ export OS_IDENTITY_API_VERSION=3
```

These values shown here are the default ones created from `keystone-manage bootstrap`.

Replace `ADMIN_PASS` with the password used in the `keystone-manage bootstrap` command in *keystone-install-configure-obs*.

Create a domain, projects, users, and roles

The Identity service provides authentication services for each OpenStack service. The authentication service uses a combination of domains, projects, users, and roles.

1. Although the default domain already exists from the *keystone-manage bootstrap* step in this guide, a formal way to create a new domain would be:

```
$ openstack domain create --description "An Example Domain" example

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | An Example Domain                |
| enabled     | True                             |
| id          | 2f4f80574fd84fe6ba9067228ae0a50c |
| name        | example                          |
| tags        | []                               |
+-------------+----------------------------------+
```

2. This guide uses a service project that contains a unique user for each service that you add to your environment. Create the `service` project:

```
$ openstack project create --domain default \
  --description "Service Project" service

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Service Project                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 24ac7f19cd944f4cba1d77469b2a73ed |
| is_domain   | False                            |
| name        | service                          |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

3. Regular (non-admin) tasks should use an unprivileged project and user. As an example, this guide creates the `myproject` project and `myuser` user.

- Create the `myproject` project:

```
$ openstack project create --domain default \
  --description "Demo Project" myproject

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Demo Project                     |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 231ad6e7ebba47d6a1e57e1cc07ae446 |
| is_domain   | False                            |
| name        | myproject                        |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

Note

Do not repeat this step when creating additional users for this project.

- Create the `myuser` user:

```
$ openstack user create --domain default \
  --password-prompt myuser

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | aeda23aa78f44e859900e22c24817832 |
| name                | myuser                           |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

- Create the `myrole` role:

```
$ openstack role create myrole

+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 997ce8d05fc143ac97d83fdfb5998552 |
| name      | myrole                           |
```

(continues on next page)

(continued from previous page)

```
+-----------+----------------------------------+
```

- Add the `myrole` role to the `myproject` project and `myuser` user:

```
$ openstack role add --project myproject --user myuser myrole
```

Note

This command provides no output.

Note

You can repeat this procedure to create additional projects and users.

Verify operation

Verify operation of the Identity service before installing other services.

Note

Perform these commands on the controller node.

1. Unset the temporary `OS_AUTH_URL` and `OS_PASSWORD` environment variable:

```
$ unset OS_AUTH_URL OS_PASSWORD
```

2. As the `admin` user, request an authentication token:

```
$ openstack --os-auth-url http://controller:5000/v3 \
  --os-project-domain-name Default --os-user-domain-name Default \
  --os-project-name admin --os-username admin token issue

Password:
+------------+-----------------------------------------------------------
↪-----+
| Field      | Value                                                     ␣
↪     |
+------------+-----------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:14:07.056119Z                               ␣
↪     |
| id         | gAAAAABWvi7_B8kKQD9wdXac8MoZiQldmjEO643d-e_j-
↪XXq9AmIegIbA7UHGPv |
|            | atnN21qtOMjCFWX7BReJEQnVOAj3nclRQgAYRsfSU_
↪MrsuWb4EDtnjU7HEpoBb4 |
|            | o6ozsA_NmFWEpLeKy0uNn_WeKbAhYygrsmQGA49dclHVnz-OMVLiyM9ws ␣
↪     |
```

(continues on next page)

(continued from previous page)

```
| project_id | 343d245e850143a096806dfaefa9afdc                                 ␣
↪     |
| user_id    | ac33776331494012 96f6c0d92d79dc16                                ␣
↪     |
+------------+------------------------------------------------------------------
↪-----+
```

Note

This command uses the password for the `admin` user.

3. As the `myuser` user created in the previous section, request an authentication token:

```
$ openstack --os-auth-url http://controller:5000/v3 \
  --os-project-domain-name Default --os-user-domain-name Default \
  --os-project-name myproject --os-username myuser token issue

Password:
+------------+------------------------------------------------------------------
↪-----+
| Field      | Value                                                            ␣
↪     |
+------------+------------------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:15:39.014479Z                                      ␣
↪     |
| id         | gAAAAABWvi9bsh7vkiby5BpCCnc-JkbGhm9wH3fabS_cY7uabOubesi-
↪Me6IGWW |
|            | yQqNegDDZ5jw7grI26vvgy1J5nCVwZ_zFRqPiz_
↪qhbq29mgbQLglbkq6FQvzBRQ |
|            | JcOzq3uwhzNxszJWmzGC7rJE_H0A_a3UFhqv8M4zMRYSbS2YF0MyFmp_U         ␣
↪     |
| project_id | ed0b60bf607743088218b0a533d5943f                                 ␣
↪     |
| user_id    | 58126687cbcc4888bfa9ab73a2256f27                                 ␣
↪     |
+------------+------------------------------------------------------------------
↪-----+
```

Create OpenStack client environment scripts

The previous sections used a combination of environment variables and command options to interact with the Identity service via the `openstack` client. To increase efficiency of client operations, OpenStack supports simple client environment scripts also known as OpenRC files. These scripts typically contain common options for all clients, but also support unique options. For more information, see the OpenStack End User Guide.

Creating the scripts

Create client environment scripts for the `admin` and `demo` projects and users. Future portions of this guide reference these scripts to load appropriate credentials for client operations.

Note

The paths of the client environment scripts are unrestricted. For convenience, you can place the scripts in any location, however ensure that they are accessible and located in a secure place appropriate for your deployment, as they do contain sensitive credentials.

1. Create and edit the `admin-openrc` file and add the following content:

   Note

   The OpenStack client also supports using a `clouds.yaml` file. For more information, see the os-client-config.

   ```
   export OS_PROJECT_DOMAIN_NAME=Default
   export OS_USER_DOMAIN_NAME=Default
   export OS_PROJECT_NAME=admin
   export OS_USERNAME=admin
   export OS_PASSWORD=ADMIN_PASS
   export OS_AUTH_URL=http://controller:5000/v3
   export OS_IDENTITY_API_VERSION=3
   export OS_IMAGE_API_VERSION=2
   ```

   Replace `ADMIN_PASS` with the password you chose for the `admin` user in the Identity service.

2. Create and edit the `demo-openrc` file and add the following content:

   ```
   export OS_PROJECT_DOMAIN_NAME=Default
   export OS_USER_DOMAIN_NAME=Default
   export OS_PROJECT_NAME=myproject
   export OS_USERNAME=myuser
   export OS_PASSWORD=DEMO_PASS
   export OS_AUTH_URL=http://controller:5000/v3
   export OS_IDENTITY_API_VERSION=3
   export OS_IMAGE_API_VERSION=2
   ```

   Replace `DEMO_PASS` with the password you chose for the `demo` user in the Identity service.

Using the scripts

To run clients as a specific project and user, you can simply load the associated client environment script prior to running them. For example:

1. Load the `admin-openrc` file to populate environment variables with the location of the Identity service and the `admin` project and user credentials:

```
$ . admin-openrc
```

2. Request an authentication token:

```
$ openstack token issue

+------------+------------------------------------------------------------
↪-----+
| Field      | Value                                                      ␣
↪     |
+------------+------------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:44:35.659723Z                                ␣
↪     |
| id         | gAAAAABWvjYj-
↪Zjfg8WXFaQnUd1DMYTBVrKw4h3fIagi5NoEmh21U72SrRv2trl |
|            | JWFYhLi2_uPR31Igf6A8mH2Rw9kv_bxNo1jbLNPLGzW_
↪u5FC7InFqx0yYtTwa1e |
|            | eq2b0f6-18KZyQhs7F3teAta143kJEWuNEYET-y7u29y0be1_64KYkM7E ␣
↪     |
| project_id | 343d245e850143a096806dfaefa9afdc                           ␣
↪     |
| user_id    | ac33776331494012961f6c0d92d79dc16                          ␣
↪     |
+------------+------------------------------------------------------------
↪-----+
```

1.2 Keystone Installation Tutorial for Red Hat Enterprise Linux and CentOS

1.2.1 Abstract

This guide will show you how to install Keystone by using packages available on Red Hat Enterprise Linux 8 and 9 and their derivatives through the RDO repository.

Explanations of configuration options and sample configuration files are included.

Warning

This guide is a work-in-progress and is subject to updates frequently. Pre-release packages have been used for testing, and some instructions may not work with final versions. Please help us make this

guide better by reporting any errors you encounter.

1.2.2 Contents

Identity service overview

The OpenStack Identity service provides a single point of integration for managing authentication, authorization, and a catalog of services.

The Identity service is typically the first service a user interacts with. Once authenticated, an end user can use their identity to access other OpenStack services. Likewise, other OpenStack services leverage the Identity service to ensure users are who they say they are and discover where other services are within the deployment. The Identity service can also integrate with some external user management systems (such as LDAP).

Users and services can locate other services by using the service catalog, which is managed by the Identity service. As the name implies, a service catalog is a collection of available services in an OpenStack deployment. Each service can have one or many endpoints and each endpoint can be one of three types: admin, internal, or public. In a production environment, different endpoint types might reside on separate networks exposed to different types of users for security reasons. For instance, the public API network might be visible from the Internet so customers can manage their clouds. The admin API network might be restricted to operators within the organization that manages cloud infrastructure. The internal API network might be restricted to the hosts that contain OpenStack services. Also, OpenStack supports multiple regions for scalability. For simplicity, this guide uses the management network for all endpoint types and the default `RegionOne` region. Together, regions, services, and endpoints created within the Identity service comprise the service catalog for a deployment. Each OpenStack service in your deployment needs a service entry with corresponding endpoints stored in the Identity service. This can all be done after the Identity service has been installed and configured.

The Identity service contains these components:

Server
: A centralized server provides authentication and authorization services using a RESTful interface.

Drivers
: Drivers or a service back end are integrated to the centralized server. They are used for accessing identity information in repositories external to OpenStack, and may already exist in the infrastructure where OpenStack is deployed (for example, SQL databases or LDAP servers).

Modules
: Middleware modules run in the address space of the OpenStack component that is using the Identity service. These modules intercept service requests, extract user credentials, and send them to the centralized server for authorization. The integration between the middleware modules and OpenStack components uses the Python Web Server Gateway Interface.

Install and configure

This section describes how to install and configure the OpenStack Identity service, code-named keystone, on the controller node. For scalability purposes, this configuration deploys Fernet tokens and the Apache HTTP server to handle requests.

Note

Ensure that you have completed the prerequisite installation steps in the Openstack Install Guide before proceeding.

Prerequisites

Before you install and configure the Identity service, you must create a database.

1. Use the database access client to connect to the database server as the `root` user:

```
$ mysql -u root -p
```

2. Create the `keystone` database:

```
MariaDB [(none)]> CREATE DATABASE keystone;
```

3. Grant proper access to the `keystone` database:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@
↪'localhost' \
IDENTIFIED BY 'KEYSTONE_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
IDENTIFIED BY 'KEYSTONE_DBPASS';
```

   Replace `KEYSTONE_DBPASS` with a suitable password.

4. Exit the database access client.

Install and configure components

Note

Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

1. Run the following command to install the packages:

```
# dnf install openstack-keystone httpd python3-mod_wsgi
```

2. Edit the `/etc/keystone/keystone.conf` file and complete the following actions:
   - In the `[database]` section, configure database access:

```
[database]
# ...
connection = mysql+pymysql://keystone:KEYSTONE_DBPASS@controller/
↪keystone
```

Replace `KEYSTONE_DBPASS` with the password you chose for the database.

Note

Comment out or remove any other `connection` options in the `[database]` section.

Note

The host, `controller` in this example, must be resolvable.

- In the `[token]` section, configure the Fernet token provider:

```
[token]
# ...
provider = fernet
```

3. Populate the Identity service database:

```
# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

4. Initialize Fernet key repositories:

Note

The `--keystone-user` and `--keystone-group` flags are used to specify the operating systems user/group that will be used to run keystone. These are provided to allow running keystone under another operating system user/group. In the example below, we call the user & group `keystone`.

```
# keystone-manage fernet_setup --keystone-user keystone --keystone-group␣
↪keystone
# keystone-manage credential_setup --keystone-user keystone --keystone-
↪group keystone
```

5. Bootstrap the Identity service:

Note

Before the Queens release, keystone needed to be run on two separate ports to accommodate the Identity v2 API which ran a separate admin-only service commonly on port 35357. With the removal of the v2 API, keystone can be run on the same port for all interfaces.

```
# keystone-manage bootstrap --bootstrap-password ADMIN_PASS \
  --bootstrap-admin-url http://controller:5000/v3/ \
  --bootstrap-internal-url http://controller:5000/v3/ \
  --bootstrap-public-url http://controller:5000/v3/ \
  --bootstrap-region-id RegionOne
```

Replace `ADMIN_PASS` with a suitable password for an administrative user.

Configure the Apache HTTP server

1. Edit the `/etc/httpd/conf/httpd.conf` file and configure the `ServerName` option to reference the controller node:

   ```
   ServerName controller
   ```

   The `ServerName` entry will need to be added if it does not already exist.

2. Create a link to the `/usr/share/keystone/wsgi-keystone.conf` file:

   ```
   # ln -s /usr/share/keystone/wsgi-keystone.conf /etc/httpd/conf.d/
   ```

SSL

A secure deployment should have the web server configured to use SSL or running behind an SSL terminator.

Finalize the installation

1. Start the Apache HTTP service and configure it to start when the system boots:

   ```
   # systemctl enable httpd.service
   # systemctl start httpd.service
   ```

2. Configure the administrative account by setting the proper environmental variables:

   ```
   $ export OS_USERNAME=admin
   $ export OS_PASSWORD=ADMIN_PASS
   $ export OS_PROJECT_NAME=admin
   $ export OS_USER_DOMAIN_NAME=Default
   $ export OS_PROJECT_DOMAIN_NAME=Default
   $ export OS_AUTH_URL=http://controller:5000/v3
   $ export OS_IDENTITY_API_VERSION=3
   ```

   These values shown here are the default ones created from `keystone-manage bootstrap`.

   Replace `ADMIN_PASS` with the password used in the `keystone-manage bootstrap` command in *keystone-install-configure-rdo*.

Create a domain, projects, users, and roles

The Identity service provides authentication services for each OpenStack service. The authentication service uses a combination of domains, projects, users, and roles.

1. Although the default domain already exists from the *keystone-manage bootstrap* step in this guide, a formal way to create a new domain would be:

```
$ openstack domain create --description "An Example Domain" example

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | An Example Domain                |
| enabled     | True                             |
| id          | 2f4f80574fd84fe6ba9067228ae0a50c |
| name        | example                          |
| tags        | []                               |
+-------------+----------------------------------+
```

2. This guide uses a service project that contains a unique user for each service that you add to your environment. Create the `service` project:

```
$ openstack project create --domain default \
  --description "Service Project" service

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Service Project                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 24ac7f19cd944f4cba1d77469b2a73ed |
| is_domain   | False                            |
| name        | service                          |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

3. Regular (non-admin) tasks should use an unprivileged project and user. As an example, this guide creates the `myproject` project and `myuser` user.

   - Create the `myproject` project:

```
$ openstack project create --domain default \
  --description "Demo Project" myproject

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Demo Project                     |
| domain_id   | default                          |
```

(continues on next page)

(continued from previous page)

```
| enabled     | True                             |
| id          | 231ad6e7ebba47d6a1e57e1cc07ae446 |
| is_domain   | False                            |
| name        | myproject                        |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

Note

Do not repeat this step when creating additional users for this project.

- Create the `myuser` user:

```
$ openstack user create --domain default \
  --password-prompt myuser

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | aeda23aa78f44e859900e22c24817832 |
| name                | myuser                           |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

- Create the `myrole` role:

```
$ openstack role create myrole

+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 997ce8d05fc143ac97d83fdfb5998552 |
| name      | myrole                           |
+-----------+----------------------------------+
```

- Add the `myrole` role to the `myproject` project and `myuser` user:

```
$ openstack role add --project myproject --user myuser myrole
```

Note

This command provides no output.

Note

You can repeat this procedure to create additional projects and users.

Verify operation

Verify operation of the Identity service before installing other services.

Note

Perform these commands on the controller node.

1. Unset the temporary `OS_AUTH_URL` and `OS_PASSWORD` environment variable:

```
$ unset OS_AUTH_URL OS_PASSWORD
```

2. As the `admin` user, request an authentication token:

```
$ openstack --os-auth-url http://controller:5000/v3 \
  --os-project-domain-name Default --os-user-domain-name Default \
  --os-project-name admin --os-username admin token issue

Password:
+------------+-----------------------------------------------------------
↪-----+
| Field      | Value                                                     ␣
↪     |
+------------+-----------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:14:07.056119Z                               ␣
↪     |
| id         | gAAAAABWvi7_B8kKQD9wdXac8MoZiQldmjEO643d-e_j-
↪XXq9AmIegIbA7UHGPv |
|            | atnN21qtOMjCFWX7BReJEQnVOAj3nclRQgAYRsfSU_
↪MrsuWb4EDtnjU7HEpoBb4 |
|            | o6ozsA_NmFWEpLeKy0uNn_WeKbAhYygrsmQGA49dclHVnz-OMVLiyM9ws ␣
↪     |
| project_id | 343d245e850143a096806dfaefa9afdc                          ␣
↪     |
| user_id    | ac33776331494012961f6c0d92d79dc16                         ␣
↪     |
+------------+-----------------------------------------------------------
↪-----+
```

Note

This command uses the password for the `admin` user.

3. As the `myuser` user created in the previous section, request an authentication token:

```
$ openstack --os-auth-url http://controller:5000/v3 \
  --os-project-domain-name Default --os-user-domain-name Default \
  --os-project-name myproject --os-username myuser token issue

Password:
+------------+-----------------------------------------------------------------
↪-----+
| Field      | Value                                                           ␣
↪     |
+------------+-----------------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:15:39.014479Z                                     ␣
↪     |
| id         | gAAAAABWvi9bsh7vkiby5BpCCnc-JkbGhm9wH3fabS_cY7uabOubesi-
↪Me6IGWW |
|            | yQqNegDDZ5jw7grI26vvgy1J5nCVwZ_zFRqPiz_
↪qhbq29mgbQLglbkq6FQvzBRQ |
|            | JcOzq3uwhzNxszJWmzGC7rJE_H0A_a3UFhqv8M4zMRYSbS2YF0MyFmp_U ␣
↪     |
| project_id | ed0b60bf607743088218b0a533d5943f                                ␣
↪     |
| user_id    | 58126687cbcc4888bfa9ab73a2256f27                                ␣
↪     |
+------------+-----------------------------------------------------------------
↪-----+
```

Create OpenStack client environment scripts

The previous sections used a combination of environment variables and command options to interact with the Identity service via the `openstack` client. To increase efficiency of client operations, OpenStack supports simple client environment scripts also known as OpenRC files. These scripts typically contain common options for all clients, but also support unique options. For more information, see the OpenStack End User Guide.

Creating the scripts

Create client environment scripts for the `admin` and `demo` projects and users. Future portions of this guide reference these scripts to load appropriate credentials for client operations.

Note

The paths of the client environment scripts are unrestricted. For convenience, you can place the scripts in any location, however ensure that they are accessible and located in a secure place appropriate for your deployment, as they do contain sensitive credentials.

1. Create and edit the `admin-openrc` file and add the following content:

   Note

   The OpenStack client also supports using a `clouds.yaml` file. For more information, see the os-client-config.

   ```
   export OS_PROJECT_DOMAIN_NAME=Default
   export OS_USER_DOMAIN_NAME=Default
   export OS_PROJECT_NAME=admin
   export OS_USERNAME=admin
   export OS_PASSWORD=ADMIN_PASS
   export OS_AUTH_URL=http://controller:5000/v3
   export OS_IDENTITY_API_VERSION=3
   export OS_IMAGE_API_VERSION=2
   ```

   Replace `ADMIN_PASS` with the password you chose for the `admin` user in the Identity service.

2. Create and edit the `demo-openrc` file and add the following content:

   ```
   export OS_PROJECT_DOMAIN_NAME=Default
   export OS_USER_DOMAIN_NAME=Default
   export OS_PROJECT_NAME=myproject
   export OS_USERNAME=myuser
   export OS_PASSWORD=DEMO_PASS
   export OS_AUTH_URL=http://controller:5000/v3
   export OS_IDENTITY_API_VERSION=3
   export OS_IMAGE_API_VERSION=2
   ```

   Replace `DEMO_PASS` with the password you chose for the `demo` user in the Identity service.

Using the scripts

To run clients as a specific project and user, you can simply load the associated client environment script prior to running them. For example:

1. Load the `admin-openrc` file to populate environment variables with the location of the Identity service and the `admin` project and user credentials:

```
$ . admin-openrc
```

2. Request an authentication token:

```
$ openstack token issue

+------------+----------------------------------------------------------
↪-----+
| Field      | Value                                                    ␣
↪     |
+------------+----------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:44:35.659723Z                              ␣
↪     |
| id         | gAAAAABWvjYj-
↪Zjfg8WXFaQnUd1DMYTBVrKw4h3fIagi5NoEmh21U72SrRv2trl |
|            | JWFYhLi2_uPR31Igf6A8mH2Rw9kv_bxNo1jbLNPLGzW_
↪u5FC7InFqx0yYtTwa1e |
|            | eq2b0f6-18KZyQhs7F3teAta143kJEWuNEYET-y7u29y0be1_64KYkM7E ␣
↪     |
| project_id | 343d245e850143a096806dfaefa9afdc                         ␣
↪     |
| user_id    | ac33776331494012961f6c0d92d79dc16                        ␣
↪     |
+------------+----------------------------------------------------------
↪-----+
```

1.3 Keystone Installation Tutorial for Ubuntu

1.3.1 Abstract

This guide will walk through an installation by using packages available through Canonicals Ubuntu Cloud archive repository for Ubuntu 16.04 (LTS).

Explanations of configuration options and sample configuration files are included.

Warning

This guide is a work-in-progress and is subject to updates frequently. Pre-release packages have been used for testing, and some instructions may not work with final versions. Please help us make this guide better by reporting any errors you encounter.

1.3.2 Contents

Identity service overview

The OpenStack Identity service provides a single point of integration for managing authentication, authorization, and a catalog of services.

The Identity service is typically the first service a user interacts with. Once authenticated, an end user can use their identity to access other OpenStack services. Likewise, other OpenStack services leverage the Identity service to ensure users are who they say they are and discover where other services are within the deployment. The Identity service can also integrate with some external user management systems (such as LDAP).

Users and services can locate other services by using the service catalog, which is managed by the Identity service. As the name implies, a service catalog is a collection of available services in an OpenStack deployment. Each service can have one or many endpoints and each endpoint can be one of three types: admin, internal, or public. In a production environment, different endpoint types might reside on separate networks exposed to different types of users for security reasons. For instance, the public API network might be visible from the Internet so customers can manage their clouds. The admin API network might be restricted to operators within the organization that manages cloud infrastructure. The internal API network might be restricted to the hosts that contain OpenStack services. Also, OpenStack supports multiple regions for scalability. For simplicity, this guide uses the management network for all endpoint types and the default `RegionOne` region. Together, regions, services, and endpoints created within the Identity service comprise the service catalog for a deployment. Each OpenStack service in your deployment needs a service entry with corresponding endpoints stored in the Identity service. This can all be done after the Identity service has been installed and configured.

The Identity service contains these components:

Server
: A centralized server provides authentication and authorization services using a RESTful interface.

Drivers
: Drivers or a service back end are integrated to the centralized server. They are used for accessing identity information in repositories external to OpenStack, and may already exist in the infrastructure where OpenStack is deployed (for example, SQL databases or LDAP servers).

Modules
: Middleware modules run in the address space of the OpenStack component that is using the Identity service. These modules intercept service requests, extract user credentials, and send them to the centralized server for authorization. The integration between the middleware modules and OpenStack components uses the Python Web Server Gateway Interface.

Install and configure

This section describes how to install and configure the OpenStack Identity service, code-named keystone, on the controller node. For scalability purposes, this configuration deploys Fernet tokens and the Apache HTTP server to handle requests.

> **Note**
>
> Ensure that you have completed the prerequisite installation steps in the Openstack Install Guide before proceeding.

Prerequisites

Before you install and configure the Identity service, you must create a database.

1. Use the database access client to connect to the database server as the `root` user:

```
# mysql
```

2. Create the `keystone` database:

```
MariaDB [(none)]> CREATE DATABASE keystone;
```

3. Grant proper access to the `keystone` database:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@
↪'localhost' \
IDENTIFIED BY 'KEYSTONE_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
IDENTIFIED BY 'KEYSTONE_DBPASS';
```

   Replace `KEYSTONE_DBPASS` with a suitable password.

4. Exit the database access client.

Install and configure components

Note

Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

Note

This guide uses the Apache HTTP server with `mod_wsgi` to serve Identity service requests on port 5000. By default, the keystone service still listens on this port. The package handles all of the Apache configuration for you (including the activation of the `mod_wsgi` apache2 module and keystone configuration in Apache).

1. Run the following command to install the packages:

```
# apt install keystone
```

2. Edit the `/etc/keystone/keystone.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

```
[database]
# ...
```

(continues on next page)

(continued from previous page)

```
connection = mysql+pymysql://keystone:KEYSTONE_DBPASS@controller/
↪keystone
```

Replace `KEYSTONE_DBPASS` with the password you chose for the database.

Note

Comment out or remove any other `connection` options in the `[database]` section.

- In the `[token]` section, configure the Fernet token provider:

```
[token]
# ...
provider = fernet
```

3. Populate the Identity service database:

```
# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

4. Initialize Fernet key repositories:

Note

The `--keystone-user` and `--keystone-group` flags are used to specify the operating systems user/group that will be used to run keystone. These are provided to allow running keystone under another operating system user/group. In the example below, we call the user & group `keystone`.

```
# keystone-manage fernet_setup --keystone-user keystone --keystone-group␣
↪keystone
# keystone-manage credential_setup --keystone-user keystone --keystone-
↪group keystone
```

5. Bootstrap the Identity service:

Note

Before the Queens release, keystone needed to be run on two separate ports to accommodate the Identity v2 API which ran a separate admin-only service commonly on port 35357. With the removal of the v2 API, keystone can be run on the same port for all interfaces.

```
# keystone-manage bootstrap --bootstrap-password ADMIN_PASS \
  --bootstrap-admin-url http://controller:5000/v3/ \
  --bootstrap-internal-url http://controller:5000/v3/ \
  --bootstrap-public-url http://controller:5000/v3/ \
  --bootstrap-region-id RegionOne
```

Replace `ADMIN_PASS` with a suitable password for an administrative user.

Configure the Apache HTTP server

1. Edit the `/etc/apache2/apache2.conf` file and configure the `ServerName` option to reference the controller node:

```
ServerName controller
```

   The `ServerName` entry will need to be added if it does not already exist.

SSL

A secure deployment should have the web server configured to use SSL or running behind an SSL terminator.

Finalize the installation

1. Restart the Apache service:

```
# service apache2 restart
```

2. Configure the administrative account by setting the proper environmental variables:

```
$ export OS_USERNAME=admin
$ export OS_PASSWORD=ADMIN_PASS
$ export OS_PROJECT_NAME=admin
$ export OS_USER_DOMAIN_NAME=Default
$ export OS_PROJECT_DOMAIN_NAME=Default
$ export OS_AUTH_URL=http://controller:5000/v3
$ export OS_IDENTITY_API_VERSION=3
```

   These values shown here are the default ones created from `keystone-manage bootstrap`.

   Replace `ADMIN_PASS` with the password used in the `keystone-manage bootstrap` command in *keystone-install-configure-ubuntu*.

Create a domain, projects, users, and roles

The Identity service provides authentication services for each OpenStack service. The authentication service uses a combination of domains, projects, users, and roles.

1. Although the default domain already exists from the *keystone-manage bootstrap* step in this guide, a formal way to create a new domain would be:

```
$ openstack domain create --description "An Example Domain" example

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | An Example Domain                |
| enabled     | True                             |
```

(continues on next page)

(continued from previous page)

```
| id          | 2f4f80574fd84fe6ba9067228ae0a50c |
| name        | example                          |
| tags        | []                               |
+-------------+----------------------------------+
```

2. This guide uses a service project that contains a unique user for each service that you add to your environment. Create the `service` project:

```
$ openstack project create --domain default \
  --description "Service Project" service

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Service Project                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 24ac7f19cd944f4cba1d77469b2a73ed |
| is_domain   | False                            |
| name        | service                          |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

3. Regular (non-admin) tasks should use an unprivileged project and user. As an example, this guide creates the `myproject` project and `myuser` user.

   - Create the `myproject` project:

```
$ openstack project create --domain default \
  --description "Demo Project" myproject

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Demo Project                     |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 231ad6e7ebba47d6a1e57e1cc07ae446 |
| is_domain   | False                            |
| name        | myproject                        |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

   Note

   Do not repeat this step when creating additional users for this project.

   - Create the `myuser` user:

```
$ openstack user create --domain default \
  --password-prompt myuser

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | aeda23aa78f44e859900e22c24817832 |
| name                | myuser                           |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

- Create the `myrole` role:

```
$ openstack role create myrole

+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 997ce8d05fc143ac97d83fdfb5998552 |
| name      | myrole                           |
+-----------+----------------------------------+
```

- Add the `myrole` role to the `myproject` project and `myuser` user:

```
$ openstack role add --project myproject --user myuser myrole
```

Note

This command provides no output.

Note

You can repeat this procedure to create additional projects and users.

Verify operation

Verify operation of the Identity service before installing other services.

Note

Perform these commands on the controller node.

1. Unset the temporary `OS_AUTH_URL` and `OS_PASSWORD` environment variable:

```
$ unset OS_AUTH_URL OS_PASSWORD
```

2. As the `admin` user, request an authentication token:

```
$ openstack --os-auth-url http://controller:5000/v3 \
  --os-project-domain-name Default --os-user-domain-name Default \
  --os-project-name admin --os-username admin token issue

Password:
+------------+-----------------------------------------------------------
↪-----+
| Field      | Value                                                     ␣
↪     |
+------------+-----------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:14:07.056119Z                               ␣
↪     |
| id         | gAAAAABWvi7_B8kKQD9wdXac8MoZiQldmjEO643d-e_j-
↪XXq9AmIegIbA7UHGPv |
|            | atnN21qtOMjCFWX7BReJEQnVOAj3nclRQgAYRsfSU_
↪MrsuWb4EDtnjU7HEpoBb4 |
|            | o6ozsA_NmFWEpLeKy0uNn_WeKbAhYygrsmQGA49dclHVnz-OMVLiyM9ws ␣
↪     |
| project_id | 343d245e850143a096806dfaefa9afdc                          ␣
↪     |
| user_id    | ac3377633149401296f6c0d92d79dc16                          ␣
↪     |
+------------+-----------------------------------------------------------
↪-----+
```

Note

This command uses the password for the `admin` user.

3. As the `myuser` user created in the previous, request an authentication token:

```
$ openstack --os-auth-url http://controller:5000/v3 \
  --os-project-domain-name Default --os-user-domain-name Default \
  --os-project-name myproject --os-username myuser token issue
```

(continues on next page)

(continued from previous page)

```
Password:
+------------+----------------------------------------------------------
↪-----+
| Field      | Value                                                    ␣
↪     |
+------------+----------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:15:39.014479Z                              ␣
↪     |
| id         | gAAAAABWvi9bsh7vkiby5BpCCnc-JkbGhm9wH3fabS_cY7uabOubesi-
↪Me6IGWW |
|            | yQqNegDDZ5jw7grI26vvgy1J5nCVwZ_zFRqPiz_
↪qhbq29mgbQLglbkq6FQvzBRQ |
|            | JcOzq3uwhzNxszJWmzGC7rJE_H0A_a3UFhqv8M4zMRYSbS2YF0MyFmp_U ␣
↪     |
| project_id | ed0b60bf607743088218b0a533d5943f                         ␣
↪     |
| user_id    | 58126687cbcc4888bfa9ab73a2256f27                         ␣
↪     |
+------------+----------------------------------------------------------
↪-----+
```

Create OpenStack client environment scripts

The previous sections used a combination of environment variables and command options to interact with the Identity service via the `openstack` client. To increase efficiency of client operations, OpenStack supports simple client environment scripts also known as OpenRC files. These scripts typically contain common options for all clients, but also support unique options. For more information, see the OpenStack End User Guide.

Creating the scripts

Create client environment scripts for the `admin` and `demo` projects and users. Future portions of this guide reference these scripts to load appropriate credentials for client operations.

Note

The paths of the client environment scripts are unrestricted. For convenience, you can place the scripts in any location, however ensure that they are accessible and located in a secure place appropriate for your deployment, as they do contain sensitive credentials.

1. Create and edit the `admin-openrc` file and add the following content:

   Note

   The OpenStack client also supports using a `clouds.yaml` file. For more information, see the os-client-config.

```
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
```

Replace `ADMIN_PASS` with the password you chose for the `admin` user in the Identity service.

2. Create and edit the `demo-openrc` file and add the following content:

```
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=myproject
export OS_USERNAME=myuser
export OS_PASSWORD=DEMO_PASS
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
```

Replace `DEMO_PASS` with the password you chose for the `demo` user in the Identity service.

Using the scripts

To run clients as a specific project and user, you can simply load the associated client environment script prior to running them. For example:

1. Load the `admin-openrc` file to populate environment variables with the location of the Identity service and the `admin` project and user credentials:

```
$ . admin-openrc
```

2. Request an authentication token:

```
$ openstack token issue

+------------+----------------------------------------------------------
↪-----+
| Field      | Value                                                    ⌋
↪     |
+------------+----------------------------------------------------------
↪-----+
| expires    | 2016-02-12T20:44:35.659723Z                              ⌋
↪     |
| id         | gAAAAABWvjYj-
↪Zjfg8WXFaQnUd1DMYTBVrKw4h3fIagi5NoEmh21U72SrRv2trl |
|            | JWFYhLi2_uPR31Igf6A8mH2Rw9kv_bxNo1jbLNPLGzW_
↪u5FC7InFqx0yYtTwa1e |
|            | eq2b0f6-18KZyQhs7F3teAta143kJEWuNEYET-y7u29y0be1_64KYkM7E ⌋
```

(continues on next page)

(continued from previous page)

```
↪      |
| project_id | 343d245e850143a096806dfaefa9afdc                                 ␣
↪      |
| user_id    | ac33776331494012 96f6c0d92d79dc16                                ␣
↪      |
+------------+------------------------------------------------------------------
↪-----+
```

CHAPTER

TWO

GETTING STARTED

2.1 Keystone Architecture

Much of the design assumes that in most deployments auth backends will be shims in front of existing user systems.

2.1.1 Services

Keystone is organized as a group of internal services exposed on one or many endpoints. Many of these services are used in a combined fashion by the frontend. For example, an authenticate call will validate user/project credentials with the Identity service and, upon success, create and return a token with the Token service.

Identity

The Identity service provides auth credential validation and data about *users* and *groups*. In the basic case, this data is managed by the Identity service, allowing it to also handle all CRUD operations associated with this data. In more complex cases, the data is instead managed by an authoritative backend service. An example of this would be when the Identity service acts as a frontend for LDAP. In that case the LDAP server is the source of truth and the role of the Identity service is to relay that information accurately.

Users

`Users` represent an individual API consumer. A user itself must be owned by a specific domain, and hence all user names are **not** globally unique, but only unique to their domain.

Groups

`Groups` are a container representing a collection of users. A group itself must be owned by a specific domain, and hence all group names are **not** globally unique, but only unique to their domain.

Resource

The Resource service provides data about *projects* and *domains*.

Projects

`Projects` represent the base unit of `ownership` in OpenStack, in that all resources in OpenStack should be owned by a specific project. A project itself must be owned by a specific domain, and hence all project names are **not** globally unique, but unique to their domain. If the domain for a project is not specified, then it is added to the default domain.

Domains

`Domains` are a high-level container for projects, users and groups. Each is owned by exactly one domain. Each domain defines a namespace where an API-visible name attribute exists. Keystone provides a default domain, aptly named Default.

In the Identity v3 API, the uniqueness of attributes is as follows:

- Domain Name. Globally unique across all domains.
- Role Name. Unique within the owning domain.
- User Name. Unique within the owning domain.
- Project Name. Unique within the owning domain.
- Group Name. Unique within the owning domain.

Due to their container architecture, domains may be used as a way to delegate management of OpenStack resources. A user in a domain may still access resources in another domain, if an appropriate assignment is granted.

Assignment

The Assignment service provides data about *roles* and *role assignments*.

Roles

`Roles` dictate the level of authorization the end user can obtain. Roles can be granted at either the domain or project level. A role can be assigned at the individual user or group level. Role names are unique within the owning domain.

Role Assignments

A 3-tuple that has a `Role`, a `Resource` and an `Identity`.

Token

The Token service validates and manages tokens used for authenticating requests once a users credentials have already been verified.

Catalog

The Catalog service provides an endpoint registry used for endpoint discovery.

2.1.2 Application Construction

Keystone is an HTTP front-end to several services. Since the Rocky release Keystone uses the Flask-RESTful library to provide a REST API interface to these services.

Keystone defines functions related to Flask-RESTful in *keystone.server.flask.common*. Keystone creates API resources which inherit from class *keystone.server.flask.common.ResourceBase* and exposes methods for each supported HTTP methods GET, PUT , POST, PATCH and DELETE. For example, the User resource will look like:

```
class UserResource(ks_flask.ResourceBase):
    collection_key = 'users'
    member_key = 'user'
    get_member_from_driver = PROVIDERS.deferred_provider_lookup(
        api='identity_api', method='get_user')

    def get(self, user_id=None):
        """Get a user resource or list users.
        GET/HEAD /v3/users
        GET/HEAD /v3/users/{user_id}
        """
        ...

    def post(self):
        """Create a user.
        POST /v3/users
        """
        ...

class UserChangePasswordResource(ks_flask.ResourceBase):
    @ks_flask.unenforced_api
     def post(self, user_id):
         ...
```

Routes for each API resource are defined by classes which inherit from *keystone.server.flask.common.APIBase*. For example, the UserAPI will look like:

```
class UserAPI(ks_flask.APIBase):
    _name = 'users'
    _import_name = __name__
    resources = [UserResource]
    resource_mapping = [
        ks_flask.construct_resource_map(
            resource=UserChangePasswordResource,
            url='/users/<string:user_id>/password',
            resource_kwargs={},
            rel='user_change_password',
            path_vars={'user_id': json_home.Parameters.USER_ID}
        ),
     ...
```

The methods `_add_resources()` or `_add_mapped_resources()` in *keystone.server.flask.common.APIBase* bind the resources with the APIs. Within each API, one or more managers are loaded (for example, see *keystone.catalog.core.Manager*), which are thin wrapper classes which load the appropriate service driver based on the keystone configuration.

- Assignment
  - *keystone.api.role_assignments*
  - *keystone.api.role_inferences*
  - *keystone.api.roles*
  - *keystone.api.os_inherit*
  - *keystone.api.system*
- Authentication
  - *keystone.api.auth*
  - *keystone.api.ec2tokens*
  - *keystone.api.s3tokens*
- Catalog
  - *keystone.api.endpoints*
  - *keystone.api.os_ep_filter*
  - *keystone.api.regions*
  - *keystone.api.services*
- Credentials
  - *keystone.api.credentials*
- Federation
  - *keystone.api.os_federation*
- Identity
  - *keystone.api.groups*
  - *keystone.api.users*

- Limits
  - *keystone.api.registered_limits*
  - *keystone.api.limits*
- Oauth1
  - *keystone.api.os_oauth1*
- Policy
  - *keystone.api.policy*
- Resource
  - *keystone.api.domains*
  - *keystone.api.projects*
- Revoke
  - *keystone.api.os_revoke*
- Trust
  - *keystone.api.trusts*

2.1.3 Service Backends

Each of the services can be configured to use a backend to allow keystone to fit a variety of environments and needs. The backend for each service is defined in the keystone.conf file with the key `driver` under a group associated with each service.

A general class exists under each backend to provide an abstract base class for any implementations, identifying the expected service implementations. The abstract base classes are stored in the services backends directory as `base.py`. The corresponding drivers for the services are:

- *keystone.assignment.backends.base.AssignmentDriverBase*
- *keystone.assignment.role_backends.base.RoleDriverBase*
- *keystone.auth.plugins.base.AuthMethodHandler*
- *keystone.catalog.backends.base.CatalogDriverBase*
- *keystone.credential.backends.base.CredentialDriverBase*
- *keystone.endpoint_policy.backends.base.EndpointPolicyDriverBase*
- *keystone.federation.backends.base.FederationDriverBase*
- *keystone.identity.backends.base.IdentityDriverBase*
- *keystone.identity.mapping_backends.base.MappingDriverBase*
- *keystone.identity.shadow_backends.base.ShadowUsersDriverBase*
- *keystone.oauth1.backends.base.Oauth1DriverBase*
- *keystone.policy.backends.base.PolicyDriverBase*
- *keystone.resource.backends.base.ResourceDriverBase*
- *keystone.resource.config_backends.base.DomainConfigDriverBase*

- *keystone.revoke.backends.base.RevokeDriverBase*
- *keystone.token.providers.base.Provider*
- *keystone.trust.backends.base.TrustDriverBase*

If you implement a backend driver for one of the keystone services, youre expected to subclass from these classes.

Templated Backend

Largely designed for a common use case around service catalogs in the keystone project, a templated backend is a catalog backend that simply expands pre-configured templates to provide catalog data.

Example paste.deploy config (uses $ instead of % to avoid ConfigParsers interpolation)

```
[DEFAULT]
catalog.RegionOne.identity.publicURL = http://localhost:$(public_port)s/v3
catalog.RegionOne.identity.adminURL = http://localhost:$(public_port)s/v3
catalog.RegionOne.identity.internalURL = http://localhost:$(public_port)s/v3
catalog.RegionOne.identity.name = 'Identity Service'
```

2.1.4 Data Model

Keystone was designed from the ground up to be amenable to multiple styles of backends. As such, many of the methods and data types will happily accept more data than they know what to do with and pass them on to a backend.

There are a few main data types:

- **User**: has account credentials, is associated with one or more projects or domains
- **Group**: a collection of users, is associated with one or more projects or domains
- **Project**: unit of ownership in OpenStack, contains one or more users
- **Domain**: unit of ownership in OpenStack, contains users, groups and projects
- **Role**: a first-class piece of metadata associated with many user-project pairs.
- **Token**: identifying credential associated with a user or user and project
- **Extras**: bucket of key-value metadata associated with a user-project pair.
- **Rule**: describes a set of requirements for performing an action.

While the general data model allows a many-to-many relationship between users and groups to projects and domains; the actual backend implementations take varying levels of advantage of that functionality.

2.1.5 Approach to CRUD

While it is expected that any real deployment at a large company will manage their users and groups in their existing user systems, a variety of CRUD operations are provided for the sake of development and testing.

CRUD is treated as an extension or additional feature to the core feature set, in that a backend is not required to support it. It is expected that backends for services that dont support the CRUD operations will raise a *keystone.exception.NotImplemented*.

2.1.6 Approach to Authorization (Policy)

Various components in the system require that different actions are allowed based on whether the user is authorized to perform that action.

For the purposes of keystone there are only a couple levels of authorization being checked for:

- Require that the performing user is considered an admin.
- Require that the performing user matches the user being referenced.

Other systems wishing to use the policy engine will require additional styles of checks and will possibly write completely custom backends. By default, keystone leverages policy enforcement that is maintained in oslo.policy.

Rules

Given a list of matches to check for, simply verify that the credentials contain the matches. For example:

```
credentials = {'user_id': 'foo', 'is_admin': 1, 'roles': ['nova:netadmin']}

# An admin only call:
policy_api.enforce(('is_admin:1',), credentials)

# An admin or owner call:
policy_api.enforce(('is_admin:1', 'user_id:foo'), credentials)

# A netadmin call:
policy_api.enforce(('roles:nova:netadmin',), credentials)
```

Credentials are generally built from the user metadata in the extras part of the Identity API. So, adding a role to the user just means adding the role to the user metadata.

Capability RBAC

(Not yet implemented.)

Another approach to authorization can be action-based, with a mapping of roles to which capabilities are allowed for that role. For example:

```
credentials = {'user_id': 'foo', 'is_admin': 1, 'roles': ['nova:netadmin']}

# add a policy
policy_api.add_policy('action:nova:add_network', ('roles:nova:netadmin',))

policy_api.enforce(('action:nova:add_network',), credentials)
```

In the backend this would look up the policy for action:nova:add_network and then do what is effectively a Simple Match style match against the credentials.

2.1.7 Approach to Authentication

Keystone provides several authentication plugins that inherit from *keystone.auth.plugins.base*. The following is a list of available plugins.

- *keystone.auth.plugins.external.Base*
- *keystone.auth.plugins.mapped.Mapped*
- *keystone.auth.plugins.oauth1.OAuth*
- *keystone.auth.plugins.password.Password*
- *keystone.auth.plugins.token.Token*
- *keystone.auth.plugins.totp.TOTP*

In the most basic plugin `password`, two pieces of information are required to authenticate with keystone, a bit of `Resource` information and a bit of `Identity`.

Take the following call POST data for instance:

```
{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                    "id": "0ca8f6",
                    "password": "secretsecret"
                }
            }
        },
        "scope": {
            "project": {
                "id": "263fd9"
```

(continues on next page)

(continued from previous page)

```
            }
        }
    }
}
```

The user (ID of 0ca8f6) is attempting to retrieve a token that is scoped to project (ID of 263fd9).

To perform the same call with names instead of IDs, we now need to supply information about the domain. This is because usernames are only unique within a given domain, but user IDs are supposed to be unique across the deployment. Thus, the auth request looks like the following:

```
{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                    "domain": {
                        "name": "acme"
                    }
                    "name": "userA",
                    "password": "secretsecret"
                }
            }
        },
        "scope": {
            "project": {
                "domain": {
                    "id": "1789d1"
                },
                "name": "project-x"
            }
        }
    }
}
```

For both the user and the project portion, we must supply either a domain ID or a domain name, in order to properly determine the correct user and project.

Alternatively, if we wanted to represent this as environment variables for a command line, it would be:

```
$ export OS_PROJECT_DOMAIN_ID=1789d1
$ export OS_USER_DOMAIN_NAME=acme
$ export OS_USERNAME=userA
$ export OS_PASSWORD=secretsecret
$ export OS_PROJECT_NAME=project-x
```

Note that the project the user is attempting to access must be in the same domain as the user.

What is Scope?

Scope is an overloaded term.

In reference to authenticating, as seen above, scope refers to the portion of the POST data that dictates what `Resource` (project, domain, or system) the user wants to access.

In reference to tokens, scope refers to the effectiveness of a token, i.e.: a *project-scoped* token is only useful on the project it was initially granted for. A *domain-scoped* token may be used to perform domain-related function. A *system-scoped* token is only useful for interacting with APIs that affect the entire deployment.

In reference to users, groups, and projects, scope often refers to the domain that the entity is owned by. i.e.: a user in domain X is scoped to domain X.

2.2 Mapping of policy target to API

The following table shows the target in the policy.yaml file for each API.

Target	API
identity:get_region	GET /v3/regions/{region_id}
identity:list_regions	GET /v3/regions
identity:create_region	POST /v3/regions
identity:update_region	PATCH /v3/regions/{region_id}
identity:delete_region	DELETE /v3/regions/{region_id}
identity:get_service	GET /v3/services/{service_id}
identity:list_services	GET /v3/services
identity:create_service	POST /v3/services
identity:update_service	PATCH /v3/services/{service__id}
identity:delete_service	DELETE /v3/services/{service__id}
identity:get_endpoint	GET /v3/endpoints/{endpoint_id}
identity:list_endpoints	GET /v3/endpoints
identity:create_endpoint	POST /v3/endpoints
identity:update_endpoint	PATCH /v3/endpoints/{endpoint_id}
identity:delete_endpoint	DELETE /v3/endpoints/{endpoint_id}
identity:get_registered_limit	GET /v3/registered_limits/{registered_limit_id}
identity:list_registered_limits	GET /v3/registered_limits
identity:create_registered_limits	POST /v3/registered_limits
identity:update_registered_limit	PATCH /v3/registered_limits/{registered_limit_id}
identity:delete_registered_limit	DELETE /v3/registered_limits/{registered_limit_id}
identity:get_limit	GET /v3/limits/{limit_id}
identity:list_limits	GET /v3/limits
identity:create_limits	POST /v3/limits
identity:update_limit	PATCH /v3/limits/{limit_id}
identity:delete_limit	DELETE /v3/limits/{limit_id}
identity:get_limit_model	GET /v3/limits/model HEAD /v3/limits/model
identity:get_domain	GET /v3/domains/{domain_id}
identity:list_domains	GET /v3/domains
identity:create_domain	POST /v3/domains
identity:update_domain	PATCH /v3/domains/{domain_id}

Table 1 – continued from pr

Target	API
identity:delete_domain	DELETE /v3/domains/{domain_id}
identity:get_project	GET /v3/projects/{project_id}
identity:list_projects	GET /v3/projects
identity:list_user_projects	GET /v3/users/{user_id}/projects
identity:create_project	POST /v3/projects
identity:update_project	PATCH /v3/projects/{project_id}
identity:delete_project	DELETE /v3/projects/{project_id}
identity:get_project_tag	GET /v3/projects/{project_id}/tags/{tag_name} HEAD /
identity:list_project_tags	GET /v3/projects/{project_id}/tags HEAD /v3/projects/{
identity:create_project_tag	PUT /v3/projects/{project_id}/tags/{tag_name}
identity:update_project_tags	PUT /v3/projects/{project_id}/tags
identity:delete_project_tag	DELETE /v3/projects/{project_id}/tags/{tag_name}
identity:delete_project_tags	DELETE /v3/projects/{project_id}/tags
identity:get_user	GET /v3/users/{user_id}
identity:list_users	GET /v3/users
identity:create_user	POST /v3/users
identity:update_user	PATCH /v3/users/{user_id}
identity:delete_user	DELETE /v3/users/{user_id}
identity:get_group	GET /v3/groups/{group_id}
identity:list_groups	GET /v3/groups
identity:list_groups_for_user	GET /v3/users/{user_id}/groups
identity:create_group	POST /v3/groups
identity:update_group	PATCH /v3/groups/{group_id}
identity:delete_group	DELETE /v3/groups/{group_id}
identity:list_users_in_group	GET /v3/groups/{group_id}/users
identity:remove_user_from_group	DELETE /v3/groups/{group_id}/users/{user_id}
identity:check_user_in_group	GET /v3/groups/{group_id}/users/{user_id}
identity:add_user_to_group	PUT /v3/groups/{group_id}/users/{user_id}
identity:get_credential	GET /v3/credentials/{credential_id}
identity:list_credentials	GET /v3/credentials
identity:create_credential	POST /v3/credentials
identity:update_credential	PATCH /v3/credentials/{credential_id}
identity:delete_credential	DELETE /v3/credentials/{credential_id}
identity:ec2_get_credential	GET /v3/users/{user_id}/credentials/OS-EC2/{credentia
identity:ec2_list_credentials	GET /v3/users/{user_id}/credentials/OS-EC2
identity:ec2_create_credential	POST /v3/users/{user_id}/credentials/OS-EC2
identity:ec2_delete_credential	DELETE /v3/users/{user_id}/credentials/OS-EC2/{crede
identity:get_role	GET /v3/roles/{role_id}
identity:list_roles	GET /v3/roles
identity:create_role	POST /v3/roles
identity:update_role	PATCH /v3/roles/{role_id}
identity:delete_role	DELETE /v3/roles/{role_id}
identity:get_domain_role	GET /v3/roles/{role_id} where role.domain_id is not nul
identity:list_domain_roles	GET /v3/roles?domain_id where role.domain_id is not n
identity:create_domain_role	POST /v3/roles where role.domain_id is not null
identity:update_domain_role	PATCH /v3/roles/{role_id} where role.domain_id is not
identity:delete_domain_role	DELETE /v3/roles/{role_id} where role.domain_id is no
identity:get_implied_role	GET /v3/roles/{prior_role_id}/implies/{implied_role_id

Table 1 – continued from pr

Target	API
identity:list_implied_roles	GET /v3/roles/{prior_role_id}/implies
identity:create_implied_role	PUT /v3/roles/{prior_role_id}/implies/{implied_role_id
identity:delete_implied_role	DELETE /v3/roles/{prior_role_id}/implies/{implied_rol
identity:list_role_inference_rules	GET /v3/role_inferences
identity:check_implied_role	HEAD /v3/roles/{prior_role_id}/implies/{implied_role_
identity:check_grant	GET *grant_resources*
identity:list_grants	GET *grant_collections*
identity:create_grant	PUT *grant_resources*
identity:revoke_grant	DELETE *grant_resources*
identity:list_system_grants_for_user	GET /v3/system/users/{user_id}/roles
identity:check_system_grant_for_user	GET /v3/system/users/{user_id}/roles/{role_id}
identity:create_system_grant_for_user	PUT /v3/system/users/{user_id}/roles/{role_id}
identity:revoke_system_grant_for_user	DELETE /v3/system/users/{user_id}/roles/{role_id}
identity:list_system_grants_for_group	GET /v3/system/groups/{group_id}/roles
identity:check_system_grant_for_group	GET /v3/system/groups/{group_id}/roles/{role_id}
identity:create_system_grant_for_group	PUT /v3/system/groups/{group_id}/roles/{role_id}
identity:revoke_system_grant_for_group	DELETE /v3/system/groups/{group_id}/roles/{role_id}
identity:list_role_assignments	GET /v3/role_assignments
identity:list_role_assignments_for_tree	GET /v3/role_assignments?include_subtree
identity:get_policy	GET /v3/policy/{policy_id}
identity:list_policies	GET /v3/policy
identity:create_policy	POST /v3/policy
identity:update_policy	PATCH /v3/policy/{policy_id}
identity:delete_policy	DELETE /v3/policy/{policy_id}
identity:check_token	HEAD /v3/auth/tokens
identity:validate_token	GET /v3/auth/tokens
identity:revocation_list	GET /v3/auth/tokens/OS-PKI/revoked
identity:revoke_token	DELETE /v3/auth/tokens
identity:create_trust	POST /v3/OS-TRUST/trusts
identity:list_trusts	GET /v3/OS-TRUST/trusts
identity:list_trusts_for_trustor	GET /v3/OS-TRUST/trusts?trustor_user_id={trustor_use
identity:list_trusts_for_trustee	GET /v3/OS-TRUST/trusts?trustee_user_id={trustee_use
identity:list_roles_for_trust	GET /v3/OS-TRUST/trusts/{trust_id}/roles
identity:get_role_for_trust	GET /v3/OS-TRUST/trusts/{trust_id}/roles/{role_id}
identity:delete_trust	DELETE /v3/OS-TRUST/trusts/{trust_id}
identity:get_trust	GET /v3/OS-TRUST/trusts/{trust_id}
identity:create_consumer	POST /v3/OS-OAUTH1/consumers
identity:get_consumer	GET /v3/OS-OAUTH1/consumers/{consumer_id}
identity:list_consumers	GET /v3/OS-OAUTH1/consumers
identity:delete_consumer	DELETE /v3/OS-OAUTH1/consumers/{consumer_id}
identity:update_consumer	PATCH /v3/OS-OAUTH1/consumers/{consumer_id}
identity:authorize_request_token	PUT /v3/OS-OAUTH1/authorize/{request_token_id}
identity:list_access_token_roles	GET /v3/users/{user_id}/OS-OAUTH1/access_tokens/{a
identity:get_access_token_role	GET /v3/users/{user_id}/OS-OAUTH1/access_tokens/{a
identity:list_access_tokens	GET /v3/users/{user_id}/OS-OAUTH1/access_tokens
identity:get_access_token	GET /v3/users/{user_id}/OS-OAUTH1/access_tokens/{a
identity:delete_access_token	DELETE /v3/users/{user_id}/OS-OAUTH1/access_toke
identity:list_projects_for_endpoint	GET /v3/OS-EP-FILTER/endpoints/{endpoint_id}/proje

Table 1 – continued from pr

Target	API
identity:add_endpoint_to_project	PUT /v3/OS-EP-FILTER/projects/{project_id}/endpoints
identity:check_endpoint_in_project	GET /v3/OS-EP-FILTER/projects/{project_id}/endpoint
identity:list_endpoints_for_project	GET /v3/OS-EP-FILTER/projects/{project_id}/endpoint
identity:remove_endpoint_from_project	DELETE /v3/OS-EP-FILTER/projects/{project_id}/endp
identity:create_endpoint_group	POST /v3/OS-EP-FILTER/endpoint_groups
identity:list_endpoint_groups	GET /v3/OS-EP-FILTER/endpoint_groups
identity:get_endpoint_group	GET /v3/OS-EP-FILTER/endpoint_groups/{endpoint_gr
identity:update_endpoint_group	PATCH /v3/OS-EP-FILTER/endpoint_groups/{endpoint_
identity:delete_endpoint_group	DELETE /v3/OS-EP-FILTER/endpoint_groups/{endpoin
identity:list_projects_associated_with_endpoint_group	GET /v3/OS-EP-FILTER/endpoint_groups/{endpoint_gr
identity:list_endpoints_associated_with_endpoint_group	GET /v3/OS-EP-FILTER/endpoint_groups/{endpoint_gr
identity:get_endpoint_group_in_project	GET /v3/OS-EP-FILTER/endpoint_groups/{endpoint_gr
identity:list_endpoint_groups_for_project	GET /v3/OS-EP-FILTER/projects/{project_id}/endpoint_
identity:add_endpoint_group_to_project	PUT /v3/OS-EP-FILTER/endpoint_groups/{endpoint_gr
identity:remove_endpoint_group_from_project	DELETE /v3/OS-EP-FILTER/endpoint_groups/{endpoin
identity:create_identity_provider	PUT /v3/OS-FEDERATION/identity_providers/{idp_id}
identity:list_identity_providers	GET /v3/OS-FEDERATION/identity_providers
identity:get_identity_provider	GET /v3/OS-FEDERATION/identity_providers/{idp_id}
identity:update_identity_provider	PATCH /v3/OS-FEDERATION/identity_providers/{idp_
identity:delete_identity_provider	DELETE /v3/OS-FEDERATION/identity_providers/{idp
identity:create_protocol	PUT /v3/OS-FEDERATION/identity_providers/{idp_id}
identity:update_protocol	PATCH /v3/OS-FEDERATION/identity_providers/{idp_
identity:get_protocol	GET /v3/OS-FEDERATION/identity_providers/{idp_id}
identity:list_protocols	GET /v3/OS-FEDERATION/identity_providers/{idp_id}
identity:delete_protocol	DELETE /v3/OS-FEDERATION/identity_providers/{idp
identity:create_mapping	PUT /v3/OS-FEDERATION/mappings/{mapping_id}
identity:get_mapping	GET /v3/OS-FEDERATION/mappings/{mapping_id}
identity:list_mappings	GET /v3/OS-FEDERATION/mappings
identity:delete_mapping	DELETE /v3/OS-FEDERATION/mappings/{mapping_i
identity:update_mapping	PATCH /v3/OS-FEDERATION/mappings/{mapping_id}
identity:create_service_provider	PUT /v3/OS-FEDERATION/service_providers/{sp_id}
identity:list_service_providers	GET /v3/OS-FEDERATION/service_providers
identity:get_service_provider	GET /v3/OS-FEDERATION/service_providers/{sp_id}
identity:update_service_provider	PATCH /v3/OS-FEDERATION/service_providers/{sp_i
identity:delete_service_provider	DELETE /v3/OS-FEDERATION/service_providers/{sp_
identity:get_auth_catalog	GET /v3/auth/catalog
identity:get_auth_projects	GET /v3/auth/projects
identity:get_auth_domains	GET /v3/auth/domains
identity:get_auth_system	GET /v3/auth/system
identity:list_projects_for_user	GET /v3/OS-FEDERATION/projects
identity:list_domains_for_user	GET /v3/OS-FEDERATION/domains
identity:list_revoke_events	GET /v3/OS-REVOKE/events
identity:create_policy_association_for_endpoint	PUT /v3/policies/{policy_id}/OS-ENDPOINT-POLICY/
identity:check_policy_association_for_endpoint	GET /v3/policies/{policy_id}/OS-ENDPOINT-POLICY/
identity:delete_policy_association_for_endpoint	DELETE /v3/policies/{policy_id}/OS-ENDPOINT-POL
identity:create_policy_association_for_service	PUT /v3/policies/{policy_id}/OS-ENDPOINT-POLICY/
identity:check_policy_association_for_service	GET /v3/policies/{policy_id}/OS-ENDPOINT-POLICY/
identity:delete_policy_association_for_service	DELETE /v3/policies/{policy_id}/OS-ENDPOINT-POL

Table 1 – continued from pr

Target	API
identity:create_policy_association_for_region_and_service	PUT /v3/policies/{policy_id}/OS-ENDPOINT-POLICY/
identity:check_policy_association_for_region_and_service	GET /v3/policies/{policy_id}/OS-ENDPOINT-POLICY/
identity:delete_policy_association_for_region_and_service	DELETE /v3/policies/{policy_id}/OS-ENDPOINT-POL
identity:get_policy_for_endpoint	GET /v3/endpoints/{endpoint_id}/OS-ENDPOINT-POL
identity:list_endpoints_for_policy	GET /v3/policies/{policy_id}/OS-ENDPOINT-POLICY/
identity:create_domain_config	PUT /v3/domains/{domain_id}/config
identity:get_domain_config	GET /v3/domains/{domain_id}/config GET /v3/domains
identity:get_security_compliance_domain_config	GET /v3/domains/{domain_id}/config/security_compliar
identity:update_domain_config	PATCH /v3/domains/{domain_id}/config PATCH /v3/do
identity:delete_domain_config	DELETE /v3/domains/{domain_id}/config DELETE /v3
identity:get_domain_config_default	GET /v3/domains/config/default GET /v3/domains/confi
identity:get_application_credential	GET /v3/users/{user_id}/application_credentials/{applic
identity:list_application_credentials	GET /v3/users/{user_id}/application_credentials
identity:create_application_credential	POST /v3/users/{user_id}/application_credential
identity:delete_application_credential	DELETE /v3/users/{user_id}/application_credential/{ap
identity:get_access_rule	GET /v3/users/{user_id}/access_rules/{access_rule_id}
identity:list_access_rules	GET /v3/users/{user_id}/access_rules
identity:delete_access_rule	DELETE /v3/users/{user_id}/access_rules/{access_rule_
identity:s3tokens_validate	POST /v3/s3tokens
identity:ec2tokens_validate	POST /v3/es2tokens

grant_resources are:

- /v3/projects/{project_id}/users/{user_id}/roles/{role_id}
- /v3/projects/{project_id}/groups/{group_id}/roles/{role_id}
- /v3/domains/{domain_id}/users/{user_id}/roles/{role_id}
- /v3/domains/{domain_id}/groups/{group_id}/roles/{role_id}
- /v3/OS-INHERIT/domains/{domain_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- /v3/OS-INHERIT/domains/{domain_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects
- /v3/OS-INHERIT/projects/{project_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- /v3/OS-INHERIT/projects/{project_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects

grant_collections are:

- /v3/projects/{project_id}/users/{user_id}/roles
- /v3/projects/{project_id}/groups/{group_id}/roles
- /v3/domains/{domain_id}/users/{user_id}/roles
- /v3/domains/{domain_id}/groups/{group_id}/roles
- /v3/OS-INHERIT/domains/{domain_id}/groups/{group_id}/roles/inherited_to_projects
- /v3/OS-INHERIT/domains/{domain_id}/users/{user_id}/roles/inherited_to_projects

2.3 Getting Involved

The OpenStack community is a very friendly group and there are places online to join in with the community. Feel free to ask questions. This document points you to some of the places where you can communicate with people.

2.3.1 How to Join the Community

Our community welcomes all people interested in open source cloud computing, and there are no formal membership requirements. The best way to join the community is to talk with others online or at a meetup and offer contributions through Launchpad, the wiki, or blogs. We welcome all types of contributions, from feature designs to documentation to testing to deployment scripts.

#openstack-keystone on OFTC IRC Network

You can find Keystone folks in irc://oftc.net/#openstack-keystone. This is usually the best place to ask questions and find your way around. IRC stands for Internet Relay Chat and it is a way to chat online in real time. You can also ask a question and come back to the log files to read the answer later. Logs for the #openstack IRC channels are stored at http://eavesdrop.openstack.org/irclogs/.

For more information regarding OpenStack IRC channels please visit the OpenStack IRC Wiki.

Keystone on Launchpad

Launchpad is a code hosting that OpenStack is using to track bugs, feature work, and releases of OpenStack. Like other OpenStack projects, Keystone source code is hosted on opendev.org

- Keystone Project Page on Launchpad
- Keystone Source Repository

Within launchpad, we use bugs to report issues as well as to track feature work. If you are looking for a place to get started contributing to keystone, please look at any bugs for Keystone that are tagged as low-hanging-fruit.

OpenStack Blog

The OpenStack blog includes a weekly newsletter that aggregates OpenStack news from around the internet, as well as providing inside information on upcoming events and posts from OpenStack contributors.

OpenStack Blog

See also: Planet OpenStack, an aggregation of blogs about OpenStack from around the internet, combined into a web site and RSS feed. If youd like to contribute with your blog posts, there are instructions for adding your blog.

Twitter

Because all the cool kids do it: @openstack. Also follow the #openstack tag for relevant tweets.

CHAPTER

THREE

CODE DOCUMENTATION

3.1 keystone

3.1.1 keystone package

Subpackages

keystone.api package

Submodules

keystone.api.auth module

class keystone.api.auth.**AuthAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.auth.AuthProjectsResource'>, '/auth/projects', [{'json_home': ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/projects', 'stable', {}), 'url': '/OS-FEDERATION/projects'}], {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/auth_projects', 'stable', {})), (<class 'keystone.api.auth.AuthDomainsResource'>, '/auth/domains', [{'json_home': ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/domains', 'stable', {}), 'url': '/OS-FEDERATION/domains'}], {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/auth_domains', 'stable', {})), (<class 'keystone.api.auth.AuthSystemResource'>, '/auth/system', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/auth_system', 'stable', {})), (<class 'keystone.api.auth.AuthCatalogResource'>, '/auth/catalog', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/auth_catalog', 'stable', {})), (<class 'keystone.api.auth.AuthTokenOSPKIResource'>, '/auth/tokens/OS-PKI/revoked', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-PKI/1.0/rel/revocations', 'stable', {})), (<class 'keystone.api.auth.AuthTokenResource'>, '/auth/tokens', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/auth_tokens', 'stable', {}))]

resources = []

class keystone.api.auth.**AuthCatalogResource**

Bases: _AuthFederationWebSSOBase

get()

Get service catalog for token.

GET/HEAD /v3/auth/catalog

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.auth.**AuthDomainsResource**

Bases: *ResourceBase*

collection_key: str = 'domains'

get()

Get possible domain scopes for token.

GET/HEAD /v3/auth/domains GET/HEAD /v3/OS-FEDERATION/domains

member_key: str = 'domain'

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.auth.**AuthFederationAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.auth.AuthFederationSaml2Resource'>, '/auth/OS-FEDERATION/saml2', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/saml2', 'stable', {})), (<class 'keystone.api.auth.AuthFederationSaml2ECPResource'>, '/auth/OS-FEDERATION/saml2/ecp', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/ecp', 'stable', {})), (<class 'keystone.api.auth.AuthFederationWebSSOResource'>, '/auth/OS-FEDERATION/websso/<string:protocol_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/websso', 'stable', {'protocol_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/param/protocol_id'})), (<class 'keystone.api.auth.AuthFederationWebSSOIDPsResource'>, '/auth/OS-FEDERATION/identity_providers/<string:idp_id>/protocols/<string:protocol_id>/websso', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/identity_providers_websso', 'stable', {'idp_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/param/idp_id', 'protocol_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/param/protocol_id'}))]

resources = []

class keystone.api.auth.**AuthFederationSaml2ECPResource**

Bases: _AuthFederationWebSSOBase

get()

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

post()

Exchange a scoped token for an ECP assertion.

POST /v3/auth/OS-FEDERATION/saml2/ecp

class keystone.api.auth.**AuthFederationSaml2Resource**

Bases: _AuthFederationWebSSOBase

get()

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

post()

Exchange a scoped token for a SAML assertion.

POST /v3/auth/OS-FEDERATION/saml2

class keystone.api.auth.**AuthFederationWebSSOIDPsResource**

Bases: _AuthFederationWebSSOBase

get(*idp_id*, *protocol_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

post(*idp_id*, *protocol_id*)

class keystone.api.auth.**AuthFederationWebSSOResource**

Bases: _AuthFederationWebSSOBase

get(*protocol_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

post(*protocol_id*)

class keystone.api.auth.**AuthProjectsResource**

Bases: *ResourceBase*

collection_key: str = 'projects'

get()

Get possible project scopes for token.

GET/HEAD /v3/auth/projects GET/HEAD /v3/OS-FEDERATION/projects

member_key: str = 'project'

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.auth.**AuthSystemResource**

Bases: _AuthFederationWebSSOBase

get()

Get possible system scopes for token.

GET/HEAD /v3/auth/system

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.auth.**AuthTokenOSPKIResource**

Bases: Resource

get()

Deprecated; get revoked token list.

GET/HEAD /v3/auth/tokens/OS-PKI/revoked

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.auth.**AuthTokenResource**

Bases: _AuthFederationWebSSOBase

delete()

Revoke a token.

DELETE /v3/auth/tokens

get()

Validate a token.

HEAD/GET /v3/auth/tokens

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

post()

Issue a token.

POST /v3/auth/tokens

keystone.api.credentials module

class keystone.api.credentials.**CredentialAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = []

resources = [<class 'keystone.api.credentials.CredentialResource'>]

class keystone.api.credentials.**CredentialResource**

Bases: *ResourceBase*

collection_key: str = 'credentials'

delete(*credential_id*)

get(*credential_id=None*)

member_key: str = 'credential'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

patch(*credential_id*)

post()

keystone.api.discovery module

class keystone.api.discovery.**DiscoveryAPI**

Bases: object

static instantiate_and_register_to_app(*flask_app*)

class keystone.api.discovery.**MimeTypes**

Bases: object

JSON = 'application/json'

JSON_HOME = 'application/json-home'

keystone.api.discovery.**get_version_v3**()

keystone.api.discovery.**get_versions**()

keystone.api.discovery.**v3_mime_type_best_match**()

keystone.api.domains module

class keystone.api.domains.**DefaultConfigGroupResource**

Bases: Resource

get(*group=None*)

Get default domain group config.

GET/HEAD /v3/domains/config/{group}/default

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.domains.**DefaultConfigOptionResource**

Bases: Resource

get(*group=None*, *option=None*)

Get default domain group option config.

GET/HEAD /v3/domains/config/{group}/{option}/default

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.domains.**DefaultConfigResource**

Bases: Resource

get()

Get default domain config.

GET/HEAD /v3/domains/config/default

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.domains.**DomainAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

CONFIG_GROUP = 'https://docs.openstack.org/api/openstack-identity/3/param/config_group'

CONFIG_OPTION = 'https://docs.openstack.org/api/openstack-identity/3/param/config_option'

```
resource_mapping = [(<class 'keystone.api.domains.DomainConfigResource'>,
'/domains/<string:domain_id>/config', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/domain_config',
'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id'})),
(<class 'keystone.api.domains.DomainConfigGroupResource'>,
'/domains/<string:domain_id>/config/<string:group>', None, {}, ('https://
docs.openstack.org/api/openstack-identity/3/rel/domain_config_group',
'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'group':  'https://docs.openstack.org/api/openstack-identity/3/param/
config_group'})), (<class
'keystone.api.domains.DomainConfigOptionResource'>,
'/domains/<string:domain_id>/config/<string:group>/<string:option>', None,
{}, ('https://docs.openstack.org/api/openstack-identity/3/rel/
domain_config_option', 'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'group':
'https://docs.openstack.org/api/openstack-identity/3/param/config_group',
'option':  'https://docs.openstack.org/api/openstack-identity/3/param/
config_option'})), (<class 'keystone.api.domains.DefaultConfigResource'>,
'/domains/config/default', None, {}, ('https://docs.openstack.org/api/
openstack-identity/3/rel/domain_config_default', 'stable', {})), (<class
'keystone.api.domains.DefaultConfigGroupResource'>,
'/domains/config/<string:group>/default', None, {}, ('https://docs.
openstack.org/api/openstack-identity/3/rel/domain_config_default_group',
'stable', {'group':  'https://docs.openstack.org/api/openstack-identity/3/
param/config_group'})), (<class
'keystone.api.domains.DefaultConfigOptionResource'>,
'/domains/config/<string:group>/<string:option>/default', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/
domain_config_default_option', 'stable', {'group':
'https://docs.openstack.org/api/openstack-identity/3/param/config_group',
'option':  'https://docs.openstack.org/api/openstack-identity/3/param/
config_option'})), (<class 'keystone.api.domains.DomainUserListResource'>,
'/domains/<string:domain_id>/users/<string:user_id>/roles', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/
domain_user_roles', 'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.domains.DomainUserResource'>, '/domains/
<string:domain_id>/users/<string:user_id>/roles/<string:role_id>', None,
{}, ('https://docs.openstack.org/api/openstack-identity/3/rel/
domain_user_role', 'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.domains.DomainGroupListResource'>,
'/domains/<string:domain_id>/groups/<string:group_id>/roles', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/
domain_group_roles', 'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'group_id':
'https://docs.openstack.org/api/openstack-identity/3/param/group_id'})),
```

resources = [<class 'keystone.api.domains.DomainResource'>]

class keystone.api.domains.**DomainConfigBase**

Bases: *ResourceBase*

delete(*domain_id=None*, *group=None*, *option=None*)

Delete domain config.

DELETE /v3/domains/{domain_id}/config DELETE /v3/domains/{domain_id}/config/{group} DELETE /v3/domains/{domain_id}/config/{group}/{option}

get(*domain_id=None*, *group=None*, *option=None*)

Check if config option exists.

GET/HEAD /v3/domains/{domain_id}/config GET/HEAD /v3/domains/{domain_id}/config/{group} GET/HEAD /v3/domains/{domain_id}/config/{group}/{option}

member_key: str = 'config'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*domain_id=None*, *group=None*, *option=None*)

Update domain config option.

PATCH /v3/domains/{domain_id}/config PATCH /v3/domains/{domain_id}/config/{group} PATCH /v3/domains/{domain_id}/config/{group}/{option}

class keystone.api.domains.**DomainConfigGroupResource**

Bases: *DomainConfigBase*

Provides config group routing functionality.

This class leans on DomainConfigBase to provide the following APIs:

GET/HEAD /v3/domains/{domain_id}/config/{group} PATCH /v3/domains/{domain_id}/config/{group} DELETE /v3/domains/{domain_id}/config/{group}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.domains.**DomainConfigOptionResource**

Bases: *DomainConfigBase*

Provides config option routing functionality.

This class leans on DomainConfigBase to provide the following APIs:

GET/HEAD /v3/domains/{domain_id}/config/{group}/{option} PATCH /v3/domains/{domain_id}/config/{group}/{option} DELETE /v3/domains/{domain_id}/config/{group}/{option}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.domains.**DomainConfigResource**

Bases: *DomainConfigBase*

Provides config routing functionality.

This class leans on DomainConfigBase to provide the following APIs:

GET/HEAD /v3/domains/{domain_id}/config PATCH /v3/domains/{domain_id}/config DELETE /v3/domains/{domain_id}/config

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*domain_id*)

Create domain config.

PUT /v3/domains/{domain_id}/config

class keystone.api.domains.**DomainGroupListResource**

Bases: Resource

get(*domain_id=None*, *group_id=None*)

List all domain grats for a specific group.

GET/HEAD /v3/domains/{domain_id}/groups/{group_id}/roles

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.domains.**DomainGroupResource**

Bases: *ResourceBase*

collection_key: str = 'grants'

delete(*domain_id=None*, *group_id=None*, *role_id=None*)

Revoke a role from a group on a domain.

DELETE /v3/domains/{domain_id}/groups/{group_id}/roles/{role_id}

get(*domain_id=None*, *group_id=None*, *role_id=None*)

Check if a group has a specific role on a domain.

GET/HEAD /v3/domains/{domain_id}/groups/{group_id}/roles/{role_id}

member_key: str = 'grant'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*domain_id=None*, *group_id=None*, *role_id=None*)

Grant a role to a group on a domain.

PUT /v3/domains/{domain_id}/groups/{group_id}/roles/{role_id}

class keystone.api.domains.**DomainResource**

Bases: *ResourceBase*

collection_key: str = 'domains'

delete(*domain_id*)

Delete domain.

DELETE /v3/domains/{domain_id}

get(*domain_id=None*)

Get domain or list domains.

GET/HEAD /v3/domains GET/HEAD /v3/domains/{domain_id}

get_member_from_driver

member_key: str = 'domain'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*domain_id*)

Update domain.

PATCH /v3/domains/{domain_id}

post()

Create domain.

POST /v3/domains

class keystone.api.domains.**DomainUserListResource**

Bases: `Resource`

get(*domain_id=None*, *user_id=None*)

Get user grant.

GET/HEAD /v3/domains/{domain_id}/users/{user_id}/roles

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.domains.**DomainUserResource**

Bases: *ResourceBase*

collection_key: str = 'grants'

delete(*domain_id=None*, *user_id=None*, *role_id=None*)

Revoke a role from user on a domain.

DELETE /v3/domains/{domain_id}/users/{user_id}/roles/{role_id}

get(*domain_id=None*, *user_id=None*, *role_id=None*)

Check if a user has a specific role on the domain.

GET/HEAD /v3/domains/{domain_id}/users/{user_id}/roles/{role_id}

member_key: str = 'grant'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*domain_id=None*, *user_id=None*, *role_id=None*)

Create a role to a user on a domain.

PUT /v3/domains/{domain_id}/users/{user_id}/roles/{role_id}

keystone.api.ec2tokens module

class keystone.api.ec2tokens.**EC2TokensAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.ec2tokens.EC2TokensResource'>, '/ec2tokens', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EC2/1.0/rel/ec2tokens', 'stable', {}))]

resources = []

class keystone.api.ec2tokens.**EC2TokensResource**

Bases: ResourceBase

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

post()

Authenticate ec2 token.

POST /v3/ec2tokens

keystone.api.endpoints module

class keystone.api.endpoints.**EndpointAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.endpoints.EndpointPolicyEndpointResource'>, '/endpoints/<string:endpoint_id>/OS-ENDPOINT-POLICY/policy', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-ENDPOINT-POLICY/1.0/rel/endpoint_policy', 'stable', {'endpoint_id': 'https://docs.openstack.org/api/openstack-identity/3/param/endpoint_id'}))]

resources = [<class 'keystone.api.endpoints.EndpointResource'>]

class keystone.api.endpoints.**EndpointPolicyEndpointResource**

Bases: Resource

get(*endpoint_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.endpoints.**EndpointResource**

Bases: *ResourceBase*

collection_key: str = 'endpoints'

delete(*endpoint_id*)

get(*endpoint_id=None*)

get_member_from_driver

member_key: str = 'endpoint'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*endpoint_id*)

post()

keystone.api.groups module

class keystone.api.groups.**GroupAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.groups.GroupUsersResource'>, '/groups/<string:group_id>/users', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/group_users', 'stable', {'group_id': 'https://docs.openstack.org/api/openstack-identity/3/param/group_id'})), (<class 'keystone.api.groups.UserGroupCRUDResource'>, '/groups/<string:group_id>/users/<string:user_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/group_user', 'stable', {'group_id': 'https://docs.openstack.org/api/openstack-identity/3/param/group_id', 'user_id': 'https://docs.openstack.org/api/openstack-identity/3/param/user_id'}))]

resources = [<class 'keystone.api.groups.GroupsResource'>]

class keystone.api.groups.**GroupUsersResource**

Bases: *ResourceBase*

get(*group_id*)

Get list of users in group.

GET/HEAD /groups/{group_id}/users

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.groups.**GroupsResource**

Bases: *ResourceBase*

collection_key: str = 'groups'

delete(*group_id*)

Delete group.

DELETE /groups/{group_id}

get(*group_id=None*)

get_member_from_driver

member_key: str = 'group'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*group_id*)

Update group.

PATCH /groups/{group_id}

post()

Create group.

POST /groups

class keystone.api.groups.**UserGroupCRUDResource**

Bases: Resource

delete(*group_id*, *user_id*)

Remove user from group.

DELETE /groups/{group_id}/users/{user_id}

get(*group_id*, *user_id*)

Check if a user is in a group.

GET/HEAD /groups/{group_id}/users/{user_id}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

put(*group_id*, *user_id*)

Add user to group.

PUT /groups/{group_id}/users/{user_id}

keystone.api.limits module

class keystone.api.limits.**LimitModelResource**

Bases: Resource

get()

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.limits.**LimitsAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.limits.LimitModelResource'>, '/limits/model', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/limit_model', 'experimental', {}))]

resources = [<class 'keystone.api.limits.LimitsResource'>]

class keystone.api.limits.**LimitsResource**

Bases: *ResourceBase*

collection_key: str = 'limits'

delete(*limit_id*)

get(*limit_id=None*)

get_member_from_driver

json_home_resource_status = 'experimental'

member_key: str = 'limit'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*limit_id*)

post()

keystone.api.os_ep_filter module

class keystone.api.os_ep_filter.**EPFilterAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

```
resource_mapping = [(<class
'keystone.api.os_ep_filter.EPFilterEndpointProjectsResource'>,
'/endpoints/<string:endpoint_id>/projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EP-FILTER/1.
0/rel/endpoint_projects', 'stable', {'endpoint_id':
'https://docs.openstack.org/api/openstack-identity/3/param/
endpoint_id'})), (<class
'keystone.api.os_ep_filter.EPFilterProjectsEndpointsResource'>,
'/projects/<string:project_id>/endpoints/<string:endpoint_id>', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EP-FILTER/1.
0/rel/project_endpoint', 'stable', {'endpoint_id':
'https://docs.openstack.org/api/openstack-identity/3/param/endpoint_id',
'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id'})),
(<class 'keystone.api.os_ep_filter.EPFilterProjectEndpointsListResource'>,
'/projects/<string:project_id>/endpoints', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EP-FILTER/1.
0/rel/project_endpoints', 'stable', {'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id'})),
(<class 'keystone.api.os_ep_filter.
EndpointFilterProjectEndpointGroupsListResource'>,
'/projects/<string:project_id>/endpoint_groups', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EP-FILTER/1.
0/rel/project_endpoint_groups', 'stable', {'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id'})),
(<class 'keystone.api.os_ep_filter.EndpointFilterEPGroupsEndpoints'>,
'/endpoint_groups/<string:endpoint_group_id>/endpoints', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EP-FILTER/1.
0/rel/endpoints_in_endpoint_group', 'stable', {'endpoint_group_id':
'https://docs.openstack.org/api/openstack-identity/3/ext/OS-EP-FILTER/1.0/
param/endpoint_group_id'})), (<class
'keystone.api.os_ep_filter.EndpointFilterEPGroupsProjects'>,
'/endpoint_groups/<string:endpoint_group_id>/projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EP-FILTER/1.
0/rel/projects_associated_with_endpoint_group', 'stable',
{'endpoint_group_id':  'https://docs.openstack.org/api/openstack-identity/
3/ext/OS-EP-FILTER/1.0/param/endpoint_group_id'})), (<class
'keystone.api.os_ep_filter.EPFilterGroupsProjectsResource'>, '/
endpoint_groups/<string:endpoint_group_id>/projects/<string:project_id>',
None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/
OS-EP-FILTER/1.0/rel/endpoint_group_to_project_association', 'stable',
{'endpoint_group_id':  'https://docs.openstack.org/api/openstack-identity/
3/ext/OS-EP-FILTER/1.0/param/endpoint_group_id', 'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id'}))]

resources = [<class 'keystone.api.os_ep_filter.EndpointGroupsResource'>]
```

class keystone.api.os_ep_filter.**EPFilterEndpointProjectsResource**

Bases: Resource

get(*endpoint_id*)

Return a list of projects associated with the endpoint.

methods: `t.ClassVar[t.Collection[str] | None] = {'GET'}`

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.os_ep_filter.**EPFilterGroupsProjectsResource**

Bases: *ResourceBase*

collection_key: `str = 'project_endpoint_groups'`

delete(*endpoint_group_id*, *project_id*)

get(*endpoint_group_id*, *project_id*)

member_key: `str = 'project_endpoint_group'`

methods: `t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}`

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*endpoint_group_id*, *project_id*)

class keystone.api.os_ep_filter.**EPFilterProjectEndpointsListResource**

Bases: `Resource`

get(*project_id*)

methods: `t.ClassVar[t.Collection[str] | None] = {'GET'}`

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.os_ep_filter.**EPFilterProjectsEndpointsResource**

Bases: `Resource`

delete(*project_id*, *endpoint_id*)

get(*project_id*, *endpoint_id*)

methods: `t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}`

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*project_id*, *endpoint_id*)

class keystone.api.os_ep_filter.**EndpointFilterEPGroupsEndpoints**

Bases: `Resource`

get(*endpoint_group_id*)

methods: `t.ClassVar[t.Collection[str] | None] = {'GET'}`

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.os_ep_filter.**EndpointFilterEPGroupsProjects**

Bases: `Resource`

get(*endpoint_group_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.os_ep_filter.EndpointFilterProjectEndpointGroupsListResource

Bases: Resource

get(*project_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.os_ep_filter.EndpointGroupsResource

Bases: *ResourceBase*

api_prefix: str = '/OS-EP-FILTER'

collection_key: str = 'endpoint_groups'

delete(*endpoint_group_id*)

get(*endpoint_group_id=None*)

json_home_parameter_rel_func(*, *extension_version='1.0'*, *parameter_name*)

json_home_resource_rel_func(*, *extension_version='1.0'*, *resource_name*)

member_key: str = 'endpoint_group'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

patch(*endpoint_group_id*)

post()

keystone.api.os_federation module

class keystone.api.os_federation.IDPProtocolsCRUDResource

Bases: _IdentityProvidersProtocolsResourceBase

delete(*idp_id*, *protocol_id*)

Delete a protocol from an IDP.

DELETE /OS-FEDERATION/identity_providers/
{idp_id}/protocols/{protocol_id}

get(*idp_id*, *protocol_id*)

Get protocols for an IDP.

HEAD/GET /OS-FEDERATION/identity_providers/
{idp_id}/protocols/{protocol_id}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*idp_id*, *protocol_id*)

Update protocol for an IDP.

PATCH /OS-FEDERATION/identity_providers/
{idp_id}/protocols/{protocol_id}

put(*idp_id*, *protocol_id*)

Create protocol for an IDP.

PUT /OS-Federation/identity_providers/{idp_id}/protocols/{protocol_id}

class keystone.api.os_federation.**IDPProtocolsListResource**

Bases: `_IdentityProvidersProtocolsResourceBase`

get(*idp_id*)

List protocols for an IDP.

HEAD/GET /OS-FEDERATION/identity_providers/{idp_id}/protocols

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.os_federation.**IdentityProvidersResource**

Bases: `_ResourceBase`

api_prefix: str = '/OS-FEDERATION'

collection_key: str = 'identity_providers'

delete(*idp_id*)

get(*idp_id=None*)

member_key: str = 'identity_provider'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*idp_id*)

put(*idp_id*)

Create an idp resource for federated authentication.

PUT /OS-FEDERATION/identity_providers/{idp_id}

class keystone.api.os_federation.**MappingResource**

Bases: `_ResourceBase`

api_prefix: str = '/OS-FEDERATION'

collection_key: str = 'mappings'

delete(*mapping_id*)

Delete a mapping.

DELETE /OS-FEDERATION/mappings/{mapping_id}

get(*mapping_id=None*)

member_key: str = 'mapping'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*mapping_id*)

Update an attribute mapping for identity federation.

PATCH /OS-FEDERATION/mappings/{mapping_id}

put(*mapping_id*)

Create a mapping.

PUT /OS-FEDERATION/mappings/{mapping_id}

class keystone.api.os_federation.OSFederationAPI(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.os_federation.SAML2MetadataResource'>, '/saml2/metadata', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/metadata', 'stable', {})), (<class 'keystone.api.os_federation.OSFederationAuthResource'>, '/identity_providers/<string:idp_id>/protocols/<string:protocol_id>/auth', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/identity_provider_protocol_auth', 'stable', {'idp_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/param/idp_id', 'protocol_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/param/protocol_id'}))]

resources = []

class keystone.api.os_federation.OSFederationAuthResource

Bases: `Resource`

get(*idp_id*, *protocol_id*)

Authenticate from dedicated uri endpoint.

GET/HEAD /OS-FEDERATION/identity_providers/
{idp_id}/protocols/{protocol_id}/auth

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

post(*idp_id*, *protocol_id*)

Authenticate from dedicated uri endpoint.

POST /OS-FEDERATION/identity_providers/
{idp_id}/protocols/{protocol_id}/auth

class keystone.api.os_federation.**OSFederationIdentityProvidersAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/jso*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = []

resources = [<class 'keystone.api.os_federation.IdentityProvidersResource'>]

class keystone.api.os_federation.**OSFederationIdentityProvidersProtocolsAPI**(*blueprint_url_prefix=*, *api_url_prefix=''*, *default_mediatype='app*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.os_federation.IDPProtocolsCRUDResource'>, '/OS-FEDERATION/identity_providers/<string:idp_id>/protocols/<string:protocol_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/identity_provider_protocol', 'stable', {'idp_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/param/idp_id', 'protocol_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/param/protocol_id'})), (<class 'keystone.api.os_federation.IDPProtocolsListResource'>, '/OS-FEDERATION/identity_providers/<string:idp_id>/protocols', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/rel/identity_provider_protocols', 'stable', {'idp_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-FEDERATION/1.0/param/idp_id'}))]

resources = []

class keystone.api.os_federation.**OSFederationMappingsAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = []

resources = [<class 'keystone.api.os_federation.MappingResource'>]

class keystone.api.os_federation.**OSFederationServiceProvidersAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = []

resources = [<class 'keystone.api.os_federation.ServiceProvidersResource'>]

class keystone.api.os_federation.**SAML2MetadataResource**

Bases: Resource

get()

Get SAML2 metadata.

GET/HEAD /OS-FEDERATION/saml2/metadata

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.os_federation.**ServiceProvidersResource**

Bases: _ResourceBase

api_prefix: str = '/OS-FEDERATION'

collection_key: str = 'service_providers'

delete(*sp_id*)

Delete a service provider.

DELETE /OS-FEDERATION/service_providers/{sp_id}

get(*sp_id=None*)

member_key: str = 'service_provider'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*sp_id*)

Update a service provider.

PATCH /OS-FEDERATION/service_providers/{sp_id}

put(*sp_id*)

Create a service provider.

PUT /OS-FEDERATION/service_providers/{sp_id}

keystone.api.os_inherit module

class keystone.api.os_inherit.**OSInheritAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

```
resource_mapping = [(<class
'keystone.api.os_inherit.OSInheritDomainGroupRolesResource'>,
'/domains/<string:domain_id>/groups/<string:group_id>/roles/
<string:role_id>/inherited_to_projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-INHERIT/1.0/
rel/domain_group_role_inherited_to_projects', 'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'group_id':
'https://docs.openstack.org/api/openstack-identity/3/param/group_id',
'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id'})),
(<class 'keystone.api.os_inherit.OSInheritDomainGroupRolesListResource'>,
'/domains/<string:domain_id>/groups/<string:group_id>/roles/
inherited_to_projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-INHERIT/1.0/
rel/domain_group_roles_inherited_to_projects', 'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'group_id':
'https://docs.openstack.org/api/openstack-identity/3/param/group_id'})),
(<class 'keystone.api.os_inherit.OSInheritDomainUserRolesResource'>,
'/domains/<string:domain_id>/users/<string:user_id>/roles/
<string:role_id>/inherited_to_projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-INHERIT/1.0/
rel/domain_user_role_inherited_to_projects', 'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.os_inherit.OSInheritDomainUserRolesListResource'>,
'/domains/<string:domain_id>/users/<string:user_id>/roles/
inherited_to_projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-INHERIT/1.0/
rel/domain_user_roles_inherited_to_projects', 'stable', {'domain_id':
'https://docs.openstack.org/api/openstack-identity/3/param/domain_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.os_inherit.OSInheritProjectUserResource'>,
'/projects/<string:project_id>/users/<string:user_id>/roles/
<string:role_id>/inherited_to_projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-INHERIT/1.0/
rel/project_user_role_inherited_to_projects', 'stable', {'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id',
'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.os_inherit.OSInheritProjectGroupResource'>,
'/projects/<string:project_id>/groups/<string:group_id>/roles/
<string:role_id>/inherited_to_projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-INHERIT/1.0/
rel/project_group_role_inherited_to_projects', 'stable', {'group_id':
'https://docs.openstack.org/api/openstack-identity/3/param/group_id',
'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id',
'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id'})]
```

resources = []

class keystone.api.os_inherit.**OSInheritDomainGroupRolesListResource**

Bases: Resource

get(*domain_id*, *group_id*)

List roles (inherited) for a group on a domain.

GET/HEAD /OS-INHERIT/domains/{domain_id}/groups/{group_id}
/roles/inherited_to_projects

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.os_inherit.**OSInheritDomainGroupRolesResource**

Bases: Resource

delete(*domain_id*, *group_id*, *role_id*)

Revoke an inherited grant for a group on a domain.

DELETE /OS-INHERIT/domains/{domain_id}/groups/{group_id}
/roles/{role_id}/inherited_to_projects

get(*domain_id*, *group_id*, *role_id*)

Check for an inherited grant for a group on a domain.

GET/HEAD /OS-INHERIT/domains/{domain_id}/groups/{group_id}
/roles/{role_id}/inherited_to_projects

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

put(*domain_id*, *group_id*, *role_id*)

Create an inherited grant for a group on a domain.

PUT /OS-INHERIT/domains/{domain_id}/groups/{group_id}
/roles/{role_id}/inherited_to_projects

class keystone.api.os_inherit.**OSInheritDomainUserRolesListResource**

Bases: Resource

get(*domain_id*, *user_id*)

List roles (inherited) for a user on a domain.

GET/HEAD /OS-INHERIT/domains/{domain_id}/users/{user_id}
/roles/inherited_to_projects

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.os_inherit.**OSInheritDomainUserRolesResource**

Bases: Resource

delete(*domain_id*, *user_id*, *role_id*)

Revoke a grant from a user on a domain.

DELETE /OS-INHERIT/domains/{domain_id}/users/{user_id}/roles
/{role_id}/inherited_to_projects

get(*domain_id*, *user_id*, *role_id*)

Check for an inherited grant for a user on a domain.

GET/HEAD /OS-INHERIT/domains/{domain_id}/users/{user_id}/roles
/{role_id}/inherited_to_projects

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*domain_id*, *user_id*, *role_id*)

Create an inherited grant for a user on a domain.

PUT /OS-INHERIT/domains/{domain_id}/users/{user_id}/roles/{role_id}
/inherited_to_projects

class keystone.api.os_inherit.**OSInheritProjectGroupResource**

Bases: Resource

delete(*project_id*, *group_id*, *role_id*)

Revoke an inherited grant for a group on a project.

DELETE /OS-INHERIT/projects/{project_id}/groups/{group_id}
/roles/{role_id}/inherited_to_projects

get(*project_id*, *group_id*, *role_id*)

Check for an inherited grant for a group on a project.

GET/HEAD /OS-INHERIT/projects/{project_id}/groups/{group_id}
/roles/{role_id}/inherited_to_projects

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*project_id*, *group_id*, *role_id*)

Create an inherited grant for a group on a project.

PUT /OS-INHERIT/projects/{project_id}/groups/{group_id}
/roles/{role_id}/inherited_to_projects

class keystone.api.os_inherit.**OSInheritProjectUserResource**

Bases: Resource

delete(*project_id*, *user_id*, *role_id*)

Revoke an inherited grant for a user on a project.

DELETE /OS-INHERIT/projects/{project_id}/users/{user_id}
/roles/{role_id}/inherited_to_projects

get(*project_id*, *user_id*, *role_id*)

Check for an inherited grant for a user on a project.

GET/HEAD /OS-INHERIT/projects/{project_id}/users/{user_id}
/roles/{role_id}/inherited_to_projects

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*project_id*, *user_id*, *role_id*)

Create an inherited grant for a user on a project.

PUT /OS-INHERIT/projects/{project_id}/users/{user_id}
/roles/{role_id}/inherited_to_projects

keystone.api.os_oauth1 module

class keystone.api.os_oauth1.**AccessTokenResource**

Bases: `_OAuth1ResourceBase`

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

post()

class keystone.api.os_oauth1.**AuthorizeResource**

Bases: `_OAuth1ResourceBase`

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*request_token_id*)

class keystone.api.os_oauth1.**ConsumerResource**

Bases: *ResourceBase*

api_prefix: str = '/OS-OAUTH1'

collection_key: str = 'consumers'

delete(*consumer_id*)

get(*consumer_id=None*)

json_home_parameter_rel_func(*, *extension_version='1.0'*, *parameter_name*)

json_home_resource_rel_func(*, *extension_version='1.0'*, *resource_name*)

member_key: str = 'consumer'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

patch(*consumer_id*)

post()

class keystone.api.os_oauth1.**OSAuth1API**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.os_oauth1.RequestTokenResource'>, '/request_token', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/rel/request_tokens', 'stable', {})), (<class 'keystone.api.os_oauth1.AccessTokenResource'>, '/access_token', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/rel/access_tokens', 'stable', {})), (<class 'keystone.api.os_oauth1.AuthorizeResource'>, '/authorize/<string:request_token_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/rel/authorize_request_token', 'stable', {'request_token_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/param/request_token_id'}))]

resources = [<class 'keystone.api.os_oauth1.ConsumerResource'>]

class keystone.api.os_oauth1.**RequestTokenResource**

Bases: _OAuth1ResourceBase

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

post()

keystone.api.os_oauth2 module

class keystone.api.os_oauth2.**AccessTokenResource**

Bases: *ResourceBase*

delete()

The method is not allowed.

get()

The method is not allowed.

head()

The method is not allowed.

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'HEAD', 'PATCH', 'POST', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch()

The method is not allowed.

post()

Get an OAuth2.0 Access Token.

POST /v3/OS-OAUTH2/token

put()

The method is not allowed.

class keystone.api.os_oauth2.**OSAuth2API**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.os_oauth2.AccessTokenResource'>, '/token', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH2/1.0/rel/token', 'stable', {}))]

keystone.api.os_revoke module

class keystone.api.os_revoke.**OSRevokeAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.os_revoke.OSRevokeResource'>, '/events', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-REVOKE/1.0/rel/events', 'stable', {}))]

resources = []

class keystone.api.os_revoke.**OSRevokeResource**

Bases: `Resource`

get()

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

keystone.api.os_simple_cert module

class keystone.api.os_simple_cert.**SimpleCertAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.os_simple_cert.SimpleCertCAResource'>, '/OS-SIMPLE-CERT/ca', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-SIMPLE-CERT/1.0/rel/ca_certificate', 'stable', {})), (<class 'keystone.api.os_simple_cert.SimpleCertListResource'>, '/OS-SIMPLE-CERT/certificates', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-SIMPLE-CERT/1.0/rel/certificates', 'stable', {}))]

resources = []

class keystone.api.os_simple_cert.**SimpleCertCAResource**

Bases: Resource

get()

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.os_simple_cert.**SimpleCertListResource**

Bases: Resource

get()

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

keystone.api.policy module

class keystone.api.policy.**EndpointPolicyAssociations**

Bases: Resource

delete(*policy_id*, *endpoint_id*)

get(*policy_id*, *endpoint_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

put(*policy_id*, *endpoint_id*)

class keystone.api.policy.**EndpointPolicyResource**

Bases: Resource

get(*policy_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.policy.**PolicyAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.policy.EndpointPolicyResource'>, '/policies/<string:policy_id>/OS-ENDPOINT-POLICY/endpoints', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-ENDPOINT-POLICY/1.0/rel/policy_endpoints', 'stable', {'policy_id': 'https://docs.openstack.org/api/openstack-identity/3/param/policy_id'})), (<class 'keystone.api.policy.EndpointPolicyAssociations'>, '/policies/<string:policy_id>/OS-ENDPOINT-POLICY/endpoints/<string:endpoint_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-ENDPOINT-POLICY/1.0/rel/endpoint_policy_association', 'stable', {'endpoint_id': 'https://docs.openstack.org/api/openstack-identity/3/param/endpoint_id', 'policy_id': 'https://docs.openstack.org/api/openstack-identity/3/param/policy_id'})), (<class 'keystone.api.policy.ServicePolicyAssociations'>, '/policies/<string:policy_id>/OS-ENDPOINT-POLICY/services/<string:service_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-ENDPOINT-POLICY/1.0/rel/service_policy_association', 'stable', {'policy_id': 'https://docs.openstack.org/api/openstack-identity/3/param/policy_id', 'service_id': 'https://docs.openstack.org/api/openstack-identity/3/param/service_id'})), (<class 'keystone.api.policy.ServiceRegionPolicyAssociations'>, '/policies/<string:policy_id>/OS-ENDPOINT-POLICY/services/<string:service_id>/regions/<string:region_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-ENDPOINT-POLICY/1.0/rel/region_and_service_policy_association', 'stable', {'policy_id': 'https://docs.openstack.org/api/openstack-identity/3/param/policy_id', 'region_id': 'https://docs.openstack.org/api/openstack-identity/3/param/region_id', 'service_id': 'https://docs.openstack.org/api/openstack-identity/3/param/service_id'}))]

resources = [<class 'keystone.api.policy.PolicyResource'>]

class keystone.api.policy.**PolicyResource**

Bases: *ResourceBase*

collection_key: str = 'policies'

delete(*policy_id*)

get(*policy_id=None*)

member_key: str = 'policy'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*policy_id*)

post()

class keystone.api.policy.**ServicePolicyAssociations**

Bases: `Resource`

delete(*policy_id*, *service_id*)

get(*policy_id*, *service_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*policy_id*, *service_id*)

class keystone.api.policy.**ServiceRegionPolicyAssociations**

Bases: `Resource`

delete(*policy_id*, *service_id*, *region_id*)

get(*policy_id*, *service_id*, *region_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*policy_id*, *service_id*, *region_id*)

keystone.api.projects module

class keystone.api.projects.**ProjectAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

```
resource_mapping = [(<class 'keystone.api.projects.ProjectTagsResource'>,
'/projects/<string:project_id>/tags', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/project_tags',
'stable', {'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id'})),
(<class 'keystone.api.projects.ProjectTagResource'>,
'/projects/<string:project_id>/tags/<string:value>', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/project_tags',
'stable', {'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id',
'value':
'https://docs.openstack.org/api/openstack-identity/3/param/tag_value'})),
(<class 'keystone.api.projects.ProjectUserGrantResource'>, '/projects/
<string:project_id>/users/<string:user_id>/roles/<string:role_id>', None,
{}, ('https://docs.openstack.org/api/openstack-identity/3/rel/
project_user_role', 'stable', {'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id',
'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.projects.ProjectUserListGrantResource'>,
'/projects/<string:project_id>/users/<string:user_id>/roles', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/
project_user_roles', 'stable', {'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.projects.ProjectGroupGrantResource'>, '/projects/
<string:project_id>/groups/<string:group_id>/roles/<string:role_id>',
None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/
project_group_role', 'stable', {'group_id':
'https://docs.openstack.org/api/openstack-identity/3/param/group_id',
'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id',
'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id'})),
(<class 'keystone.api.projects.ProjectGroupListGrantResource'>,
'/projects/<string:project_id>/groups/<string:group_id>/roles', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/
project_group_roles', 'stable', {'group_id':
'https://docs.openstack.org/api/openstack-identity/3/param/group_id',
'project_id':
'https://docs.openstack.org/api/openstack-identity/3/param/project_id'}))]

resources = [<class 'keystone.api.projects.ProjectResource'>]
```

class keystone.api.projects.**ProjectGroupGrantResource**

Bases: _ProjectGrantResourceBase

delete(*project_id*, *group_id*, *role_id*)

Delete grant of role for group on project.

DELETE /v3/projects/{project_id}/groups/{group_id}/roles/{role_id}

get(*project_id*, *group_id*, *role_id*)

Check grant for project, group, role.

GET/HEAD /v3/projects/{project_id/groups/{group_id}/roles/{role_id}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*project_id*, *group_id*, *role_id*)

Grant role for group on project.

PUT /v3/projects/{project_id}/groups/{group_id}/roles/{role_id}

class keystone.api.projects.**ProjectGroupListGrantResource**

Bases: `_ProjectGrantResourceBase`

get(*project_id*, *group_id*)

List grants for group on project.

GET/HEAD /v3/projects/{project_id}/groups/{group_id}

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.projects.**ProjectResource**

Bases: *ResourceBase*

collection_key: str = 'projects'

delete(*project_id*)

Delete project.

DELETE /v3/projects/{project_id}

get(*project_id=None*)

Get project or list projects.

GET/HEAD /v3/projects GET/HEAD /v3/projects/{project_id}

get_member_from_driver

member_key: str = 'project'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*project_id*)

Update project.

PATCH /v3/projects/{project_id}

post()

Create project.

POST /v3/projects

class keystone.api.projects.**ProjectTagResource**

Bases: _ProjectTagResourceBase

delete(*project_id*, *value*)

Delete a single tag from a project.

/v3/projects/{project_id}/tags/{value}

get(*project_id*, *value*)

Get information for a single tag associated with a given project.

GET /v3/projects/{project_id}/tags/{value}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

put(*project_id*, *value*)

Add a single tag to a project.

PUT /v3/projects/{project_id}/tags/{value}

class keystone.api.projects.**ProjectTagsResource**

Bases: _ProjectTagResourceBase

delete(*project_id*)

Delete all tags associated with a given project.

DELETE /v3/projects/{project_id}/tags

get(*project_id*)

List tags associated with a given project.

GET /v3/projects/{project_id}/tags

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

put(*project_id*)

Update all tags associated with a given project.

PUT /v3/projects/{project_id}/tags

class keystone.api.projects.**ProjectUserGrantResource**

Bases: _ProjectGrantResourceBase

delete(*project_id*, *user_id*, *role_id*)

Delete grant of role for user on project.

DELETE /v3/projects/{project_id}/users/{user_id}/roles/{role_id}

get(*project_id*, *user_id*, *role_id*)

Check grant for project, user, role.

GET/HEAD /v3/projects/{project_id/users/{user_id}/roles/{role_id}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*project_id*, *user_id*, *role_id*)

Grant role for user on project.

PUT /v3/projects/{project_id}/users/{user_id}/roles/{role_id}

class keystone.api.projects.**ProjectUserListGrantResource**

Bases: `_ProjectGrantResourceBase`

get(*project_id*, *user_id*)

List grants for user on project.

GET/HEAD /v3/projects/{project_id}/users/{user_id}

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

keystone.api.regions module

class keystone.api.regions.**RegionAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = []

resources = [<class 'keystone.api.regions.RegionResource'>]

class keystone.api.regions.**RegionResource**

Bases: *ResourceBase*

collection_key: str = 'regions'

delete(*region_id*)

get(*region_id=None*)

member_key: str = 'region'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*region_id*)

post()

put(*region_id*)

keystone.api.registered_limits module

class keystone.api.registered_limits.**RegisteredLimitResource**

Bases: *ResourceBase*

collection_key: str = 'registered_limits'

delete(*registered_limit_id*)

get(*registered_limit_id=None*)

json_home_resource_status = 'experimental'

member_key: str = 'registered_limit'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*registered_limit_id*)

post()

class keystone.api.registered_limits.**RegisteredLimitsAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = []

resources = [<class 'keystone.api.registered_limits.RegisteredLimitResource'>]

keystone.api.role_assignments module

class keystone.api.role_assignments.**RoleAssignmentsAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.role_assignments.RoleAssignmentsResource'>, '/role_assignments', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/role_assignments', 'stable', {}))]

resources = []

class keystone.api.role_assignments.**RoleAssignmentsResource**

Bases: *ResourceBase*

collection_key: str = 'role_assignments'

get()

List all role assignments.

GET/HEAD /v3/role_assignments

member_key: str = 'role_assignment'

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

keystone.api.role_inferences module

class keystone.api.role_inferences.**RoleInferencesAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.role_inferences.RoleInferencesResource'>, '/role_inferences', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/role_inferences', 'stable', {}))]

resources = []

class keystone.api.role_inferences.**RoleInferencesResource**

Bases: `Resource`

get()

List role inference rules.

GET/HEAD /v3/role_inferences

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

keystone.api.roles module

class keystone.api.roles.**RoleAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.roles.RoleImplicationListResource'>, '/roles/<string:prior_role_id>/implies', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/implied_roles', 'stable', {'prior_role_id': 'https://docs.openstack.org/api/openstack-identity/3/param/role_id'})), (<class 'keystone.api.roles.RoleImplicationResource'>, '/roles/<string:prior_role_id>/implies/<string:implied_role_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/rel/implied_role', 'stable', {'implied_role_id': 'https://docs.openstack.org/api/openstack-identity/3/param/role_id', 'prior_role_id': 'https://docs.openstack.org/api/openstack-identity/3/param/role_id'}))]

resources = [<class 'keystone.api.roles.RoleResource'>]

class keystone.api.roles.**RoleImplicationListResource**

Bases: Resource

get(*prior_role_id*)

List Implied Roles.

GET/HEAD /v3/roles/{prior_role_id}/implies

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.roles.**RoleImplicationResource**

Bases: Resource

delete(*prior_role_id*, *implied_role_id*)

Delete implied role.

DELETE /v3/roles/{prior_role_id}/implies/{implied_role_id}

get(*prior_role_id*, *implied_role_id*)

Get implied role.

GET/HEAD /v3/roles/{prior_role_id}/implies/{implied_role_id}

head(*prior_role_id*, *implied_role_id=None*)

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'HEAD', 'PUT'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

put(*prior_role_id*, *implied_role_id*)

Create implied role.

PUT /v3/roles/{prior_role_id}/implies/{implied_role_id}

class keystone.api.roles.**RoleResource**

Bases: *ResourceBase*

collection_key: str = 'roles'

delete(*role_id*)

Delete role.

DELETE /v3/roles/{role_id}

get(*role_id=None*)

Get role or list roles.

GET/HEAD /v3/roles GET/HEAD /v3/roles/{role_id}

get_member_from_driver

member_key: str = 'role'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*role_id*)

Update role.

PATCH /v3/roles/{role_id}

post()

Create role.

POST /v3/roles

keystone.api.s3tokens module

class keystone.api.s3tokens.**S3Api**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.s3tokens.S3Resource'>, '/s3tokens', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/s3tokens/1.0/rel/s3tokens', 'stable', {}))]

resources = []

class keystone.api.s3tokens.**S3Resource**

Bases: ResourceBase

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

post()

Authenticate s3token.

POST /v3/s3tokens

keystone.api.services module

class keystone.api.services.**ServiceAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = []

resources = [<class 'keystone.api.services.ServicesResource'>]

class keystone.api.services.**ServicesResource**

Bases: *ResourceBase*

collection_key: str = 'services'

delete(*service_id*)

get(*service_id=None*)

member_key: str = 'service'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*service_id*)

post()

keystone.api.system module

class keystone.api.system.**SystemAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

```
resource_mapping = [(<class
'keystone.api.system.SystemUsersListResource'>,
'/system/users/<string:user_id>/roles', None, {}, ('https://docs.
openstack.org/api/openstack-identity/3/rel/system_user_roles', 'stable',
{'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'}),
(<class 'keystone.api.system.SystemUsersResource'>,
'/system/users/<string:user_id>/roles/<string:role_id>', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/
system_user_role', 'stable', {'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'}),
(<class 'keystone.api.system.SystemGroupsRolesListResource'>,
'/system/groups/<string:group_id>/roles', None, {}, ('https://docs.
openstack.org/api/openstack-identity/3/rel/system_group_roles', 'stable',
{'group_id':
'https://docs.openstack.org/api/openstack-identity/3/param/group_id'}),
(<class 'keystone.api.system.SystemGroupsRolestResource'>,
'/system/groups/<string:group_id>/roles/<string:role_id>', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/
system_group_role', 'stable', {'group_id':
'https://docs.openstack.org/api/openstack-identity/3/param/group_id',
'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id'}))]

resources = []
```

class keystone.api.system.**SystemGroupsRolesListResource**

Bases: Resource

get(*group_id*)

List all system grants for a specific group.

GET/HEAD /system/groups/{group_id}/roles

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.system.**SystemGroupsRolestResource**

Bases: Resource

delete(*group_id*, *role_id*)

Revoke a role from the group on the system.

DELETE /system/groups/{group_id}/roles/{role_id}

get(*group_id*, *role_id*)

Check if a group has a specific role on the system.

GET/HEAD /system/groups/{group_id}/roles/{role_id}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*group_id*, *role_id*)

Grant a role to a group on the system.

PUT /system/groups/{group_id}/roles/{role_id}

class keystone.api.system.**SystemUsersListResource**

Bases: `Resource`

get(*user_id*)

List all system grants for a specific user.

GET/HEAD /system/users/{user_id}/roles

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.system.**SystemUsersResource**

Bases: `Resource`

delete(*user_id*, *role_id*)

Revoke a role from user on the system.

DELETE /system/users/{user_id}/roles/{role_id}

get(*user_id*, *role_id*)

Check if a user has a specific role on the system.

GET/HEAD /system/users/{user_id}/roles/{role_id}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PUT'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

put(*user_id*, *role_id*)

Grant a role to a user on the system.

PUT /system/users/{user_id}/roles/{role_id}

keystone.api.trusts module

class keystone.api.trusts.**RoleForTrustResource**

Bases: `Resource`

get(*trust_id*, *role_id*)

Get a role that has been assigned to a trust.

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

property oslo_context

class keystone.api.trusts.**RolesForTrustListResource**

Bases: Resource

get(*trust_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

property oslo_context

class keystone.api.trusts.**TrustAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

resource_mapping = [(<class 'keystone.api.trusts.RolesForTrustListResource'>, '/trusts/<string:trust_id>/roles', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-TRUST/1.0/rel/trust_roles', 'stable', {'trust_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-TRUST/1.0/param/trust_id'})), (<class 'keystone.api.trusts.RoleForTrustResource'>, '/trusts/<string:trust_id>/roles/<string:role_id>', None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-TRUST/1.0/rel/trust_role', 'stable', {'role_id': 'https://docs.openstack.org/api/openstack-identity/3/param/role_id', 'trust_id': 'https://docs.openstack.org/api/openstack-identity/3/ext/OS-TRUST/1.0/param/trust_id'}))]

resources = [<class 'keystone.api.trusts.TrustResource'>]

class keystone.api.trusts.**TrustResource**

Bases: *ResourceBase*

api_prefix: str = '/OS-TRUST'

collection_key: str = 'trusts'

delete(*trust_id*)

get(*trust_id=None*)

Dispatch for GET/HEAD or LIST trusts.

json_home_parameter_rel_func(*, *extension_version='1.0'*, *parameter_name*)

json_home_resource_rel_func(*, *extension_version='1.0'*, *resource_name*)

member_key: str = 'trust'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'POST'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

post()

Create a new trust.

The User creating the trust must be the trustor.

keystone.api.users module

class keystone.api.users.**OAuth1AccessTokenCRUDResource**

Bases: _OAuth1ResourceBase

delete(*user_id*, *access_token_id*)

Delete specific access token.

DELETE /v3/users/{user_id}/OS-OAUTH1/access_tokens/{access_token_id}

get(*user_id*, *access_token_id*)

Get specific access token.

GET/HEAD /v3/users/{user_id}/OS-OAUTH1/access_tokens/{access_token_id}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.users.**OAuth1AccessTokenRoleListResource**

Bases: *ResourceBase*

collection_key: str = 'roles'

get(*user_id*, *access_token_id*)

List roles for a user access token.

GET/HEAD /v3/users/{user_id}/OS-OAUTH1/access_tokens/
{access_token_id}/roles

member_key: str = 'role'

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (["GET", "HEAD", "OPTIONS"]) as route and add_url_rule by default.

class keystone.api.users.**OAuth1AccessTokenRoleResource**

Bases: *ResourceBase*

collection_key: str = 'roles'

get(*user_id*, *access_token_id*, *role_id*)

Get role for access token.

GET/HEAD /v3/users/{user_id}/OS-OAUTH1/access_tokens/
{access_token_id}/roles/{role_id}

member_key: str = 'role'

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.users.**OAuth1ListAccessTokensResource**

Bases: `_OAuth1ResourceBase`

get(*user_id*)

List OAuth1 Access Tokens for user.

GET /v3/users/{user_id}/OS-OAUTH1/access_tokens

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.users.**UserAPI**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: *APIBase*

```
resource_mapping = [(<class
'keystone.api.users.UserChangePasswordResource'>,
'/users/<string:user_id>/password', None, {}, ('https://docs.openstack.
org/api/openstack-identity/3/rel/user_change_password', 'stable',
{'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.users.UserGroupsResource'>,
'/users/<string:user_id>/groups', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/user_groups',
'stable', {'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.users.UserProjectsResource'>,
'/users/<string:user_id>/projects', None, {},
('https://docs.openstack.org/api/openstack-identity/3/rel/user_projects',
'stable', {'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.users.UserOSEC2CredentialsResourceListCreate'>,
'/users/<string:user_id>/credentials/OS-EC2', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EC2/1.0/rel/
user_credentials', 'stable', {'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.users.UserOSEC2CredentialsResourceGetDelete'>,
'/users/<string:user_id>/credentials/OS-EC2/<string:credential_id>', None,
{}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-EC2/1.0/
rel/user_credential', 'stable', {'credential_id':
'https://docs.openstack.org/api/openstack-identity/3/param/credential_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.users.OAuth1ListAccessTokensResource'>,
'/users/<string:user_id>/OS-OAUTH1/access_tokens', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/
rel/user_access_tokens', 'stable', {'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.users.OAuth1AccessTokenCRUDResource'>, '/users/
<string:user_id>/OS-OAUTH1/access_tokens/<string:access_token_id>', None,
{}, ('https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.
0/rel/user_access_token', 'stable', {'access_token_id':
'https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/
param/access_token_id', 'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.users.OAuth1AccessTokenRoleListResource'>, '/users/
<string:user_id>/OS-OAUTH1/access_tokens/<string:access_token_id>/roles',
None, {}, ('https://docs.openstack.org/api/openstack-identity/3/ext/
OS-OAUTH1/1.0/rel/user_access_token_roles', 'stable', {'access_token_id':
'https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/
param/access_token_id', 'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
(<class 'keystone.api.users.OAuth1AccessTokenRoleResource'>,
'/users/<string:user_id>/OS-OAUTH1/access_tokens/<string:access_token_id>/
roles/<string:role_id>', None, {},
('https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/
rel/user_access_token_role', 'stable', {'access_token_id':
'https://docs.openstack.org/api/openstack-identity/3/ext/OS-OAUTH1/1.0/
param/access_token_id', 'role_id':
'https://docs.openstack.org/api/openstack-identity/3/param/role_id',
'user_id':
'https://docs.openstack.org/api/openstack-identity/3/param/user_id'})),
```

resources = [<class 'keystone.api.users.UserResource'>]

class keystone.api.users.**UserAccessRuleGetDeleteResource**

Bases: *ResourceBase*

collection_key: str = 'access_rules'

delete(*user_id*, *access_rule_id*)

Delete access rule resource.

DELETE /v3/users/{user_id}/access_rules/{access_rule_id}

get(*user_id*, *access_rule_id*)

Get access rule resource.

GET/HEAD /v3/users/{user_id}/access_rules/{access_rule_id}

member_key: str = 'access_rule'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.users.**UserAccessRuleListResource**

Bases: *ResourceBase*

collection_key: str = 'access_rules'

get(*user_id*)

List access rules for user.

GET/HEAD /v3/users/{user_id}/access_rules

member_key: str = 'access_rule'

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.users.**UserAppCredGetDeleteResource**

Bases: *ResourceBase*

collection_key: str = 'application_credentials'

delete(*user_id*, *application_credential_id*)

Delete application credential resource.

DELETE /v3/users/{user_id}/application_credentials/
{application_credential_id}

get(*user_id*, *application_credential_id*)

Get application credential resource.

GET/HEAD /v3/users/{user_id}/application_credentials/
{application_credential_id}

member_key: str = 'application_credential'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.users.**UserAppCredListCreateResource**

Bases: *ResourceBase*

collection_key: str = 'application_credentials'

get(*user_id*)

List application credentials for user.

GET/HEAD /v3/users/{user_id}/application_credentials

member_key: str = 'application_credential'

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

post(*user_id*)

Create application credential.

POST /v3/users/{user_id}/application_credentials

class keystone.api.users.**UserChangePasswordResource**

Bases: *ResourceBase*

get(*user_id*)

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

post(*user_id*)

class keystone.api.users.**UserGroupsResource**

Bases: *ResourceBase*

collection_key: str = 'groups'

get(*user_id*)

Get groups for a user.

GET/HEAD /v3/users/{user_id}/groups

get_member_from_driver

member_key: str = 'group'

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.users.**UserOSEC2CredentialsResourceGetDelete**

Bases: `_UserOSEC2CredBaseResource`

delete(*user_id*, *credential_id*)

Delete a specific EC2 credential.

DELETE /users/{user_id}/credentials/OS-EC2/{credential_id}

get(*user_id*, *credential_id*)

Get a specific EC2 credential.

GET/HEAD /users/{user_id}/credentials/OS-EC2/{credential_id}

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.users.**UserOSEC2CredentialsResourceListCreate**

Bases: `_UserOSEC2CredBaseResource`

get(*user_id*)

List EC2 Credentials for user.

GET/HEAD /v3/users/{user_id}/credentials/OS-EC2

methods: t.ClassVar[t.Collection[str] | None] = {'GET', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

post(*user_id*)

Create EC2 Credential for user.

POST /v3/users/{user_id}/credentials/OS-EC2

class keystone.api.users.**UserProjectsResource**

Bases: *ResourceBase*

collection_key: str = 'projects'

get(*user_id*)

get_member_from_driver

member_key: str = 'project'

methods: t.ClassVar[t.Collection[str] | None] = {'GET'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

class keystone.api.users.**UserResource**

Bases: *ResourceBase*

collection_key: str = 'users'

delete(*user_id*)

Delete a user.

DELETE /v3/users/{user_id}

get(*user_id=None*)

Get a user resource or list users.

GET/HEAD /v3/users GET/HEAD /v3/users/{user_id}

get_member_from_driver

member_key: str = 'user'

methods: t.ClassVar[t.Collection[str] | None] = {'DELETE', 'GET', 'PATCH', 'POST'}

The methods this view is registered for. Uses the same default (`["GET", "HEAD", "OPTIONS"]`) as `route` and `add_url_rule` by default.

patch(*user_id*)

Update a user.

PATCH /v3/users/{user_id}

post()

Create a user.

POST /v3/users

Module contents

keystone.application_credential package

Subpackages

keystone.application_credential.backends package

Submodules

keystone.application_credential.backends.base module

class keystone.application_credential.backends.base.**ApplicationCredentialDriverBase**

Bases: `object`

abstract authenticate(*application_credential_id*, *secret*)

Validate an application credential.

Parameters

- **application_credential_id** (*str*) Application Credential ID
- **secret** (*str*) Secret

Raises

AssertionError If id or secret is invalid.

abstract create_application_credential(*application_credential*, *roles*)

Create a new application credential.

Parameters

- **application_credential** (*dict*) Application Credential data
- **roles** (*list*) A list of roles that apply to the application_credential.

Returns
a new application credential

abstract delete_access_rule(*access_rule_id*)

Delete one access rule.

Parameters
access_rule_id (*str*) Access Rule ID

abstract delete_access_rules_for_user(*user_id*)

Delete all access rules for user.

This is called when the user itself is deleted.

Parameters
user_id (*str*) User ID

abstract delete_application_credential(*application_credential_id*)

Delete a single application credential.

Parameters
application_credential_id (*str*) ID of the application credential to delete.

abstract delete_application_credentials_for_user(*user_id*)

Delete all application credentials for a user.

Parameters
user_id ID of a user to whose application credentials should be deleted.

abstract delete_application_credentials_for_user_on_project(*user_id*, *project_id*)

Delete all application credentials for a user on a given project.

Parameters

- **user_id** (*str*) ID of a user to whose application credentials should be deleted.
- **project_id** (*str*) ID of a project on which to filter application credentials.

abstract get_access_rule(*access_rule_id*)

Get an access rule by its ID.

Parameters
access_rule_id (*str*) Access Rule ID

abstract get_application_credential(*application_credential_id*)

Get an application credential by the credential id.

Parameters
application_credential_id (*str*) Application Credential ID

abstract list_access_rules_for_user(*user_id*)

List the access rules that a user has created.

Access rules are only created as attributes of application credentials, they cannot be created independently.

Parameters
user_id (*str*) User ID

abstract list_application_credentials_for_user(*user_id*, *hints*)

List application credentials for a user.

Parameters

- **user_id** (*str*) User ID
- **hints** contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

keystone.application_credential.backends.sql module

class keystone.application_credential.backends.sql.**AccessRuleModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

application_credential

attributes = ['external_id', 'user_id', 'service', 'path', 'method']

external_id

id

method

path

service

user_id

class keystone.application_credential.backends.sql.**ApplicationCredential**

Bases: *ApplicationCredentialDriverBase*

authenticate(*application_credential_id*, *secret*)

Validate an application credential.

Parameters

- **application_credential_id** (*str*) Application Credential ID
- **secret** (*str*) Secret

Raises
AssertionError If id or secret is invalid.

create_application_credential(*application_credential*, *roles*, *access_rules=None*)

Create a new application credential.

Parameters

- **application_credential** (*dict*) Application Credential data
- **roles** (*list*) A list of roles that apply to the application_credential.

Returns

a new application credential

delete_access_rule(*access_rule_id*)

Delete one access rule.

Parameters

access_rule_id (*str*) Access Rule ID

delete_access_rules_for_user(*user_id*)

Delete all access rules for user.

This is called when the user itself is deleted.

Parameters

user_id (*str*) User ID

delete_application_credential(*application_credential_id*)

Delete a single application credential.

Parameters

application_credential_id (*str*) ID of the application credential to delete.

delete_application_credentials_for_user(*user_id*)

Delete all application credentials for a user.

Parameters

user_id ID of a user to whose application credentials should be deleted.

delete_application_credentials_for_user_on_project(*user_id*, *project_id*)

Delete all application credentials for a user on a given project.

Parameters

- **user_id** (*str*) ID of a user to whose application credentials should be deleted.
- **project_id** (*str*) ID of a project on which to filter application credentials.

get_access_rule(*access_rule_id*)

Get an access rule by its ID.

Parameters

access_rule_id (*str*) Access Rule ID

get_application_credential(*application_credential_id*)

Get an application credential by the credential id.

Parameters

application_credential_id (*str*) Application Credential ID

list_access_rules_for_user(*user_id*, *hints*)

List the access rules that a user has created.

Access rules are only created as attributes of application credentials, they cannot be created independently.

Parameters
user_id (*str*) User ID

list_application_credentials_for_user(*user_id*, *hints*)

List application credentials for a user.

Parameters

- **user_id** (*str*) User ID
- **hints** contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

class keystone.application_credential.backends.sql.ApplicationCredentialAccessRuleModel(**arg* **kv*

Bases: Base, ModelDictMixin

access_rule_id

application_credential_id

attributes = ['application_credential_id', 'access_rule_id']

class keystone.application_credential.backends.sql.ApplicationCredentialModel(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

access_rules

attributes = ['internal_id', 'id', 'name', 'secret_hash', 'description', 'user_id', 'project_id', 'system', 'expires_at', 'unrestricted']

description

expires_at

id

internal_id

name

project_id

roles

secret_hash

system

unrestricted

user_id

class keystone.application_credential.backends.sql.**ApplicationCredentialRoleModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

application_credential_id

attributes = ['application_credential_id', 'role_id']

role_id

Module contents

Submodules

keystone.application_credential.core module

Main entry point into the Application Credential service.

class keystone.application_credential.core.**Manager**

Bases: *Manager*

Default pivot point for the Application Credential backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

authenticate(*application_credential_id*, *secret*)

Authenticate with an application credential.

Parameters

- **application_credential_id** (*str*) Application Credential ID
- **secret** (*str*) Application Credential secret

create_application_credential(*application_credential*, *initiator=None*)

Create a new application credential.

Parameters

- **application_credential** (*dict*) Application Credential data
- **initiator** CADF initiator

Returns

a new application credential

delete_access_rule(*access_rule_id*, *initiator=None*)

Delete an access rule.

Parameters

- **str** access_rule_id: Access Rule ID
- **initiator** CADF initiator

Raises
keystone.exception.AccessRuleNotFound If the access rule doesnt exist.

`delete_application_credential`(*application_credential_id*, *initiator=None*)

Delete an application credential.

Parameters

- **application_credential_id** (*str*) Application Credential ID
- **initiator** CADF initiator

Raises
keystone.exception.ApplicationCredentialNotFound If the application credential doesnt exist.

`driver_namespace: str = 'keystone.application_credential'`

`get_access_rule`(*access_rule_id*)

Get access rule details.

Parameters
access_rule_id (*str*) Access Rule ID

Returns
an access rule

`get_application_credential`(*application_credential_id*)

Get application credential details.

Parameters
application_credential_id (*str*) Application Credential ID

Returns
an application credential

`list_access_rules_for_user`(*user_id*, *hints=None*)

List access rules for user.

Parameters
user_id (*str*) User ID

Returns
a list of access rules

`list_application_credentials`(*user_id*, *hints=None*)

List application credentials for a user.

Parameters

- **user_id** (*str*) User ID
- **hints** (*dict*) Properties to filter on

Returns
a list of application credentials

keystone.application_credential.schema module

Module contents

keystone.assignment package

Subpackages

keystone.assignment.backends package

Submodules

keystone.assignment.backends.base module

class keystone.assignment.backends.base.**AssignmentDriverBase**

Bases: object

abstract add_role_to_user_and_project(*user_id*, *project_id*, *role_id*)

Add a role to a user within given project.

Raises

keystone.exception.Conflict If a duplicate role assignment exists.

abstract check_grant_role_id(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

Check an assignment/grant role id.

Raises

keystone.exception.RoleAssignmentNotFound If the role assignment doesnt exist.

Returns

None or raises an exception if grant not found

abstract check_system_grant(*role_id*, *actor_id*, *target_id*, *inherited*)

Check if a user or group has a specific role on the system.

Parameters

- **role_id** the unique ID of the role to grant to the user
- **actor_id** the unique ID of the user or group
- **target_id** the unique ID or string representing the target
- **inherited** a boolean denoting if the assignment is inherited or not

abstract create_grant(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

Create a new assignment/grant.

If the assignment is to a domain, then optionally it may be specified as inherited to owned projects (this requires the OS-INHERIT extension to be enabled).

abstract create_system_grant(*role_id*, *actor_id*, *target_id*, *assignment_type*, *inherited*)

Grant a user or group a role on the system.

Parameters

- **role_id** the unique ID of the role to grant to the user
- **actor_id** the unique ID of the user or group
- **target_id** the unique ID or string representing the target
- **assignment_type** a string describing the relationship of the assignment
- **inherited** a boolean denoting if the assignment is inherited or not

abstract delete_domain_assignments(*domain_id*)

Delete all assignments for a domain.

abstract delete_grant(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

Delete assignments/grants.

Raises

keystone.exception.RoleAssignmentNotFound If the role assignment doesnt exist.

abstract delete_group_assignments(*group_id*)

Delete all assignments for a group.

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

abstract delete_project_assignments(*project_id*)

Delete all assignments for a project.

Raises

keystone.exception.ProjectNotFound If the project doesnt exist.

abstract delete_role_assignments(*role_id*)

Delete all assignments for a role.

abstract delete_system_grant(*role_id*, *actor_id*, *target_id*, *inherited*)

Remove a system assignment from a user or group.

Parameters

- **role_id** the unique ID of the role to grant to the user
- **actor_id** the unique ID of the user or group
- **target_id** the unique ID or string representing the target
- **inherited** a boolean denoting if the assignment is inherited or not

abstract delete_user_assignments(*user_id*)

Delete all assignments for a user.

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

abstract list_grant_role_ids(*user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

List role ids for assignments/grants.

abstract list_role_assignments(*role_id=None*, *user_id=None*, *group_ids=None*, *domain_id=None*, *project_ids=None*, *inherited_to_projects=None*)

Return a list of role assignments for actors on targets.

Available parameters represent values in which the returned role assignments attributes need to be filtered on.

abstract list_system_grants(*actor_id*, *target_id*, *assignment_type*)

Return a list of all system assignments for a specific entity.

Parameters

- **actor_id** the unique ID of the actor
- **target_id** the unique ID of the target
- **assignment_type** the type of assignment to return

abstract list_system_grants_by_role(*role_id*)

Return a list of system assignments associated to a role.

Parameters

role_id the unique ID of the role to grant to the user

abstract remove_role_from_user_and_project(*user_id*, *project_id*, *role_id*)

Remove a role from a user within given project.

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

keystone.assignment.backends.sql module

class keystone.assignment.backends.sql.**Assignment**

Bases: *AssignmentDriverBase*

add_role_to_user_and_project(*user_id*, *project_id*, *role_id*)

Add a role to a user within given project.

Raises

keystone.exception.Conflict If a duplicate role assignment exists.

check_grant_role_id(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

Check an assignment/grant role id.

Raises

keystone.exception.RoleAssignmentNotFound If the role assignment doesnt exist.

Returns

None or raises an exception if grant not found

check_system_grant(*role_id*, *actor_id*, *target_id*, *inherited*)

Check if a user or group has a specific role on the system.

Parameters

- **role_id** the unique ID of the role to grant to the user
- **actor_id** the unique ID of the user or group
- **target_id** the unique ID or string representing the target
- **inherited** a boolean denoting if the assignment is inherited or not

create_grant(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

Create a new assignment/grant.

If the assignment is to a domain, then optionally it may be specified as inherited to owned projects (this requires the OS-INHERIT extension to be enabled).

create_system_grant(*role_id*, *actor_id*, *target_id*, *assignment_type*, *inherited*)

Grant a user or group a role on the system.

Parameters

- **role_id** the unique ID of the role to grant to the user
- **actor_id** the unique ID of the user or group
- **target_id** the unique ID or string representing the target
- **assignment_type** a string describing the relationship of the assignment
- **inherited** a boolean denoting if the assignment is inherited or not

classmethod default_role_driver()

delete_domain_assignments(*domain_id*)

Delete all assignments for a domain.

delete_grant(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

Delete assignments/grants.

Raises

keystone.exception.RoleAssignmentNotFound If the role assignment doesnt exist.

delete_group_assignments(*group_id*)

Delete all assignments for a group.

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

delete_project_assignments(*project_id*)

Delete all assignments for a project.

Raises

keystone.exception.ProjectNotFound If the project doesnt exist.

delete_role_assignments(*role_id*)

Delete all assignments for a role.

delete_system_grant(*role_id*, *actor_id*, *target_id*, *inherited*)

Remove a system assignment from a user or group.

Parameters

- **role_id** the unique ID of the role to grant to the user
- **actor_id** the unique ID of the user or group
- **target_id** the unique ID or string representing the target
- **inherited** a boolean denoting if the assignment is inherited or not

delete_user_assignments(*user_id*)

Delete all assignments for a user.

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

list_grant_role_ids(*user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

List role ids for assignments/grants.

list_role_assignments(*role_id=None*, *user_id=None*, *group_ids=None*, *domain_id=None*, *project_ids=None*, *inherited_to_projects=None*)

Return a list of role assignments for actors on targets.

Available parameters represent values in which the returned role assignments attributes need to be filtered on.

list_system_grants(*actor_id*, *target_id*, *assignment_type*)

Return a list of all system assignments for a specific entity.

Parameters

- **actor_id** the unique ID of the actor
- **target_id** the unique ID of the target
- **assignment_type** the type of assignment to return

list_system_grants_by_role(*role_id*)

Return a list of system assignments associated to a role.

Parameters

role_id the unique ID of the role to grant to the user

remove_role_from_user_and_project(*user_id*, *project_id*, *role_id*)

Remove a role from a user within given project.

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

class keystone.assignment.backends.sql.**AssignmentType**

Bases: object

GROUP_DOMAIN = 'GroupDomain'

GROUP_PROJECT = 'GroupProject'

USER_DOMAIN = 'UserDomain'

USER_PROJECT = 'UserProject'

classmethod calculate_type(*user_id*, *group_id*, *project_id*, *domain_id*)

class keystone.assignment.backends.sql.RoleAssignment(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

actor_id

attributes = ['type', 'actor_id', 'target_id', 'role_id', 'inherited']

inherited

role_id

target_id

to_dict()

Override parent method with a simpler implementation.

RoleAssignment doesnt have non-indexed extra attributes, so the parent implementation is not applicable.

type

class keystone.assignment.backends.sql.SystemRoleAssignment(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

actor_id

attributes = ['type', 'actor_id', 'target_id', 'role_id', 'inherited']

inherited

role_id

target_id

to_dict()

Override parent method with a simpler implementation.

RoleAssignment doesnt have non-indexed extra attributes, so the parent implementation is not applicable.

type

Module contents

keystone.assignment.role_backends package

Submodules

keystone.assignment.role_backends.base module

class keystone.assignment.role_backends.base.**RoleDriverBase**

Bases: object

abstract create_implied_role(*prior_role_id*, *implied_role_id*)

Create a role inference rule.

Raises

keystone.exception.RoleNotFound: If the role doesnt exist.

abstract create_role(*role_id*, *role*)

Create a new role.

Raises

keystone.exception.Conflict If a duplicate role exists.

abstract delete_implied_role(*prior_role_id*, *implied_role_id*)

Delete a role inference rule.

Raises

keystone.exception.ImpliedRoleNotFound If the implied role doesnt exist.

abstract delete_role(*role_id*)

Delete an existing role.

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

abstract get_implied_role(*prior_role_id*, *implied_role_id*)

Get a role inference rule.

Raises

keystone.exception.ImpliedRoleNotFound If the implied role doesnt exist.

abstract get_role(*role_id*)

Get a role by ID.

Returns

role_ref

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

abstract list_implied_roles(*prior_role_id*)

List roles implied from the prior role ID.

abstract list_role_inference_rules()

List all the rules used to imply one role from another.

abstract list_roles(*hints*)

List roles in the system.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
a list of role_refs or an empty list.

abstract list_roles_from_ids(*role_ids*)

List roles for the provided list of ids.

Parameters
role_ids list of ids

Returns
a list of role_refs.

This method is used internally by the assignment manager to bulk read a set of roles given their ids.

abstract update_role(*role_id*, *role*)

Update an existing role.

Raises

- *keystone.exception.RoleNotFound* If the role doesnt exist.
- *keystone.exception.Conflict* If a duplicate role exists.

keystone.assignment.role_backends.resource_options module

keystone.assignment.role_backends.resource_options.**register_role_options**()

keystone.assignment.role_backends.sql module

class keystone.assignment.role_backends.sql.**Role**

Bases: *RoleDriverBase*

create_implied_role(*prior_role_id*, *implied_role_id*)

Create a role inference rule.

Raises
keystone.exception.RoleNotFound: If the role doesnt exist.

create_role(*role_id*, *role*)

Create a new role.

Raises
keystone.exception.Conflict If a duplicate role exists.

delete_implied_role(*prior_role_id*, *implied_role_id*)

Delete a role inference rule.

Raises

keystone.exception.ImpliedRoleNotFound If the implied role doesnt exist.

delete_role(*role_id*)

Delete an existing role.

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

get_implied_role(*prior_role_id*, *implied_role_id*)

Get a role inference rule.

Raises

keystone.exception.ImpliedRoleNotFound If the implied role doesnt exist.

get_role(*role_id*)

Get a role by ID.

Returns

role_ref

Raises

keystone.exception.RoleNotFound If the role doesnt exist.

list_implied_roles(*prior_role_id*)

List roles implied from the prior role ID.

list_role_inference_rules()

List all the rules used to imply one role from another.

list_roles(*hints*)

List roles in the system.

Parameters

hints filter hints which the driver should implement if at all possible.

Returns

a list of role_refs or an empty list.

list_roles_from_ids(*ids*)

List roles for the provided list of ids.

Parameters

role_ids list of ids

Returns

a list of role_refs.

This method is used internally by the assignment manager to bulk read a set of roles given their ids.

update_role(*role_id*, *role*)

Update an existing role.

Raises

- *keystone.exception.RoleNotFound* If the role doesnt exist.
- *keystone.exception.Conflict* If a duplicate role exists.

keystone.assignment.role_backends.sql_model module

class keystone.assignment.role_backends.sql_model.**ImpliedRoleTable**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['prior_role_id', 'implied_role_id']

classmethod from_dict(*dictionary*)

Return a model instance from a dictionary.

implied_role_id

prior_role_id

to_dict()

Return a dictionary with models attributes.

overrides the *to_dict* function from the base class to avoid having an *extra* field.

class keystone.assignment.role_backends.sql_model.**RoleOption**(*option_id*, *option_value*)

Bases: Base

option_id

option_value

role_id

class keystone.assignment.role_backends.sql_model.**RoleTable**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'name', 'domain_id', 'description']

description

domain_id

extra

classmethod from_dict(*role_dict*)

id

name

resource_options_registry = <keystone.common.resource_options.core.ResourceOptionRegistry object>

to_dict(*include_extra_dict=False*)

Return the models attributes as a dictionary.

If include_extra_dict is True, extra attributes are literally included in the resulting dictionary twice, for backwards-compatibility with a broken implementation.

Module contents

Submodules

keystone.assignment.core module

Main entry point into the Assignment service.

class keystone.assignment.core.**Manager**

Bases: *Manager*

Default pivot point for the Assignment backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

add_implied_roles(*role_refs*)

Expand out implied roles.

The role_refs passed in have had all inheritance and group assignments expanded out. We now need to look at the role_id in each ref and see if it is a prior role for some implied roles. If it is, then we need to duplicate that ref, one for each implied role. We store the prior role in the indirect dict that is part of such a duplicated ref, so that a caller can determine where the assignment came from.

add_role_to_user_and_project(*user_id*, *project_id*, *role_id*)

check_system_grant_for_group(*group_id*, *role_id*)

Check if a group has a specific role on the system.

Parameters

- **group_id** the ID of the group in the assignment
- **role_id** the ID of the system role in the assignment

Raises

keystone.exception.RoleAssignmentNotFound if the group doesnt have a role assignment matching the role_id on the system

check_system_grant_for_user(*user_id*, *role_id*)

Check if a user has a specific role on the system.

Parameters

- **user_id** the ID of the user in the assignment
- **role_id** the ID of the system role in the assignment

Raises

keystone.exception.RoleAssignmentNotFound if the user doesnt have a role assignment matching the role_id on the system

create_grant(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*, *initiator=None*)

create_system_grant_for_group(*group_id*, *role_id*)

Grant a group a role on the system.

Parameters

- **group_id** the ID of the group
- **role_id** the ID of the role to grant on the system

create_system_grant_for_user(*user_id*, *role_id*)

Grant a user a role on the system.

Parameters

- **user_id** the ID of the user
- **role_id** the ID of the role to grant on the system

delete_grant(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*, *initiator=None*)

delete_group_assignments(*group_id*)

delete_system_grant_for_group(*group_id*, *role_id*)

Remove a system grant from a group.

Parameters

- **group_id** the ID of the group
- **role_id** the ID of the role to remove from the group on the system

Raises

keystone.exception.RoleAssignmentNotFound if the group doesnt have a role assignment with role_id on the system

delete_system_grant_for_user(*user_id*, *role_id*)

Remove a system grant from a user.

Parameters

- **user_id** the ID of the user
- **role_id** the ID of the role to remove from the user on the system

Raises

keystone.exception.RoleAssignmentNotFound if the user doesnt have a role assignment with role_id on the system

delete_user_assignments(*user_id*)

driver_namespace: str = 'keystone.assignment'

get_grant(*role_id*, *user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

get_roles_for_groups(*group_ids*, *project_id=None*, *domain_id=None*)

Get a list of roles for this group on domain and/or project.

get_roles_for_trustor_and_project(*trustor_id*, *project_id*)

Get the roles associated with a trustor within given project.

This includes roles directly assigned to the trustor on the project, as well as those by virtue of group membership or inheritance, but it doesnt include the domain roles.

Returns
a list of role ids.

Raises
keystone.exception.ProjectNotFound If the project doesnt exist.

get_roles_for_user_and_domain(*user_id*, *domain_id*)

Get the roles associated with a user within given domain.

Returns
a list of role ids.

Raises
keystone.exception.DomainNotFound If the domain doesnt exist.

get_roles_for_user_and_project(*user_id*, *project_id*)

Get the roles associated with a user within given project.

This includes roles directly assigned to the user on the project, as well as those by virtue of group membership or inheritance.

Returns
a list of role ids.

Raises
keystone.exception.ProjectNotFound If the project doesnt exist.

list_all_system_grants()

Return a list of all system grants.

list_domains_for_groups(*group_ids*)

list_domains_for_user(*user_id*)

list_grants(*user_id=None*, *group_id=None*, *domain_id=None*, *project_id=None*, *inherited_to_projects=False*)

list_projects_for_groups(*group_ids*)

list_projects_for_user(*user_id*)

list_role_assignments(*role_id=None*, *user_id=None*, *group_id=None*, *system=None*, *domain_id=None*, *project_id=None*, *include_subtree=False*, *inherited=None*, *effective=None*, *include_names=False*, *source_from_group_ids=None*, *strip_domain_roles=True*)

List role assignments, honoring effective mode and provided filters.

Returns a list of role assignments, where their attributes match the provided filters (role_id, user_id, group_id, domain_id, project_id and inherited). If include_subtree is True, then assignments on all descendants of the project specified by project_id are also included. The inherited filter defaults to None, meaning to get both non-inherited and inherited role assignments.

If effective mode is specified, this means that rather than simply return the assignments that match the filters, any group or inheritance assignments will be expanded. Group assignments will become assignments for all the users in that group, and inherited assignments will be shown on the projects below the assignment point. Think of effective mode as being the list of assignments that actually affect a user, for example the roles that would be placed in a token.

If include_names is set to true the entities names are returned in addition to their ids.

source_from_group_ids is a list of group IDs and, if specified, then only those assignments that are derived from membership of these groups are considered, and any such assignments will not be expanded into their user membership assignments. This is different to a group filter of the resulting list, instead being a restriction on which assignments should be considered before expansion of inheritance. This option is only used internally (i.e. it is not exposed at the API level) and is only supported in effective mode (since in regular mode there is no difference between this and a group filter, other than it is a list of groups).

In effective mode, any domain specific roles are usually stripped from the returned assignments (since such roles are not placed in tokens). This stripping can be disabled by specifying strip_domain_roles=False, which is useful for internal calls like trusts which need to examine the full set of roles.

list_system_grants_for_group(*group_id*)

Return a list of roles the group has on the system.

Parameters
group_id the ID of the group

Returns
a list of role assignments the group has system-wide

list_system_grants_for_user(*user_id*)

Return a list of roles the user has on the system.

Parameters
user_id the ID of the user

Returns
a list of role assignments the user has system-wide

list_user_ids_for_project(*project_id*)

remove_role_from_user_and_project(*user_id*, *project_id*, *role_id*)

class keystone.assignment.core.**RoleManager**

Bases: *Manager*

Default pivot point for the Role backend.

create_implied_role(*prior_role_id*, *implied_role_id*)

create_role(*role_id*, *role*, *initiator=None*)

delete_implied_role(*prior_role_id*, *implied_role_id*)

delete_role(*role_id*, *initiator=None*)

driver_namespace: str = 'keystone.role'

get_role(*role_id*)

get_unique_role_by_name(*role_name*, *hints=None*)

list_roles(*hints=None*)

update_role(*role_id*, *role*, *initiator=None*)

keystone.assignment.schema module

Module contents

keystone.auth package

Subpackages

keystone.auth.plugins package

Submodules

keystone.auth.plugins.application_credential module

class keystone.auth.plugins.application_credential.**ApplicationCredential**

Bases: *AuthMethodHandler*

authenticate(*auth_payload*)

Authenticate an application.

keystone.auth.plugins.base module

class keystone.auth.plugins.base.**AuthHandlerResponse**(*status*, *response_body*, *response_data*)

Bases: tuple

response_body

Alias for field number 1

response_data

Alias for field number 2

status

Alias for field number 0

class keystone.auth.plugins.base.**AuthMethodHandler**

Bases: *ProviderAPIMixin*

Abstract base class for an authentication plugin.

abstract authenticate(*auth_payload*)

Authenticate user and return an authentication context.

Parameters

auth_payload (*dict*) the payload content of the authentication request for a given method

If successful, plugin must set `user_id` in `response_data`. `method_name` is used to convey any additional authentication methods in case authentication is for re-scoping. For example, if the authentication is for re-scoping, plugin must append the previous method names into `method_names`; NOTE: This behavior is exclusive to the re-scope type action. Heres an example of `response_data` on successful authentication:

```
{
    "methods": [
        "password",
        "token"
    ],
    "user_id": "abc123"
}
```

Plugins are invoked in the order in which they are specified in the `methods` attribute of the `identity` object. For example, `custom-plugin` is invoked before `password`, which is invoked before `token` in the following authentication request:

```
{
    "auth": {
        "identity": {
            "custom-plugin": {
                "custom-data": "sdfdfsfsfsdfsf"
            },
            "methods": [
                "custom-plugin",
                "password",
                "token"
            ],
            "password": {
                "user": {
                    "id": "s23sfad1",
                    "password": "secret"
                }
            },
            "token": {
                "id": "sdfafasdfsfasfasdfds"
            }
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

Returns
AuthHandlerResponse with status set to `True` if auth was successful. If *status* is `False` and this is a multi-step auth, the `response_body` can be in a form of a dict for the next step in authentication.

Raises
keystone.exception.Unauthorized for authentication failure

keystone.auth.plugins.core module

class keystone.auth.plugins.core.**AppCredInfo**

Bases: *BaseUserInfo*

class keystone.auth.plugins.core.**BaseUserInfo**

Bases: *ProviderAPIMixin*

classmethod create(*auth_payload*, *method_name*)

class keystone.auth.plugins.core.**TOTPUserInfo**

Bases: *BaseUserInfo*

class keystone.auth.plugins.core.**UserAuthInfo**

Bases: *BaseUserInfo*

keystone.auth.plugins.core.**construct_method_map_from_config**()

Determine authentication method types for deployment.

Returns
a dictionary containing the methods and their indexes

keystone.auth.plugins.core.**convert_integer_to_method_list**(*method_int*)

Convert an integer to a list of methods.

Parameters
method_int an integer representing methods

Returns
a corresponding list of methods

keystone.auth.plugins.core.**convert_method_list_to_integer**(*methods*)

Convert the method type(s) to an integer.

Parameters
methods a list of method names

Returns
an integer representing the methods

keystone.auth.plugins.external module

Keystone External Authentication Plugins.

class keystone.auth.plugins.external.**Base**

Bases: *AuthMethodHandler*

authenticate(*auth_payload*)

Use REMOTE_USER to look up the user in the identity backend.

The user_id from the actual user from the REMOTE_USER env variable is placed in the response_data.

class keystone.auth.plugins.external.**DefaultDomain**

Bases: *Base*

class keystone.auth.plugins.external.**Domain**

Bases: *Base*

class keystone.auth.plugins.external.**KerberosDomain**

Bases: *Domain*

Allows *kerberos* as a method.

keystone.auth.plugins.mapped module

class keystone.auth.plugins.mapped.**Mapped**

Bases: *AuthMethodHandler*

authenticate(*auth_payload*)

Authenticate mapped user and set an authentication context.

Parameters

auth_payload the content of the authentication for a given method

In addition to `user_id` in `response_data`, this plugin sets `group_ids`, `OS-FEDERATION:identity_provider` and `OS-FEDERATION:protocol`

keystone.auth.plugins.mapped.**apply_mapping_filter**(*identity_provider*, *protocol*, *assertion*, *resource_api*, *federation_api*, *identity_api*)

keystone.auth.plugins.mapped.**configure_project_domain**(*shadow_project*, *idp_domain_id*, *resource_api*)

Configure federated projects domain.

We set the domain to be the default (idp_domain_id) if the project from the attribute mapping comes without a domain.

keystone.auth.plugins.mapped.**extract_assertion_data**()

keystone.auth.plugins.mapped.**handle_projects_from_mapping**(*shadow_projects*, *idp_domain_id*, *existing_roles*, *user*, *assignment_api*, *resource_api*)

keystone.auth.plugins.mapped.**handle_scoped_token**(*token*, *federation_api*, *identity_api*)

keystone.auth.plugins.mapped.**handle_unscoped_token**(*auth_payload*, *resource_api*, *federation_api*, *identity_api*, *assignment_api*, *role_api*)

keystone.auth.plugins.mapped.**validate_and_prepare_federated_user**(*mapped_properties*, *idp_domain_id*, *resource_api*)

Setup federated username.

Function covers all the cases for properly setting user id, a primary identifier for identity objects. Initial version of the mapping engine assumed user is identified by `name` and his `id` is built from the name. We, however need to be able to accept local rules that identify user by either id or name/domain.

The following use-cases are covered:

1) If neither user_name nor user_id is set raise exception.Unauthorized

2) If user_id is set and user_name not, set user_name equal to user_id

3) If user_id is not set and user_name is, set user_id as url safe version of user_name.

Furthermore, we set the IdP as the user domain, if the user definition does not come with a domain definition.

Parameters

- **mapped_properties** Properties issued by a RuleProcessor.
- **idp_domain_id** The domain ID of the IdP registered in OpenStack.
- **resource_api** The resource API used to access the database layer.

Type
dictionary

Type
string

Type
object

Raises
keystone.exception.Unauthorized If neither *user_name* nor *user_id* is set.

Returns
tuple with user identification

Return type
tuple

keystone.auth.plugins.oauth1 module

class keystone.auth.plugins.oauth1.**OAuth**

Bases: *AuthMethodHandler*

authenticate(*auth_payload*)

Turn a signed request with an access key into a keystone token.

keystone.auth.plugins.password module

class keystone.auth.plugins.password.**Password**

Bases: *AuthMethodHandler*

authenticate(*auth_payload*)

Try to authenticate against the identity backend.

keystone.auth.plugins.token module

class keystone.auth.plugins.token.**Token**

Bases: *AuthMethodHandler*

authenticate(*auth_payload*)

Authenticate user and return an authentication context.

Parameters

auth_payload (*dict*) the payload content of the authentication request for a given method

If successful, plugin must set `user_id` in `response_data`. `method_name` is used to convey any additional authentication methods in case authentication is for re-scoping. For example, if the authentication is for re-scoping, plugin must append the previous method names into `method_names`; NOTE: This behavior is exclusive to the re-scope type action. Heres an example of `response_data` on successful authentication:

```
{
    "methods": [
        "password",
        "token"
    ],
    "user_id": "abc123"
}
```

Plugins are invoked in the order in which they are specified in the `methods` attribute of the `identity` object. For example, `custom-plugin` is invoked before `password`, which is invoked before `token` in the following authentication request:

```
{
    "auth": {
        "identity": {
            "custom-plugin": {
```

(continues on next page)

(continued from previous page)

```
                "custom-data": "sdfdfsfsfsdfsf"
            },
            "methods": [
                "custom-plugin",
                "password",
                "token"
            ],
            "password": {
                "user": {
                    "id": "s23sfad1",
                    "password": "secret"
                }
            },
            "token": {
                "id": "sdfafasdfsfasfasdfds"
            }
        }
    }
}
```

Returns

AuthHandlerResponse with status set to `True` if auth was successful. If *status* is `False` and this is a multi-step auth, the `response_body` can be in a form of a dict for the next step in authentication.

Raises

keystone.exception.Unauthorized for authentication failure

keystone.auth.plugins.token.**token_authenticate**(*token*)

keystone.auth.plugins.totp module

Time-based One-time Password Algorithm (TOTP) auth plugin.

TOTP is an algorithm that computes a one-time password from a shared secret key and the current time.

TOTP is an implementation of a hash-based message authentication code (HMAC). It combines a secret key with the current timestamp using a cryptographic hash function to generate a one-time password. The timestamp typically increases in 30-second intervals, so passwords generated close together in time from the same secret key will be equal.

class keystone.auth.plugins.totp.**TOTP**

Bases: *AuthMethodHandler*

authenticate(*auth_payload*)

Try to authenticate using TOTP.

Module contents

Submodules

keystone.auth.core module

class keystone.auth.core.**AuthContext**

Bases: dict

Retrofitting auth_context to reconcile identity attributes.

The identity attributes must not have conflicting values among the auth plug-ins. The only exception is *expires_at*, which is set to its earliest value.

IDENTITY_ATTRIBUTES = frozenset({'access_token_id', 'domain_id', 'expires_at', 'project_id', 'user_id'})

update(*E=None*, ***F*)

Override update to prevent conflicting values.

class keystone.auth.core.**AuthInfo**(*auth=None*)

Bases: *ProviderAPIMixin*

Encapsulation of auth request.

static create(*auth=None*, *scope_only=False*)

get_method_data(*method*)

Get the auth method payload.

Returns
auth method payload

get_method_names()

Return the identity method names.

Returns
list of auth method names

get_scope()

Get scope information.

Verify and return the scoping information.

Returns
(domain_id, project_id, trust_ref, unscoped, system). If scope to a project, (None, project_id, None, None, None) will be returned. If scoped to a domain, (domain_id, None, None, None, None) will be returned. If scoped to a trust, (None, project_id, trust_ref, None, None), Will be returned, where the project_id comes from the trust definition. If unscoped, (None, None, None, unscoped, None) will be returned. If system_scoped, (None, None, None, None, all) will be returned.

set_scope(*domain_id=None*, *project_id=None*, *trust=None*, *unscoped=None*, *system=None*)

Set scope information.

class keystone.auth.core.**UserMFARulesValidator**

Bases: *ProviderAPIMixin*

Helper object that can validate the MFA Rules.

classmethod check_auth_methods_against_rules(*user_id*, *auth_methods*)

Validate the MFA rules against the successful auth methods.

Parameters

- **user_id** (*str*) The users ID (uuid).
- **auth_methods** (*set*) List of methods that were used for auth

Returns

Boolean, `True` means rules match and auth may proceed, `False` means rules do not match.

keystone.auth.core.**get_auth_method**(*method_name*)

keystone.auth.core.**load_auth_method**(*method*)

keystone.auth.core.**load_auth_methods**()

keystone.auth.schema module

keystone.auth.schema.**validate_issue_token_auth**(*auth=None*)

Module contents

keystone.catalog package

Subpackages

keystone.catalog.backends package

Submodules

keystone.catalog.backends.base module

class keystone.catalog.backends.base.**CatalogDriverBase**

Bases: *ProviderAPIMixin*

Interface description for the Catalog driver.

abstract add_endpoint_group_to_project(*endpoint_group_id*, *project_id*)

Add an endpoint group to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint to associate
- **project_id** (*string*) identity of project to associate

Raises
keystone.exception.Conflict If the endpoint group was already added to the project.

Returns
None.

abstract add_endpoint_to_project(*endpoint_id*, *project_id*)

Create an endpoint to project association.

Parameters

- **endpoint_id** (*string*) identity of endpoint to associate
- **project_id** (*string*) identity of the project to be associated with

Raises
keystone.exception.Conflict: If the endpoint was already added to project.

Returns
None.

abstract check_endpoint_in_project(*endpoint_id*, *project_id*)

Check if an endpoint is associated with a project.

Parameters

- **endpoint_id** (*string*) identity of endpoint to check
- **project_id** (*string*) identity of the project associated with

Raises
keystone.exception.NotFound If the endpoint was not found in the project.

Returns
None.

abstract create_endpoint(*endpoint_id*, *endpoint_ref*)

Create a new endpoint for a service.

Raises

- *keystone.exception.Conflict* If a duplicate endpoint exists.
- *keystone.exception.ServiceNotFound* If the service doesnt exist.

abstract create_endpoint_group(*endpoint_group*)

Create an endpoint group.

Parameters
endpoint_group (*dictionary*) endpoint group to create

Raises
keystone.exception.Conflict: If a duplicate endpoint group already exists.

Returns
an endpoint group representation.

abstract create_region(*region_ref*)

Create a new region.

Raises

- *keystone.exception.Conflict* If the region already exists.
- *keystone.exception.RegionNotFound* If the parent region is invalid.

abstract create_service(*service_id*, *service_ref*)

Create a new service.

Raises

keystone.exception.Conflict If a duplicate service exists.

abstract delete_association_by_endpoint(*endpoint_id*)

Remove all the endpoints to project association with endpoint.

Parameters

endpoint_id (*string*) identity of endpoint to check

Returns

None

abstract delete_association_by_project(*project_id*)

Remove all the endpoints to project association with project.

Parameters

project_id (*string*) identity of the project to check

Returns

None

abstract delete_endpoint(*endpoint_id*)

Delete an endpoint for a service.

Raises

keystone.exception.EndpointNotFound If the endpoint doesnt exist.

abstract delete_endpoint_group(*endpoint_group_id*)

Delete an endpoint group.

Parameters

endpoint_group_id (*string*) identity of endpoint group to delete

Raises

keystone.exception.NotFound If the endpoint group was not found.

Returns

None.

abstract delete_endpoint_group_association_by_project(*project_id*)

Remove endpoint group to project associations.

Parameters

project_id (*string*) identity of the project to check

Returns

None

abstract delete_region(*region_id*)

Delete an existing region.

Raises
keystone.exception.RegionNotFound If the region doesnt exist.

abstract delete_service(*service_id*)

Delete an existing service.

Raises
keystone.exception.ServiceNotFound If the service doesnt exist.

abstract get_catalog(*user_id*, *project_id*)

Retrieve and format the current service catalog.

Example:

```
{ 'RegionOne':
    {'compute': {
        'adminURL': u'http://host:8774/v1.1/project_id',
        'internalURL': u'http://host:8774/v1.1/project_id',
        'name': 'Compute Service',
        'publicURL': u'http://host:8774/v1.1/project_id'},
     'ec2': {
        'adminURL': 'http://host:8773/services/Admin',
        'internalURL': 'http://host:8773/services/Cloud',
        'name': 'EC2 Service',
        'publicURL': 'http://host:8773/services/Cloud'}}
```

Returns
A nested dict representing the service catalog or an empty dict.

Raises
keystone.exception.NotFound If the endpoint doesnt exist.

abstract get_endpoint(*endpoint_id*)

Get endpoint by id.

Returns
endpoint_ref dict

Raises
keystone.exception.EndpointNotFound If the endpoint doesnt exist.

abstract get_endpoint_group(*endpoint_group_id*)

Get an endpoint group.

Parameters
endpoint_group_id (*string*) identity of endpoint group to retrieve

Raises
keystone.exception.NotFound If the endpoint group was not found.

Returns
an endpoint group representation.

abstract get_endpoint_group_in_project(*endpoint_group_id*, *project_id*)

Get endpoint group to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint group to retrieve
- **project_id** (*string*) identity of project to associate

Raises

keystone.exception.NotFound If the endpoint group to the project association was not found.

Returns

a project endpoint group representation.

abstract get_region(*region_id*)

Get region by id.

Returns

region_ref dict

Raises

keystone.exception.RegionNotFound If the region doesnt exist.

abstract get_service(*service_id*)

Get service by id.

Returns

service_ref dict

Raises

keystone.exception.ServiceNotFound If the service doesnt exist.

get_v3_catalog(*user_id*, *project_id*)

Retrieve and format the current V3 service catalog.

Example:

```
[
    {
        "endpoints": [
        {
            "interface": "public",
            "id": "--endpoint-id--",
            "region": "RegionOne",
            "url": "http://external:8776/v1/--project-id--"
        },
        {
            "interface": "internal",
            "id": "--endpoint-id--",
            "region": "RegionOne",
            "url": "http://internal:8776/v1/--project-id--"
        }],
    "id": "--service-id--",
    "type": "volume"
}]
```

Returns
A list representing the service catalog or an empty list

Raises
keystone.exception.NotFound If the endpoint doesnt exist.

abstract list_endpoint_groups(*hints*)

List all endpoint groups.

Returns
None.

abstract list_endpoint_groups_for_project(*project_id*)

List all endpoint group to project associations for a project.

Parameters
project_id (*string*) identity of project to associate

Returns
None.

abstract list_endpoints(*hints*)

List all endpoints.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
list of endpoint_refs or an empty list.

abstract list_endpoints_for_project(*project_id*)

List all endpoints associated with a project.

Parameters
project_id (*string*) identity of the project to check

Returns
a list of identity endpoint ids or an empty list.

abstract list_projects_associated_with_endpoint_group(*endpoint_group_id*)

List all projects associated with endpoint group.

Parameters
endpoint_group_id (*string*) identity of endpoint to associate

Returns
None.

abstract list_projects_for_endpoint(*endpoint_id*)

List all projects associated with an endpoint.

Parameters
endpoint_id (*string*) identity of endpoint to check

Returns
a list of projects or an empty list.

abstract list_regions(*hints*)

List all regions.

Parameters

hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns

list of region_refs or an empty list.

abstract list_services(*hints*)

List all services.

Parameters

hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns

list of service_refs or an empty list.

abstract remove_endpoint_from_project(*endpoint_id*, *project_id*)

Remove an endpoint to project association.

Parameters

- **endpoint_id** (*string*) identity of endpoint to remove
- **project_id** (*string*) identity of the project associated with

Raises

keystone.exception.NotFound If the endpoint was not found in the project.

Returns

None.

abstract remove_endpoint_group_from_project(*endpoint_group_id*, *project_id*)

Remove an endpoint to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint to associate
- **project_id** (*string*) identity of project to associate

Raises

keystone.exception.NotFound If endpoint group project association was not found.

Returns

None.

abstract update_endpoint(*endpoint_id*, *endpoint_ref*)

Get endpoint by id.

Returns

endpoint_ref dict

Raises

- *keystone.exception.EndpointNotFound* If the endpoint doesnt exist.

- *keystone.exception.ServiceNotFound* If the service doesnt exist.

abstract update_endpoint_group(*endpoint_group_id*, *endpoint_group*)

Update an endpoint group.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint group to retrieve
- **endpoint_group** (*dictionary*) A full or partial endpoint_group

Raises

keystone.exception.NotFound If the endpoint group was not found.

Returns

an endpoint group representation.

abstract update_region(*region_id*, *region_ref*)

Update region by id.

Returns

region_ref dict

Raises

keystone.exception.RegionNotFound If the region doesnt exist.

abstract update_service(*service_id*, *service_ref*)

Update service by id.

Returns

service_ref dict

Raises

keystone.exception.ServiceNotFound If the service doesnt exist.

keystone.catalog.backends.sql module

class keystone.catalog.backends.sql.**Catalog**

Bases: *CatalogDriverBase*

add_endpoint_group_to_project(*endpoint_group_id*, *project_id*)

Add an endpoint group to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint to associate
- **project_id** (*string*) identity of project to associate

Raises

keystone.exception.Conflict If the endpoint group was already added to the project.

Returns

None.

add_endpoint_to_project(*endpoint_id*, *project_id*)

Create an endpoint to project association.

Parameters

- **endpoint_id** (*string*) identity of endpoint to associate
- **project_id** (*string*) identity of the project to be associated with

Raises
keystone.exception.Conflict: If the endpoint was already added to project.

Returns
None.

check_endpoint_in_project(*endpoint_id*, *project_id*)

Check if an endpoint is associated with a project.

Parameters

- **endpoint_id** (*string*) identity of endpoint to check
- **project_id** (*string*) identity of the project associated with

Raises
keystone.exception.NotFound If the endpoint was not found in the project.

Returns
None.

create_endpoint(*endpoint_id*, *endpoint*)

Create a new endpoint for a service.

Raises

- *keystone.exception.Conflict* If a duplicate endpoint exists.
- *keystone.exception.ServiceNotFound* If the service doesnt exist.

create_endpoint_group(*endpoint_group_id*, *endpoint_group*)

Create an endpoint group.

Parameters
endpoint_group (*dictionary*) endpoint group to create

Raises
keystone.exception.Conflict: If a duplicate endpoint group already exists.

Returns
an endpoint group representation.

create_region(*region_ref*)

Create a new region.

Raises

- *keystone.exception.Conflict* If the region already exists.
- *keystone.exception.RegionNotFound* If the parent region is invalid.

create_service(*service_id*, *service_ref*)

Create a new service.

Raises
keystone.exception.Conflict If a duplicate service exists.

delete_association_by_endpoint(*endpoint_id*)

Remove all the endpoints to project association with endpoint.

Parameters
endpoint_id (*string*) identity of endpoint to check

Returns
None

delete_association_by_project(*project_id*)

Remove all the endpoints to project association with project.

Parameters
project_id (*string*) identity of the project to check

Returns
None

delete_endpoint(*endpoint_id*)

Delete an endpoint for a service.

Raises
keystone.exception.EndpointNotFound If the endpoint doesnt exist.

delete_endpoint_group(*endpoint_group_id*)

Delete an endpoint group.

Parameters
endpoint_group_id (*string*) identity of endpoint group to delete

Raises
keystone.exception.NotFound If the endpoint group was not found.

Returns
None.

delete_endpoint_group_association_by_project(*project_id*)

Remove endpoint group to project associations.

Parameters
project_id (*string*) identity of the project to check

Returns
None

delete_region(*region_id*)

Delete an existing region.

Raises
keystone.exception.RegionNotFound If the region doesnt exist.

delete_service(*service_id*)

Delete an existing service.

Raises
keystone.exception.ServiceNotFound If the service doesnt exist.

get_catalog(*user_id*, *project_id*)

Retrieve and format the V2 service catalog.

Parameters

- **user_id** The id of the user who has been authenticated for creating service catalog.
- **project_id** The id of the project. project_id will be None in the case this being called to create a catalog to go in a domain scoped token. In this case, any endpoint that requires a project_id as part of their URL will be skipped (as would a whole service if, as a consequence, it has no valid endpoints).

Returns

A nested dict representing the service catalog or an empty dict.

get_endpoint(*endpoint_id*)

Get endpoint by id.

Returns

endpoint_ref dict

Raises

keystone.exception.EndpointNotFound If the endpoint doesnt exist.

get_endpoint_group(*endpoint_group_id*)

Get an endpoint group.

Parameters

endpoint_group_id (*string*) identity of endpoint group to retrieve

Raises

keystone.exception.NotFound If the endpoint group was not found.

Returns

an endpoint group representation.

get_endpoint_group_in_project(*endpoint_group_id*, *project_id*)

Get endpoint group to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint group to retrieve
- **project_id** (*string*) identity of project to associate

Raises

keystone.exception.NotFound If the endpoint group to the project association was not found.

Returns

a project endpoint group representation.

get_region(*region_id*)

Get region by id.

Returns

region_ref dict

Raises
keystone.exception.RegionNotFound If the region doesnt exist.

`get_service`(*service_id*)

Get service by id.

Returns
service_ref dict

Raises
keystone.exception.ServiceNotFound If the service doesnt exist.

`get_v3_catalog`(*user_id*, *project_id*)

Retrieve and format the current V3 service catalog.

Parameters

- **user_id** The id of the user who has been authenticated for creating service catalog.
- **project_id** The id of the project. project_id will be None in the case this being called to create a catalog to go in a domain scoped token. In this case, any endpoint that requires a project_id as part of their URL will be skipped.

Returns
A list representing the service catalog or an empty list

`list_endpoint_groups`(*hints*)

List all endpoint groups.

Returns
None.

`list_endpoint_groups_for_project`(*project_id*)

List all endpoint group to project associations for a project.

Parameters
project_id (*string*) identity of project to associate

Returns
None.

`list_endpoints`(*hints*)

List all endpoints.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
list of endpoint_refs or an empty list.

`list_endpoints_for_project`(*project_id*)

List all endpoints associated with a project.

Parameters
project_id (*string*) identity of the project to check

Returns
a list of identity endpoint ids or an empty list.

`list_projects_associated_with_endpoint_group`(*endpoint_group_id*)

List all projects associated with endpoint group.

Parameters
endpoint_group_id (*string*) identity of endpoint to associate

Returns
None.

`list_projects_for_endpoint`(*endpoint_id*)

List all projects associated with an endpoint.

Parameters
endpoint_id (*string*) identity of endpoint to check

Returns
a list of projects or an empty list.

`list_regions`(*hints*)

List all regions.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
list of region_refs or an empty list.

`list_services`(*hints*)

List all services.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
list of service_refs or an empty list.

`remove_endpoint_from_project`(*endpoint_id*, *project_id*)

Remove an endpoint to project association.

Parameters

- **endpoint_id** (*string*) identity of endpoint to remove
- **project_id** (*string*) identity of the project associated with

Raises
keystone.exception.NotFound If the endpoint was not found in the project.

Returns
None.

`remove_endpoint_group_from_project`(*endpoint_group_id*, *project_id*)

Remove an endpoint to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint to associate

- **project_id** (*string*) identity of project to associate

Raises
keystone.exception.NotFound If endpoint group project association was not found.

Returns
None.

update_endpoint(*endpoint_id*, *endpoint_ref*)

Get endpoint by id.

Returns
endpoint_ref dict

Raises

- *keystone.exception.EndpointNotFound* If the endpoint doesnt exist.
- *keystone.exception.ServiceNotFound* If the service doesnt exist.

update_endpoint_group(*endpoint_group_id*, *endpoint_group*)

Update an endpoint group.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint group to retrieve
- **endpoint_group** (*dictionary*) A full or partial endpoint_group

Raises
keystone.exception.NotFound If the endpoint group was not found.

Returns
an endpoint group representation.

update_region(*region_id*, *region_ref*)

Update region by id.

Returns
region_ref dict

Raises
keystone.exception.RegionNotFound If the region doesnt exist.

update_service(*service_id*, *service_ref*)

Update service by id.

Returns
service_ref dict

Raises
keystone.exception.ServiceNotFound If the service doesnt exist.

class keystone.catalog.backends.sql.**Endpoint**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'interface', 'region_id', 'service_id', 'url', 'legacy_endpoint_id', 'enabled']

enabled

extra

classmethod from_dict(*endpoint_dict*)

Override from_dict to set enabled if missing.

id

interface

legacy_endpoint_id

region_id

service_id

url

class keystone.catalog.backends.sql.**EndpointGroup**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

Endpoint Groups table.

attributes = ['id', 'name', 'description', 'filters']

description

filters

id

mutable_attributes = frozenset({'description', 'filters', 'name'})

name

class keystone.catalog.backends.sql.**ProjectEndpoint**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

project-endpoint relationship table.

attributes = ['endpoint_id', 'project_id']

endpoint_id

project_id

class keystone.catalog.backends.sql.**ProjectEndpointGroupMembership**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

Project to Endpoint group relationship table.

attributes = ['endpoint_group_id', 'project_id']

endpoint_group_id

project_id

class keystone.catalog.backends.sql.**Region**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'description', 'parent_region_id']

description

endpoints

extra

id

parent_region_id

class keystone.catalog.backends.sql.**Service**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'type', 'enabled']

enabled

endpoints

extra

id

type

keystone.catalog.backends.templated module

class keystone.catalog.backends.templated.**Catalog**(*templates=None*)

Bases: *CatalogDriverBase*

A backend that generates endpoints for the Catalog based on templates.

It is usually configured via config entries that look like:

catalog.$REGION.$SERVICE.$key = $value

and is stored in a similar looking hierarchy.

When expanding the template it will pass in a dict made up of the conf instance plus a few additional key-values, notably project_id and user_id.

It does not care what the keys and values are but it is worth noting that keystone_compat will expect certain keys to be there so that it can munge them into the output format keystone expects. These keys are:

name - the name of the service, most likely repeated for all services of the same type, across regions.

adminURL - the url of the admin endpoint

publicURL - the url of the public endpoint

internalURL - the url of the internal endpoint

add_endpoint_group_to_project(*endpoint_group_id*, *project_id*)

Add an endpoint group to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint to associate
- **project_id** (*string*) identity of project to associate

Raises

keystone.exception.Conflict If the endpoint group was already added to the project.

Returns

None.

add_endpoint_to_project(*endpoint_id*, *project_id*)

Create an endpoint to project association.

Parameters

- **endpoint_id** (*string*) identity of endpoint to associate
- **project_id** (*string*) identity of the project to be associated with

Raises

keystone.exception.Conflict: If the endpoint was already added to project.

Returns

None.

check_endpoint_in_project(*endpoint_id*, *project_id*)

Check if an endpoint is associated with a project.

Parameters

- **endpoint_id** (*string*) identity of endpoint to check
- **project_id** (*string*) identity of the project associated with

Raises

keystone.exception.NotFound If the endpoint was not found in the project.

Returns

None.

create_endpoint(*endpoint_id*, *endpoint_ref*)

Create a new endpoint for a service.

Raises

- *keystone.exception.Conflict* If a duplicate endpoint exists.
- *keystone.exception.ServiceNotFound* If the service doesnt exist.

create_endpoint_group(*endpoint_group*)

Create an endpoint group.

Parameters

endpoint_group (*dictionary*) endpoint group to create

Raises
keystone.exception.Conflict: If a duplicate endpoint group already exists.

Returns
an endpoint group representation.

`create_region`(*region_ref*)

Create a new region.

Raises

- *keystone.exception.Conflict* If the region already exists.
- *keystone.exception.RegionNotFound* If the parent region is invalid.

`create_service`(*service_id*, *service_ref*)

Create a new service.

Raises
keystone.exception.Conflict If a duplicate service exists.

`delete_association_by_endpoint`(*endpoint_id*)

Remove all the endpoints to project association with endpoint.

Parameters
endpoint_id (*string*) identity of endpoint to check

Returns
None

`delete_association_by_project`(*project_id*)

Remove all the endpoints to project association with project.

Parameters
project_id (*string*) identity of the project to check

Returns
None

`delete_endpoint`(*endpoint_id*)

Delete an endpoint for a service.

Raises
keystone.exception.EndpointNotFound If the endpoint doesnt exist.

`delete_endpoint_group`(*endpoint_group_id*)

Delete an endpoint group.

Parameters
endpoint_group_id (*string*) identity of endpoint group to delete

Raises
keystone.exception.NotFound If the endpoint group was not found.

Returns
None.

`delete_endpoint_group_association_by_project`(*project_id*)

Remove endpoint group to project associations.

Parameters
project_id (*string*) identity of the project to check

Returns
None

delete_region(*region_id*)

Delete an existing region.

Raises
keystone.exception.RegionNotFound If the region doesnt exist.

delete_service(*service_id*)

Delete an existing service.

Raises
keystone.exception.ServiceNotFound If the service doesnt exist.

get_catalog(*user_id*, *project_id*)

Retrieve and format the V2 service catalog.

Parameters

- **user_id** The id of the user who has been authenticated for creating service catalog.
- **project_id** The id of the project. project_id will be None in the case this being called to create a catalog to go in a domain scoped token. In this case, any endpoint that requires a project_id as part of their URL will be skipped.

Returns
A nested dict representing the service catalog or an empty dict.

get_endpoint(*endpoint_id*)

Get endpoint by id.

Returns
endpoint_ref dict

Raises
keystone.exception.EndpointNotFound If the endpoint doesnt exist.

get_endpoint_group(*endpoint_group_id*)

Get an endpoint group.

Parameters
endpoint_group_id (*string*) identity of endpoint group to retrieve

Raises
keystone.exception.NotFound If the endpoint group was not found.

Returns
an endpoint group representation.

get_endpoint_group_in_project(*endpoint_group_id*, *project_id*)

Get endpoint group to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint group to retrieve

- **project_id** (*string*) identity of project to associate

Raises
keystone.exception.NotFound If the endpoint group to the project association was not found.

Returns
a project endpoint group representation.

get_region(*region_id*)

Get region by id.

Returns
region_ref dict

Raises
keystone.exception.RegionNotFound If the region doesnt exist.

get_service(*service_id*)

Get service by id.

Returns
service_ref dict

Raises
keystone.exception.ServiceNotFound If the service doesnt exist.

get_v3_catalog(*user_id*, *project_id*)

Retrieve and format the current V3 service catalog.

This implementation builds the V3 catalog from the V2 catalog.

Parameters

- **user_id** The id of the user who has been authenticated for creating service catalog.
- **project_id** The id of the project. project_id will be None in the case this being called to create a catalog to go in a domain scoped token. In this case, any endpoint that requires a project_id as part of their URL will be skipped.

Returns
A list representing the service catalog or an empty list

list_endpoint_groups(*hints*)

List all endpoint groups.

Returns
None.

list_endpoint_groups_for_project(*project_id*)

List all endpoint group to project associations for a project.

Parameters
project_id (*string*) identity of project to associate

Returns
None.

`list_endpoints`(*hints*)

List all endpoints.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
list of endpoint_refs or an empty list.

`list_endpoints_for_project`(*project_id*)

List all endpoints associated with a project.

Parameters
project_id (*string*) identity of the project to check

Returns
a list of identity endpoint ids or an empty list.

`list_projects_associated_with_endpoint_group`(*endpoint_group_id*)

List all projects associated with endpoint group.

Parameters
endpoint_group_id (*string*) identity of endpoint to associate

Returns
None.

`list_projects_for_endpoint`(*endpoint_id*)

List all projects associated with an endpoint.

Parameters
endpoint_id (*string*) identity of endpoint to check

Returns
a list of projects or an empty list.

`list_regions`(*hints*)

List all regions.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
list of region_refs or an empty list.

`list_services`(*hints*)

List all services.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
list of service_refs or an empty list.

remove_endpoint_from_project(*endpoint_id*, *project_id*)

Remove an endpoint to project association.

Parameters

- **endpoint_id** (*string*) identity of endpoint to remove
- **project_id** (*string*) identity of the project associated with

Raises

keystone.exception.NotFound If the endpoint was not found in the project.

Returns

None.

remove_endpoint_group_from_project(*endpoint_group_id*, *project_id*)

Remove an endpoint to project association.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint to associate
- **project_id** (*string*) identity of project to associate

Raises

keystone.exception.NotFound If endpoint group project association was not found.

Returns

None.

update_endpoint(*endpoint_id*, *endpoint_ref*)

Get endpoint by id.

Returns

endpoint_ref dict

Raises

- *keystone.exception.EndpointNotFound* If the endpoint doesnt exist.
- *keystone.exception.ServiceNotFound* If the service doesnt exist.

update_endpoint_group(*endpoint_group_id*, *endpoint_group*)

Update an endpoint group.

Parameters

- **endpoint_group_id** (*string*) identity of endpoint group to retrieve
- **endpoint_group** (*dictionary*) A full or partial endpoint_group

Raises

keystone.exception.NotFound If the endpoint group was not found.

Returns

an endpoint group representation.

update_region(*region_id*, *region_ref*)

Update region by id.

Returns
region_ref dict

Raises
keystone.exception.RegionNotFound If the region doesnt exist.

update_service(*service_id*, *service_ref*)

Update service by id.

Returns
service_ref dict

Raises
keystone.exception.ServiceNotFound If the service doesnt exist.

keystone.catalog.backends.templated.**parse_templates**(*template_lines*)

Module contents

Submodules

keystone.catalog.core module

Main entry point into the Catalog service.

class keystone.catalog.core.**Manager**

Bases: *Manager*

Default pivot point for the Catalog backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

add_endpoint_group_to_project(*endpoint_group_id*, *project_id*)

add_endpoint_to_project(*endpoint_id*, *project_id*)

create_endpoint(*endpoint_id*, *endpoint_ref*, *initiator=None*)

create_region(*region_ref*, *initiator=None*)

create_service(*service_id*, *service_ref*, *initiator=None*)

delete_association_by_endpoint(*endpoint_id*)

delete_association_by_project(*project_id*)

delete_endpoint(*endpoint_id*, *initiator=None*)

delete_endpoint_group_association_by_project(*project_id*)

delete_region(*region_id*, *initiator=None*)

delete_service(*service_id*, *initiator=None*)

driver_namespace: str = 'keystone.catalog'

get_endpoint(*endpoint_id*)

get_endpoint_groups_for_project(*project_id*)

get_endpoints_filtered_by_endpoint_group(*endpoint_group_id*)

get_region(*region_id*)

get_service(*service_id*)

get_v3_catalog(*user_id*, *project_id*)

list_endpoints(*hints=None*)

list_endpoints_for_project(*project_id*)

List all endpoints associated with a project.

Parameters

project_id (*string*) project identifier to check

Returns

a list of endpoint ids or an empty list.

list_regions(*hints=None*)

list_services(*hints=None*)

remove_endpoint_from_project(*endpoint_id*, *project_id*)

remove_endpoint_group_from_project(*endpoint_group_id*, *project_id*)

update_endpoint(*endpoint_id*, *endpoint_ref*, *initiator=None*)

update_region(*region_id*, *region_ref*, *initiator=None*)

update_service(*service_id*, *service_ref*, *initiator=None*)

keystone.catalog.schema module

Module contents

keystone.cmd package

Subpackages

keystone.cmd.doctor package

Submodules

keystone.cmd.doctor.caching module

keystone.cmd.doctor.caching.**symptom_caching_disabled**()

keystone.conf [cache] enabled is not enabled.

Caching greatly improves the performance of keystone, and it is highly recommended that you enable it.

keystone.cmd.doctor.caching.**symptom_caching_enabled_without_a_backend**()

Caching is not completely configured.

Although caching is enabled in *keystone.conf [cache] enabled*, the default backend is still set to the no-op backend. Instead, configure keystone to point to a real caching backend like memcached.

keystone.cmd.doctor.caching.**symptom_connection_to_memcached**()

Memcached isnt reachable.

Caching is enabled and the *keystone.conf [cache] backend* option is configured but one or more Memcached servers are not reachable or marked as dead. Please ensure *keystone.conf [cache] memcache_servers* is configured properly.

keystone.cmd.doctor.credential module

keystone.cmd.doctor.credential.**symptom_keys_in_credential_fernet_key_repository**()

Credential key repository is empty.

After configuring keystone to use the Fernet credential provider, you should use *keystone-manage credential_setup* to initially populate your key repository with keys, and periodically rotate your keys with *keystone-manage credential_rotate*.

keystone.cmd.doctor.credential.**symptom_unique_key_repositories**()

Key repositories for encryption should be unique.

Even though credentials are encrypted using the same mechanism as Fernet tokens, they should have key repository locations that are independent of one another. Using the same repository to encrypt credentials and tokens can be considered a security vulnerability because ciphertext from the keys used to encrypt credentials is exposed as the token ID. Sharing a key repository can also lead to premature key removal during key rotation. This could result in indecipherable credentials, rendering them completely useless, or early token invalidation because the key that was used to encrypt the entity has been deleted.

Ensure *keystone.conf [credential] key_repository* and *keystone.conf [fernet_tokens] key_repository* are not pointing to the same location.

keystone.cmd.doctor.credential.**symptom_usability_of_credential_fernet_key_repository**()

Credential key repository is not setup correctly.

The credential Fernet key repository is expected to be readable by the user running keystone, but not world-readable, because it contains security sensitive secrets.

keystone.cmd.doctor.database module

keystone.cmd.doctor.database.**symptom_database_connection_is_not_SQLite**()

SQLite is not recommended for production deployments.

SQLite does not enforce type checking and has limited support for migrations, making it unsuitable for use in keystone. Please change your *keystone.conf [database] connection* value to point to a supported database driver, such as MySQL.

keystone.cmd.doctor.debug module

keystone.cmd.doctor.debug.**symptom_debug_mode_is_enabled**()

Debug mode should be set to False.

Debug mode can be used to get more information back when trying to isolate a problem, but it is not recommended to be enabled when running a production environment.

Ensure *keystone.conf debug* is set to False

keystone.cmd.doctor.federation module

keystone.cmd.doctor.federation.**symptom_comma_in_SAML_private_key_file_path**()

[saml] certfile should not contain a comma (,).

Because a comma is part of the API between keystone and the external xmlsec1 binary which utilizes the key, keystone cannot include a comma in the path to the private key file.

keystone.cmd.doctor.federation.**symptom_comma_in_SAML_public_certificate_path**()

[saml] certfile should not contain a comma (,).

Because a comma is part of the API between keystone and the external xmlsec1 binary which utilizes the certificate, keystone cannot include a comma in the path to the public certificate file.

keystone.cmd.doctor.ldap module

keystone.cmd.doctor.ldap.**symptom_LDAP_file_based_domain_specific_configs**()

Domain specific driver directory is invalid or contains invalid files.

If *keystone.conf [identity] domain_specific_drivers_enabled* is set to *true*, then support is enabled for individual domains to have their own identity drivers. The configurations for these can either be stored in a config file or in the database. The case we handle in this symptom is when they are stored in config files, which is indicated by *keystone.conf [identity] domain_configurations_from_database* being set to *false*.

keystone.cmd.doctor.ldap.**symptom_LDAP_file_based_domain_specific_configs_formatted_correctly**

LDAP domain specific configuration files are not formatted correctly.

If *keystone.conf [identity] domain_specific_drivers_enabled* is set to *true*, then support is enabled for individual domains to have their own identity drivers. The configurations for these

can either be stored in a config file or in the database. The case we handle in this symptom is when they are stored in config files, which is indicated by *keystone.conf [identity] domain_configurations_from_database* being set to false. The config files located in the directory specified by *keystone.conf [identity] domain_config_dir* should be in the form of *keystone.<domain_name>.conf* and their contents should look something like this:

[ldap] url = ldap://ldapservice.thecustomer.com query_scope = sub

user_tree_dn = ou=Users,dc=openstack,dc=org user_objectclass = MyOrgPerson user_id_attribute = uid

keystone.cmd.doctor.ldap.**symptom_LDAP_group_members_are_ids_disabled**()

[ldap] group_members_are_ids is not enabled.

Because youve set *keystone.conf [ldap] group_objectclass = posixGroup*, we would have also expected you to enable set *keystone.conf [ldap] group_members_are_ids* because we suspect youre using Open Directory, which would contain user IDs in a *posixGroup* rather than LDAP DNs, as other object classes typically would.

keystone.cmd.doctor.ldap.**symptom_LDAP_user_enabled_emulation_dn_ignored**()

[ldap] user_enabled_emulation_dn is being ignored.

There is no reason to set this value unless *keystone.conf [ldap] user_enabled_emulation* is also enabled.

keystone.cmd.doctor.ldap.**symptom_LDAP_user_enabled_emulation_use_group_config_ignored**()

[ldap] user_enabled_emulation_use_group_config is being ignored.

There is no reason to set this value unless *keystone.conf [ldap] user_enabled_emulation* is also enabled.

keystone.cmd.doctor.security_compliance module

keystone.cmd.doctor.security_compliance.**symptom_invalid_password_regular_expression**()

Invalid password regular expression.

The password regular expression is invalid and users will not be able to make password changes until this has been corrected.

Ensure *[security_compliance] password_regex* is a valid regular expression.

keystone.cmd.doctor.security_compliance.**symptom_minimum_password_age_greater_than_expires_da**

Minimum password age should be less than the password expires days.

If the minimum password age is greater than or equal to the password expires days, then users would not be able to change their passwords before they expire.

Ensure *[security_compliance] minimum_password_age* is less than the *[security_compliance] password_expires_days*.

keystone.cmd.doctor.security_compliance.**symptom_password_regular_expression_description_not_**

Password regular expression description is not set.

The password regular expression is set, but the description is not. Thus, if a user fails the password regular expression, they will not receive a message to explain why their requested password was insufficient.

Ensure *[security_compliance] password_regex_description* is set with a description of your password regular expression in a language for humans.

keystone.cmd.doctor.tokens module

keystone.cmd.doctor.tokens.**symptom_unreasonable_max_token_size**()

keystone.conf [DEFAULT] max_token_size should be adjusted.

This option is intended to protect keystone from unreasonably sized tokens, where reasonable is mostly dependent on the *keystone.conf [token] provider* that youre using. If youre using one of the following token providers, then you should set *keystone.conf [DEFAULT] max_token_size* accordingly:

- For Fernet, set *keystone.conf [DEFAULT] max_token_size = 255*, because Fernet tokens should never exceed this length in most deployments. However, if you are also using *keystone.conf [identity] driver = ldap*, Fernet tokens may not be built using an efficient packing method, depending on the IDs returned from LDAP, resulting in longer Fernet tokens (adjust your *max_token_size* accordingly).

keystone.cmd.doctor.tokens_fernet module

keystone.cmd.doctor.tokens_fernet.**symptom_keys_in_Fernet_key_repository**()

Fernet key repository is empty.

After configuring keystone to use the Fernet token provider, you should use *keystone-manage fernet_setup* to initially populate your key repository with keys, and periodically rotate your keys with *keystone-manage fernet_rotate*.

keystone.cmd.doctor.tokens_fernet.**symptom_usability_of_Fernet_key_repository**()

Fernet key repository is not setup correctly.

The Fernet key repository is expected to be readable by the user running keystone, but not world-readable, because it contains security-sensitive secrets.

Module contents

keystone.cmd.doctor.**diagnose**()

Report diagnosis for any symptoms we find.

Returns true when any symptoms are found, false otherwise.

keystone.cmd.doctor.**gather_symptoms**()

Gather all of the objects in this module that are named symptom_*.

Submodules

keystone.cmd.bootstrap module

class keystone.cmd.bootstrap.**Bootstrapper**

Bases: object

bootstrap()

keystone.cmd.cli module

class keystone.cmd.cli.**BaseApp**

Bases: object

classmethod add_argument_parser(*subparsers*)

name: str

class keystone.cmd.cli.**BasePermissionsSetup**

Bases: *BaseApp*

Common user/group setup for file permissions.

classmethod add_argument_parser(*subparsers*)

static get_user_group()

classmethod initialize_fernet_repository(*keystone_user_id*, *keystone_group_id*, *config_group=None*)

classmethod rotate_fernet_repository(*keystone_user_id*, *keystone_group_id*, *config_group=None*)

class keystone.cmd.cli.**BootStrap**

Bases: *BaseApp*

Perform the basic bootstrap process.

classmethod add_argument_parser(*subparsers*)

do_bootstrap()

Perform the bootstrap actions.

Create bootstrap user, project, and role so that CMS, humans, or scripts can continue to perform initial setup (domains, projects, services, endpoints, etc) of Keystone when standing up a new deployment.

classmethod main()

name: str = 'bootstrap'

class keystone.cmd.cli.**CreateJWSKeyPair**

Bases: *BasePermissionsSetup*

Create a key pair for signing and validating JWS tokens.

This command creates a public and private key pair to use for signing and validating JWS token signatures. The key pair is written to the directory where the command is invoked.

classmethod add_argument_parser(*subparsers*)

classmethod main()

name: str = 'create_jws_keypair'

class keystone.cmd.cli.**CredentialMigrate**

Bases: *BasePermissionsSetup*

Provides the ability to encrypt credentials using a new primary key.

This assumes that there is already a credential key repository in place and that the database backend has been upgraded to at least the Newton schema. If the credential repository doesnt exist yet, you can use `keystone-manage credential_setup` to create one.

classmethod main()

migrate_credentials()

name: str = 'credential_migrate'

class keystone.cmd.cli.**CredentialRotate**

Bases: *BasePermissionsSetup*

Rotate Fernet encryption keys for credential encryption.

This assumes you have already run *keystone-manage credential_setup*.

A new primary key is placed into rotation only if all credentials are encrypted with the current primary key. If any credentials are encrypted with a secondary key the rotation will abort. This protects against removing a key that is still required to decrypt credentials. Once a key is removed from the repository, it is impossible to recover the original data without restoring from a backup external to keystone (more on backups below). To make sure all credentials are encrypted with the latest primary key, please see the *keystone-manage credential_migrate* command. Since the maximum number of keys in the credential repository is 3, once all credentials are encrypted with the latest primary key we can safely introduce a new primary key. All credentials will still be decryptable since they are all encrypted with the only secondary key in the repository.

It is imperitive to understand the importance of backing up keys used to encrypt credentials. In the event keys are overrotated, applying a key repository from backup can help recover otherwise useless credentials. Persisting snapshots of the key repository in secure and encrypted source control, or a dedicated key management system are good examples of encryption key backups.

The *keystone-manage credential_rotate* and *keystone-manage credential_migrate* commands are intended to be done in sequence. After performing a rotation, a migration must be done before performing another rotation. This ensures we dont over-rotate encryption keys.

classmethod main()

name: str = 'credential_rotate'

validate_primary_key()

class keystone.cmd.cli.**CredentialSetup**

Bases: *BasePermissionsSetup*

Setup a Fernet key repository for credential encryption.

The purpose of this command is very similar to *keystone-manage fernet_setup* only the keys included in this repository are for encrypting and decrypting credential secrets instead of token payloads. Keys can be rotated using *keystone-manage credential_rotate*.

classmethod main()

name: str = 'credential_setup'

class keystone.cmd.cli.**DbSync**

Bases: *BaseApp*

Sync the database.

classmethod add_argument_parser(*subparsers*)

classmethod check_db_sync_status()

static main()

name: str = 'db_sync'

class keystone.cmd.cli.**DbVersion**

Bases: *BaseApp*

Print the current migration version of the database.

static main()

name: str = 'db_version'

class keystone.cmd.cli.**Doctor**

Bases: *BaseApp*

Diagnose common problems with keystone deployments.

classmethod add_argument_parser(*subparsers*)

static main()

name: str = 'doctor'

class keystone.cmd.cli.**DomainConfigUpload**

Bases: *BaseApp*

Upload the domain specific configuration files to the database.

classmethod add_argument_parser(*subparsers*)

static main()

name: str = 'domain_config_upload'

class keystone.cmd.cli.**DomainConfigUploadFiles**(*domain_config_finder=<function _domain_config_finder>*)

Bases: object

load_backends()

read_domain_configs_from_files()

Read configs from file(s) and load into database.

The command line parameters have already been parsed and the CONF command option will have been set. It is either set to the name of an explicit domain, or its None to indicate that we want all domain config files.

run()

valid_options()

Validate the options, returning True if they are indeed valid.

It would be nice to use the argparse automated checking for this validation, but the only way I can see doing that is to make the default (i.e. if no optional parameters are specified) to upload all configuration files - and that sounds too dangerous as a default. So we use it in a slightly unconventional way, where all parameters are optional, but you must specify at least one.

class keystone.cmd.cli.**FernetRotate**

Bases: *BasePermissionsSetup*

Rotate Fernet encryption keys.

This assumes you have already run keystone-manage fernet_setup.

A new primary key is placed into rotation, which is used for new tokens. The old primary key is demoted to secondary, which can then still be used for validating tokens. Excess secondary keys (beyond [fernet_tokens] max_active_keys) are revoked. Revoked keys are permanently deleted. A new staged key will be created and used to validate tokens. The next time key rotation takes place, the staged key will be put into rotation as the primary key.

Rotating keys too frequently, or with [fernet_tokens] max_active_keys set too low, will cause tokens to become invalid prior to their expiration.

classmethod main()

name: str = 'fernet_rotate'

class keystone.cmd.cli.**FernetSetup**

Bases: *BasePermissionsSetup*

Setup key repositories for Fernet tokens and auth receipts.

This also creates a primary key used for both creating and validating Fernet tokens and auth receipts. To improve security, you should rotate your keys (using keystone-manage fernet_rotate, for example).

classmethod main()

name: str = 'fernet_setup'

class keystone.cmd.cli.**MappingEngineTester**

Bases: *BaseApp*

Execute mapping engine locally.

classmethod add_argument_parser(*subparsers*)

classmethod main()

name: str = 'mapping_engine'

normalize_assertion()

normalize_rules()

read_assertion(*path*)

read_rules(*path*)

class keystone.cmd.cli.**MappingPopulate**

Bases: *BaseApp*

Pre-populate entries from domain-specific backends.

Running this command is not required. It should only be run right after the LDAP was configured, when many new users were added, or when mapping_purge is run.

This command will take a while to run. It is perfectly fine for it to run more than several minutes.

classmethod add_argument_parser(*subparsers*)

classmethod load_backends()

classmethod main()

Process entries for id_mapping_api.

name: str = 'mapping_populate'

class keystone.cmd.cli.**MappingPurge**

Bases: *BaseApp*

Purge the mapping table.

classmethod add_argument_parser(*subparsers*)

static main()

name: str = 'mapping_purge'

class keystone.cmd.cli.**ProjectSetup**

Bases: *BaseApp*

Create project with specified UUID.

classmethod add_argument_parser(*subparsers*)

do_project_setup()

Create project with specified UUID.

classmethod main()

name: str = 'project_setup'

class keystone.cmd.cli.**ReceiptRotate**

Bases: *BasePermissionsSetup*

Rotate auth receipts encryption keys.

This assumes you have already run keystone-manage receipt_setup.

A new primary key is placed into rotation, which is used for new receipts. The old primary key is demoted to secondary, which can then still be used for validating receipts. Excess secondary keys (beyond [receipt] max_active_keys) are revoked. Revoked keys are permanently deleted. A new staged key will be created and used to validate receipts. The next time key rotation takes place, the staged key will be put into rotation as the primary key.

Rotating keys too frequently, or with [receipt] max_active_keys set too low, will cause receipts to become invalid prior to their expiration.

classmethod main()

name: str = 'receipt_rotate'

class keystone.cmd.cli.**ReceiptSetup**

Bases: *BasePermissionsSetup*

Setup a key repository for auth receipts.

This also creates a primary key used for both creating and validating receipts. To improve security, you should rotate your keys (using keystone-manage receipt_rotate, for example).

classmethod main()

name: str = 'receipt_setup'

class keystone.cmd.cli.**ResetLastActive**

Bases: *BaseApp*

Reset null values for all users to current time.

classmethod add_argument_parser(*subparsers*)

static main()

name: str = 'reset_last_active'

class keystone.cmd.cli.**SamlIdentityProviderMetadata**

Bases: *BaseApp*

Generate Identity Provider metadata.

static main()

name: str = 'saml_idp_metadata'

class keystone.cmd.cli.**TokenRotate**

Bases: *BasePermissionsSetup*

Rotate token encryption keys.

This assumes you have already run keystone-manage token_setup.

A new primary key is placed into rotation, which is used for new tokens. The old primary key is demoted to secondary, which can then still be used for validating tokens. Excess secondary keys (beyond [token] max_active_keys) are revoked. Revoked keys are permanently deleted. A new staged key will be created and used to validate tokens. The next time key rotation takes place, the staged key will be put into rotation as the primary key.

Rotating keys too frequently, or with [token] max_active_keys set too low, will cause tokens to become invalid prior to their expiration.

classmethod main()

name: str = 'token_rotate'

class keystone.cmd.cli.**TokenSetup**

Bases: *BasePermissionsSetup*

Setup a key repository for tokens.

This also creates a primary key used for both creating and validating tokens. To improve security, you should rotate your keys (using keystone-manage token_rotate, for example).

classmethod main()

name: str = 'token_setup'

class keystone.cmd.cli.**TrustFlush**

Bases: *BaseApp*

Flush expired and non-expired soft deleted trusts from the backend.

classmethod add_argument_parser(*subparsers*)

classmethod main()

name: str = 'trust_flush'

class keystone.cmd.cli.**UserSetup**

Bases: *BaseApp*

Create user with specified UUID.

classmethod add_argument_parser(*subparsers*)

do_user_setup()

Create user with specified UUID.

classmethod main()

name: str = 'user_setup'

keystone.cmd.cli.**add_command_parsers**(*subparsers*)

keystone.cmd.cli.**main**(*argv=None*, *developer_config_file=None*)

Main entry point into the keystone-manage CLI utility.

Parameters

- **argv** (*list*) Arguments supplied via the command line using the sys standard library.
- **developer_config_file** (*string*) The location of a configuration file normally found in development environments.

keystone.cmd.idutils module

class keystone.cmd.idutils.**Identity**

Bases: object

project_setup()

user_setup()

keystone.cmd.manage module

keystone.cmd.manage.**main**()

keystone.cmd.status module

class keystone.cmd.status.**Checks**

Bases: UpgradeCommands

Programmable upgrade checks.

Each method here should be a programmable check that helps check for things that might cause issues for deployers in the upgrade process. A good example of an upgrade check would be to ensure all roles defined in policies actually exist within the roles backend.

check_default_roles_are_immutable()

check_trust_policies_are_not_empty()

keystone.cmd.status.**main**()

Module contents

keystone.common package

Subpackages

keystone.common.cache package

Submodules

keystone.common.cache.core module

Keystone Caching Layer Implementation.

class keystone.common.cache.core.**DistributedInvalidationStrategy**(*region_manager*)

Bases: RegionInvalidationStrategy

invalidate(*hard=None*)

Region invalidation.

CacheRegion propagated call. The default invalidation system works by setting a current timestamp (using time.time()) to consider all older timestamps effectively invalidated.

is_hard_invalidated(*timestamp*)

Check timestamp to determine if it was hard invalidated.

Returns

Boolean. True if timestamp is older than the last region invalidation time and region is invalidated in hard mode.

is_invalidated(*timestamp*)

Check timestamp to determine if it was invalidated.

Returns

Boolean. True if timestamp is older than the last region invalidation time.

is_soft_invalidated(*timestamp*)

Check timestamp to determine if it was soft invalidated.

Returns

Boolean. True if timestamp is older than the last region invalidation time and region is invalidated in soft mode.

was_hard_invalidated()

Indicate the region was invalidated in hard mode.

Returns

Boolean. True if region was invalidated in hard mode.

was_soft_invalidated()

Indicate the region was invalidated in soft mode.

Returns

Boolean. True if region was invalidated in soft mode.

class keystone.common.cache.core.**RegionInvalidationManager**(*invalidation_region*, *region_name*)

Bases: object

REGION_KEY_PREFIX = '<<<region>>>:'

invalidate_region()

is_region_key(*key*)

property region_id

keystone.common.cache.core.**configure_cache**(*region=None*)

keystone.common.cache.core.**configure_invalidation_region**()

keystone.common.cache.core.**create_region**(*name*)

Create a dopile region.

Wraps oslo_cache.core.create_region. This is used to ensure that the Region is properly patched and allows us to more easily specify a region name.

Parameters
name (*str*) The region name

Returns
The new region.

Return type
dogpile.cache.region.CacheRegion

keystone.common.cache.core.**get_memoization_decorator**(*group*, *expiration_group=None*, *region=None*)

keystone.common.cache.core.**key_mangler_factory**(*invalidation_manager*, *orig_key_mangler*)

Module contents

keystone.common.policies package

Submodules

keystone.common.policies.access_rule module

keystone.common.policies.access_rule.**list_rules**()

keystone.common.policies.access_token module

keystone.common.policies.access_token.**list_rules**()

keystone.common.policies.application_credential module

keystone.common.policies.application_credential.**list_rules**()

keystone.common.policies.auth module

keystone.common.policies.auth.**list_rules**()

keystone.common.policies.base module

keystone.common.policies.base.**list_rules**()

keystone.common.policies.consumer module

keystone.common.policies.consumer.**list_rules**()

keystone.common.policies.credential module

keystone.common.policies.credential.**list_rules**()

keystone.common.policies.domain module

keystone.common.policies.domain.**list_rules**()

keystone.common.policies.domain_config module

keystone.common.policies.domain_config.**list_rules**()

keystone.common.policies.ec2_credential module

keystone.common.policies.ec2_credential.**list_rules**()

keystone.common.policies.ec2tokens module

keystone.common.policies.ec2tokens.**list_rules**()

keystone.common.policies.endpoint module

keystone.common.policies.endpoint.**list_rules**()

keystone.common.policies.endpoint_group module

keystone.common.policies.endpoint_group.**list_rules**()

keystone.common.policies.grant module

keystone.common.policies.grant.**list_operations**(*paths*, *methods*)

keystone.common.policies.grant.**list_rules**()

keystone.common.policies.group module

keystone.common.policies.group.**list_rules**()

keystone.common.policies.identity_provider module

keystone.common.policies.identity_provider.**list_rules**()

keystone.common.policies.implied_role module

keystone.common.policies.implied_role.**list_rules**()

keystone.common.policies.limit module

keystone.common.policies.limit.**list_rules**()

keystone.common.policies.mapping module

keystone.common.policies.mapping.**list_rules**()

keystone.common.policies.policy module

keystone.common.policies.policy.**list_rules**()

keystone.common.policies.policy_association module

keystone.common.policies.policy_association.**list_rules**()

keystone.common.policies.project module

keystone.common.policies.project.**list_rules**()

keystone.common.policies.project_endpoint module

keystone.common.policies.project_endpoint.**list_rules**()

keystone.common.policies.protocol module

keystone.common.policies.protocol.**list_rules**()

keystone.common.policies.region module

keystone.common.policies.region.**list_rules**()

keystone.common.policies.registered_limit module

keystone.common.policies.registered_limit.**list_rules**()

keystone.common.policies.revoke_event module

keystone.common.policies.revoke_event.**list_rules**()

keystone.common.policies.role module

keystone.common.policies.role.**list_rules**()

keystone.common.policies.role_assignment module

keystone.common.policies.role_assignment.**list_rules**()

keystone.common.policies.s3tokens module

keystone.common.policies.s3tokens.**list_rules**()

keystone.common.policies.service module

keystone.common.policies.service.**list_rules**()

keystone.common.policies.service_provider module

keystone.common.policies.service_provider.**list_rules**()

keystone.common.policies.token module

keystone.common.policies.token.**list_rules**()

keystone.common.policies.token_revocation module

keystone.common.policies.token_revocation.**list_rules**()

keystone.common.policies.trust module

keystone.common.policies.trust.**list_rules**()

keystone.common.policies.user module

keystone.common.policies.user.**list_rules**()

Module contents

keystone.common.policies.**list_rules**()

keystone.common.rbac_enforcer package

Submodules

keystone.common.rbac_enforcer.enforcer module

class keystone.common.rbac_enforcer.enforcer.**RBACEnforcer**

Bases: object

Enforce RBAC on API calls.

ACTION_STORE_ATTR = 'keystone:RBAC:action_name'

classmethod enforce_call(*enforcer=None*, *action=None*, *target_attr=None*, *member_target_type=None*, *member_target=None*, *filters=None*, *build_target=None*)

Enforce RBAC on the current request.

This will do some legwork and then instantiate the Enforcer if an enforcer is not passed in.

Parameters

- **enforcer** (*RBACEnforcer*) A pre-instantiated Enforcer object (optional)
- **action** (*str*) the name of the rule/policy enforcement to be checked against, e.g. *identity:get_user* (optional may be replaced by decorating the method/function with *policy_enforcer_action*.
- **target_attr** (*dict*) complete override of the target data. This will replace all other generated target data meaning *member_target_type* and *member_target* are ignored. This will also prevent extraction of data from the X-Subject-Token. The *target* dict should contain a series of key-value pairs such as *{user: user_ref_dict}*.
- **member_target_type** (*str*) the type of the target, e.g. user. Both this and *member_target* must be passed if either is passed.
- **member_target** (*dict*) the (dict form) reference of the member object. Both this and *member_target_type* must be passed if either is passed.
- **filters** (*iterable*) A variable number of optional string filters, these are used to extract values from the query params. The filters are added to the request data that is passed to the enforcer and may be used to determine policy action. In practice these are mainly supplied in the various list APIs and are un-used in the default supplied policies.
- **build_target** (*function*) A function to build the target for enforcement. This is explicitly done after authentication in order to not leak existance data before auth.

classmethod policy_enforcer_action(*action*)

Decorator to set policy enforcement action name.

static register_rules(*enforcer*)

suppress_deprecation_warnings = False

keystone.common.rbac_enforcer.policy module

keystone.common.rbac_enforcer.policy.**get_enforcer**()

Entrypoint that must return the raw oslo.policy enforcer obj.

This is utilized by the command-line policy tools.

Returns

oslo_policy.policy.Enforcer

keystone.common.rbac_enforcer.policy.**reset**()

Module contents

class keystone.common.rbac_enforcer.**RBACEnforcer**

Bases: object

Enforce RBAC on API calls.

ACTION_STORE_ATTR = 'keystone:RBAC:action_name'

classmethod enforce_call(*enforcer=None*, *action=None*, *target_attr=None*, *member_target_type=None*, *member_target=None*, *filters=None*, *build_target=None*)

Enforce RBAC on the current request.

This will do some legwork and then instantiate the Enforcer if an enforcer is not passed in.

Parameters

- **enforcer** (*RBACEnforcer*) A pre-instantiated Enforcer object (optional)
- **action** (*str*) the name of the rule/policy enforcement to be checked against, e.g. *identity:get_user* (optional may be replaced by decorating the method/function with *policy_enforcer_action*.
- **target_attr** (*dict*) complete override of the target data. This will replace all other generated target data meaning *member_target_type* and *member_target* are ignored. This will also prevent extraction of data from the X-Subject-Token. The *target* dict should contain a series of key-value pairs such as *{user: user_ref_dict}*.
- **member_target_type** (*str*) the type of the target, e.g. user. Both this and *member_target* must be passed if either is passed.
- **member_target** (*dict*) the (dict form) reference of the member object. Both this and *member_target_type* must be passed if either is passed.
- **filters** (*iterable*) A variable number of optional string filters, these are used to extract values from the query params. The filters are added to the request data that is passed to the enforcer and may be used to determine policy action. In practice these are mainly supplied in the various list APIs and are un-used in the default supplied policies.
- **build_target** (*function*) A function to build the target for enforcement. This is explicitly done after authentication in order to not leak existance data before auth.

classmethod policy_enforcer_action(*action*)

Decorator to set policy enforcement action name.

static register_rules(*enforcer*)

suppress_deprecation_warnings = False

keystone.common.resource_options package

Subpackages

keystone.common.resource_options.options package

Submodules

keystone.common.resource_options.options.immutable module

keystone.common.resource_options.options.immutable.**check_immutable_delete**(*resource_ref*, *resource_type*, *resource_id*)

Check if a delete is allowed on a resource.

Parameters

- **resource_ref** dict reference of the resource
- **resource_type** resource type (str) e.g. project
- **resource_id** id of the resource (str) e.g. project[id]

Raises
ResourceDeleteForbidden

keystone.common.resource_options.options.immutable.**check_immutable_update**(*original_resource_ref*, *new_resource_ref*, *type*, *resource_id*)

Check if an update is allowed to an immutable resource.

Valid cases where an update is allowed:

- Resource is not immutable
- Resource is immutable, and update to set immutable to False or None

Parameters

- **original_resource_ref** a dict resource reference representing the current resource
- **new_resource_ref** a dict reference of the updates to perform
- **type** the resource type, e.g. project
- **resource_id** the id of the resource (e.g. project[id]), usually a UUID

Raises
ResourceUpdateForbidden

keystone.common.resource_options.options.immutable.**check_resource_immutable**(*resource_ref*)

Check to see if a resource is immutable.

Parameters

resource_ref a dict reference of a resource to inspect

Module contents

keystone.common.resource_options.options.**check_immutable_delete**(*resource_ref*, *resource_type*, *resource_id*)

Check if a delete is allowed on a resource.

Parameters

- **resource_ref** dict reference of the resource
- **resource_type** resource type (str) e.g. project
- **resource_id** id of the resource (str) e.g. project[id]

Raises

ResourceDeleteForbidden

keystone.common.resource_options.options.**check_immutable_update**(*original_resource_ref*, *new_resource_ref*, *type*, *resource_id*)

Check if an update is allowed to an immutable resource.

Valid cases where an update is allowed:

- Resource is not immutable
- Resource is immutable, and update to set immutable to False or None

Parameters

- **original_resource_ref** a dict resource reference representing the current resource
- **new_resource_ref** a dict reference of the updates to perform
- **type** the resource type, e.g. project
- **resource_id** the id of the resource (e.g. project[id]), usually a UUID

Raises

ResourceUpdateForbidden

keystone.common.resource_options.options.**check_resource_immutable**(*resource_ref*)

Check to see if a resource is immutable.

Parameters

resource_ref a dict reference of a resource to inspect

Submodules

keystone.common.resource_options.core module

Options specific to resources managed by Keystone (Domain, User, etc).

class keystone.common.resource_options.core.**ResourceOption**(*option_id*, *option_name*, *validator=<function _validator>*, *json_schema_validation=None*)

Bases: object

property json_schema

property option_id

property option_name

class keystone.common.resource_options.core.**ResourceOptionRegistry**(*registry_name*)

Bases: object

get_option_by_id(*opt_id*)

get_option_by_name(*name*)

property json_schema

property option_ids

property option_names

property options

property options_by_name

register_option(*option*)

keystone.common.resource_options.core.**boolean_validator**(*value*)

keystone.common.resource_options.core.**get_resource_option**(*model*, *option_id*)

Get the resource option information from the models mapper.

keystone.common.resource_options.core.**ref_mapper_to_dict_options**(*ref*)

Convert the values in _resource_option_mapper to options dict.

NOTE: this is to be called from the relevant *to_dict* methods or similar and must be called from within the active session context.

Parameters

ref the DB model ref to extract options from

Returns

Dict of options as expected to be returned out of to_dict in the *options* key.

keystone.common.resource_options.core.**resource_options_ref_to_mapper**(*ref*, *option_class*)

Convert the _resource_options property-dict to options attr map.

The model must have the resource option mapper located in the `_resource_option_mapper` attribute.

The model must have the resource option registry located in the `resource_options_registry` attribute.

The option dict with key(opt_id), value(opt_value) will be pulled from `ref._resource_options`.

NOTE: This function MUST be called within the active writer session context!

Parameters

- **ref** The DB model reference that is actually stored to the backend.
- **option_class** Class that is used to store the resource option in the DB.

Module contents

keystone.common.sql package

Submodules

keystone.common.sql.upgrades module

keystone.common.sql.upgrades.**check_bootstrap_new_branch**(*branch*, *version_path*, *addn_kwargs*)

Bootstrap a new migration branch if it does not exist.

keystone.common.sql.upgrades.**contract_schema**(*engine=None*)

Contract the database.

This is run manually by the keystone-manage command once the keystone nodes have been upgraded to the latest release and will remove any old tables/columns that are no longer required.

keystone.common.sql.upgrades.**expand_schema**(*engine=None*)

Expand the database schema ahead of data migration.

This is run manually by the keystone-manage command before the first keystone node is migrated to the latest release.

keystone.common.sql.upgrades.**get_alembic_config**()

keystone.common.sql.upgrades.**get_current_heads**()

Get the current head of each the expand and contract branches.

keystone.common.sql.upgrades.**get_db_version**(*branch='expand'*, ***, *engine=None*)

keystone.common.sql.upgrades.**get_head_revisions**()

Get the available head for each the expand and contract branches.

keystone.common.sql.upgrades.**get_version_branch_path**(*release=None*, *branch=None*)

Get the path to a version branch.

keystone.common.sql.upgrades.**migrate_data**(*engine=None*)

Migrate data to match the new schema.

This is run manually by the keystone-manage command once the keystone schema has been expanded for the new release.

keystone.common.sql.upgrades.**offline_sync_database_to_version**(*version=None*, *, *engine=None*)

Perform and off-line sync of the database.

Migrate the database up to the latest version, doing the equivalent of the cycle of expand, migrate and contract, for when an offline upgrade is being performed.

If a version is specified then only migrate the database up to that version. Downgrading is not supported. If version is specified, then only the main database migration is carried out - and the expand, migration and contract phases will NOT be run.

Module contents

keystone.common.validation package

Submodules

keystone.common.validation.parameter_types module

Common parameter types for validating a request reference.

keystone.common.validation.validators module

Internal implementation of request body validating middleware.

class keystone.common.validation.validators.**SchemaValidator**(*schema*)

Bases: object

Resource reference validator class.

validate(**args*, ***kwargs*)

validator_org

alias of Draft4Validator

keystone.common.validation.validators.**validate_password**(*password*)

Module contents

Request body validating middleware for OpenStack Identity resources.

keystone.common.validation.**lazy_validate**(*request_body_schema*, *resource_to_validate*)

A non-decorator way to validate a request, to be used inline.

Parameters

- **request_body_schema** a schema to validate the resource reference
- **resource_to_validate** dictionary to validate

Raises

- ***keystone.exception.ValidationError*** if *resource_to_validate* is None. (see wrapper method below).
- **TypeError** at decoration time when the expected resource to validate isnt found in the decorated methods signature

keystone.common.validation.**nullable**(*property_schema*)

Clone a property schema into one that is nullable.

Parameters

property_schema (*dict*) schema to clone into a nullable schema

Returns

a new dict schema

Submodules

keystone.common.authorization module

keystone.common.context module

class keystone.common.context.**RequestContext**(***kwargs*)

Bases: RequestContext

to_policy_values()

Add keystone-specific policy values to policy representation.

This method converts generic policy values to a dictionary form using the base implementation from oslo_context.context.RequestContext. Afterwards, it is going to pull keystone-specific values off the context and represent them as items in the policy values dictionary. This is because keystone uses default policies that rely on these values, so we need to guarantee they are present during policy enforcement if they are present on the context object.

This method is automatically called in oslo_policy.policy.Enforcer.enforce() if oslo.policy knows its dealing with a context object.

keystone.common.driver_hints module

class keystone.common.driver_hints.**Hints**

Bases: object

Encapsulate driver hints for listing entities.

Hints are modifiers that affect the return of entities from a list_<entities> operation. They are typically passed to a driver to give direction as to what filtering, pagination or list limiting actions are being requested.

It is optional for a driver to action some or all of the list hints, but any filters that it does satisfy must be marked as such by calling removing the filter from the list.

A Hint object contains filters, which is a list of dicts that can be accessed publicly. Also it contains a dict called limit, which will indicate the amount of data we want to limit our listing to.

If the filter is discovered to never match, then *cannot_match* can be set to indicate that there will not be any matches and the backend work can be short-circuited.

Each filter term consists of:

- name: the name of the attribute being matched
- value: the value against which it is being matched
- **comparator: the operation, which can be one of equals,** contains, startswith or endswith
- **case_sensitive: whether any comparison should take account of** case

add_filter(*name*, *value*, *comparator='equals'*, *case_sensitive=False*)

Add a filter to the filters list, which is publicly accessible.

get_exact_filter_by_name(*name*)

Return a filter key and value if exact filter exists for name.

set_limit(*limit*, *truncated=False*)

Set a limit to indicate the list should be truncated.

keystone.common.driver_hints.**truncated**(*f*)

Ensure list truncation is detected in Driver list entity methods.

This is designed to wrap Driver list_{entity} methods in order to calculate if the resultant list has been truncated. Provided a limit dict is found in the hints list, we increment the limit by one so as to ask the wrapped function for one more entity than the limit, and then once the list has been generated, we check to see if the original limit has been exceeded, in which case we truncate back to that limit and set the truncated boolean to true in the hints limit dict.

keystone.common.fernet_utils module

class keystone.common.fernet_utils.**FernetUtils**(*key_repository*, *max_active_keys*, *config_group*)

Bases: object

create_key_directory(*keystone_user_id=None*, *keystone_group_id=None*)

Attempt to create the key directory if it doesnt exist.

initialize_key_repository(*keystone_user_id=None*, *keystone_group_id=None*)

Create a key repository and bootstrap it with a key.

Parameters

- **keystone_user_id** User ID of the Keystone user.
- **keystone_group_id** Group ID of the Keystone user.

load_keys(*use_null_key=False*)

Load keys from disk into a list.

The first key in the list is the primary key used for encryption. All other keys are active secondary keys that can be used for decrypting tokens.

Parameters

use_null_key If true, a known key containing null bytes will be appended to the list of returned keys.

rotate_keys(*keystone_user_id=None*, *keystone_group_id=None*)

Create a new primary key and revoke excess active keys.

Parameters

- **keystone_user_id** User ID of the Keystone user.
- **keystone_group_id** Group ID of the Keystone user.

Key rotation utilizes the following behaviors:

- The highest key number is used as the primary key (used for encryption).
- All keys can be used for decryption.
- New keys are always created as key 0, which serves as a placeholder before promoting it to be the primary key.

This strategy allows you to safely perform rotation on one node in a cluster, before syncing the results of the rotation to all other nodes (during both key rotation and synchronization, all nodes must recognize all primary keys).

validate_key_repository(*requires_write=False*)

Validate permissions on the key repository directory.

keystone.common.json_home module

class keystone.common.json_home.**JsonHomeResources**

Bases: object

JSON Home resource data.

classmethod append_resource(*rel*, *data*)

classmethod resources()

class keystone.common.json_home.**Parameters**

Bases: object

Relationships for Common parameters.

ACCESS_RULE_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/access_rule_id'

APPLICATION_CRED_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/application_credential_id'

DOMAIN_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/domain_id'

ENDPOINT_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/endpoint_id'

GROUP_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/group_id'

LIMIT_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/limit_id'

POLICY_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/policy_id'

PROJECT_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/project_id'

REGION_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/region_id'

REGISTERED_LIMIT_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/registered_limit_id'

ROLE_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/role_id'

SERVICE_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/service_id'

TAG_VALUE = 'https://docs.openstack.org/api/openstack-identity/3/param/tag_value'

USER_ID = 'https://docs.openstack.org/api/openstack-identity/3/param/user_id'

class keystone.common.json_home.**Status**

Bases: object

Status values supported.

DEPRECATED = 'deprecated'

EXPERIMENTAL = 'experimental'

STABLE = 'stable'

classmethod **update_resource_data**(*resource_data*, *status*)

keystone.common.json_home.**build_v3_extension_parameter_relation**(*extension_name*, *extension_version*, *parameter_name*)

keystone.common.json_home.**build_v3_extension_resource_relation**(*extension_name*, *extension_version*, *resource_name*)

keystone.common.json_home.**build_v3_parameter_relation**(*parameter_name*)

keystone.common.json_home.**build_v3_resource_relation**(*resource_name*)

keystone.common.json_home.**translate_urls**(*json_home*, *new_prefix*)

Given a JSON Home document, sticks new_prefix on each of the urls.

keystone.common.jwt_utils module

keystone.common.jwt_utils.**create_jws_keypair**(*private_key_path*, *public_key_path*)

Create an ECDSA key pair using an secp256r1, or NIST P-256, curve.

Parameters

- **private_key_path** location to save the private key
- **public_key_path** location to save the public key

keystone.common.manager module

class keystone.common.manager.**Manager**(*driver_name*)

Bases: object

Base class for intermediary request layer.

The Manager layer exists to support additional logic that applies to all or some of the methods exposed by a service that are not specific to the HTTP interface.

It also provides a stable entry point to dynamic backends.

An example of a probable use case is logging all the calls.

driver_namespace: str

keystone.common.manager.**load_driver**(*namespace*, *driver_name*, **args*)

keystone.common.manager.**response_truncated**(*f*)

Truncate the list returned by the wrapped function.

This is designed to wrap Manager list_{entity} methods to ensure that any list limits that are defined are passed to the driver layer. If a hints list is provided, the wrapper will insert the relevant limit into the hints so that the underlying driver call can try and honor it. If the driver does truncate the response, it will update the truncated attribute in the limit entry in the hints list, which enables the caller of this function to know if truncation has taken place. If, however, the driver layer is unable to perform truncation, the limit entry is simply left in the hints list for the caller to handle.

A _get_list_limit() method is required to be present in the object class hierarchy, which returns the limit for this backend to which we will truncate.

If a hints list is not provided in the arguments of the wrapped call then any limits set in the config file are ignored. This allows internal use of such wrapped methods where the entire data set is needed as input for the calculations of some other API (e.g. get role assignments for a given project).

keystone.common.password_hashing module

keystone.common.password_hashing.**check_password**(*password*, *hashed*)

Check that a plaintext password matches hashed.

hashpw returns the salt value concatenated with the actual hash value. It extracts the actual salt if this value is then passed as the salt.

keystone.common.password_hashing.**hash_password**(*password*)

Hash a password. Harder.

keystone.common.password_hashing.**hash_user_password**(*user*)

Hash a user dicts password without modifying the passed-in dict.

keystone.common.password_hashing.**verify_length_and_trunc_password**(*password*)

Verify and truncate the provided password to the max_password_length.

We also need to check that the configured password hashing algorithm does not silently truncate the password. For example, passlib.hash.bcrypt does this: https://passlib.readthedocs.io/en/stable/lib/passlib.hash.bcrypt.html#security-issues

keystone.common.profiler module

keystone.common.profiler.**setup**(*name*, *host='0.0.0.0'*)

Setup OSprofiler notifier and enable profiling.

Parameters

- **name** name of the service that will be profiled
- **host** hostname or host IP address that the service will be running on. By default host will be set to 0.0.0.0, but more specified host name / address usage is highly recommended.

keystone.common.provider_api module

exception keystone.common.provider_api.**DuplicateProviderError**

Bases: Exception

Attempting to register a duplicate API provider.

class keystone.common.provider_api.**ProviderAPIMixin**

Bases: object

Allow referencing provider apis on self via __getattr__.

Be sure this class is first in the class definition for inheritance.

class keystone.common.provider_api.**ProviderAPIRegistry**

Bases: object

deferred_provider_lookup(*api*, *method*)

Create descriptor that performs lookup of api and method on demand.

For specialized cases, such as the enforcer get_member_from_driver which needs to be effectively a classmethod, this method returns a smart descriptor object that does the lookup at runtime instead of at import time.

Parameters

- **api** (*str*) The api to use, e.g. identity_api
- **method** (*str*) the method on the api to return

lock_provider_registry()

locked = False

keystone.common.render_token module

keystone.common.render_token.**render_token_response_from_model**(*token*, *include_catalog=True*)

keystone.common.tokenless_auth module

class keystone.common.tokenless_auth.**TokenlessAuthHelper**(*env*)

Bases: *ProviderAPIMixin*

get_mapped_user(*project_id=None*, *domain_id=None*)

Map client certificate to an existing user.

If user is ephemeral, there is no validation on the user himself; however it will be mapped to a corresponding group(s) and the scope of this ephemeral user is the same as what is assigned to the group.

Parameters

- **project_id** Project scope of the mapped user.
- **domain_id** Domain scope of the mapped user.

Returns
A dictionary that contains the keys, such as user_id, user_name, domain_id, domain_name

Return type
dict

`get_scope()`

keystone.common.utils module

`class keystone.common.utils.SmarterEncoder(*, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None)`

Bases: `JSONEncoder`

Help for JSON encoding dict-like objects.

`default(obj)`

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

`class keystone.common.utils.WhiteListedItemFilter(whitelist, data)`

Bases: `object`

`keystone.common.utils.attr_as_boolean(val_attr)`

Return the boolean value, decoded from a string.

We test explicitly for a value meaning False, which can be one of several formats as specified in oslo strutils.FALSE_STRINGS. All other string values (including an empty string) are treated as meaning True.

`keystone.common.utils.auth_str_equal(provided, known)`

Constant-time string comparison.

Params provided
the first string

Params known
the second string

Returns
True if the strings are equal.

This function takes two strings and compares them. It is intended to be used when doing a comparison for authentication purposes to help guard against timing attacks. When using the function for this purpose, always provide the user-provided password as the first argument. The time this function will take is always a factor of the length of this string.

keystone.common.utils.**check_endpoint_url**(*url*)

Check substitution of url.

The invalid urls are as follows: urls with substitutions that is not in the whitelist

Check the substitutions in the URL to make sure they are valid and on the whitelist.

Parameters
url (*str*) the URL to validate

Return type
None

Raises
keystone.exception.URLValidationError if the URL is invalid

keystone.common.utils.**create_directory**(*directory*, *keystone_user_id=None*, *keystone_group_id=None*)

Attempt to create a directory if it doesnt exist.

Parameters

- **directory** string containing the path of the directory to create.
- **keystone_user_id** the system ID of the process running keystone.
- **keystone_group_id** the system ID of the group running keystone.

keystone.common.utils.**flatten_dict**(*d*, *parent_key=''*)

Flatten a nested dictionary.

Converts a dictionary with nested values to a single level flat dictionary, with dotted notation for each key.

keystone.common.utils.**format_url**(*url*, *substitutions*, *silent_keyerror_failures=None*)

Format a user-defined URL with the given substitutions.

Parameters

- **url** (*string*) the URL to be formatted
- **substitutions** (*dict*) the dictionary used for substitution
- **silent_keyerror_failures** (*list*) keys for which we should be silent if there is a KeyError exception on substitution attempt

Returns
a formatted URL

keystone.common.utils.**get_certificate_issuer_dn**(*cert_pem*)

Get issuer DN from the PEM certificate content.

Parameters
cert_pem (*str*) the PEM certificate content

Return type
JSON data for issuer DN

Raises
keystone.exception.ValidationError if the PEM certificate content is invalid

keystone.common.utils.**get_certificate_subject_dn**(*cert_pem*)

Get subject DN from the PEM certificate content.

Parameters
cert_pem (*str*) the PEM certificate content

Return type
JSON data for subject DN

Raises
keystone.exception.ValidationError if the PEM certificate content is invalid

keystone.common.utils.**get_certificate_thumbprint**(*cert_pem*)

Get certificate thumbprint from the PEM certificate content.

Parameters
cert_pem (*str*) the PEM certificate content

Return type
certificate thumbprint

keystone.common.utils.**get_unix_group**(*group=None*)

Get the gid and group name.

This is a convenience utility which accepts a variety of input which might represent a unix group. If successful it returns the gid and name. Valid input is:

string
A string is first considered to be a group name and a lookup is attempted under that name. If no name is found then an attempt is made to convert the string to an integer and perform a lookup as a gid.

int
An integer is interpreted as a gid.

None
None is interpreted to mean use the current processs effective group.

If the input is a valid type but no group is found a KeyError is raised. If the input is not a valid type a TypeError is raised.

Parameters
group (*object*) string, int or None specifying the group to lookup.

Returns
tuple of (gid, name)

keystone.common.utils.**get_unix_user**(*user=None*)

Get the uid and user name.

This is a convenience utility which accepts a variety of input which might represent a unix user. If successful it returns the uid and name. Valid input is:

string
A string is first considered to be a user name and a lookup is attempted under that name. If no name is found then an attempt is made to convert the string to an integer and perform a lookup as a uid.

int
An integer is interpreted as a uid.

None
None is interpreted to mean use the current processs effective user.

If the input is a valid type but no user is found a KeyError is raised. If the input is not a valid type a TypeError is raised.

Parameters
user (*object*) string, int or None specifying the user to lookup.

Returns
tuple of (uid, name)

keystone.common.utils.**hash_access_key**(*access*)

keystone.common.utils.**is_not_url_safe**(*name*)

Check if a string contains any url reserved characters.

keystone.common.utils.**isotime**(*at=None*, *subsecond=False*)

Stringify time in ISO 8601 format.

Python provides a similar instance method for datetime.datetime objects called *isoformat()*. The format of the strings generated by *isoformat()* has a couple of problems:

1) The strings generated by *isotime()* are used in tokens and other public APIs that we cant change without a deprecation period. The strings generated by *isoformat()* are not the same format, so we cant just change to it.

2) The strings generated by *isoformat()* do not include the microseconds if the value happens to be 0. This will likely show up as random failures as parsers may be written to always expect microseconds, and it will parse correctly most of the time.

Parameters

- **at** (*datetime.datetime*) Optional datetime object to return at a string. If not provided, the time when the function was called will be used.
- **subsecond** (*bool*) If true, the returned string will represent microsecond precision, but only precise to the second. For example, a *datetime.datetime(2016, 9, 14, 14, 1, 13, 970223)* will be returned as *2016-09-14T14:01:13.000000Z*.

Returns
A time string represented in ISO 8601 format.

Return type
str

keystone.common.utils.**list_url_unsafe_chars**(*name*)

Return a list of the reserved characters.

keystone.common.utils.**lower_case_hostname**(*url*)

Change the URLs hostname to lowercase.

keystone.common.utils.**nested_contexts**(**contexts*)

keystone.common.utils.**parse_expiration_date**(*expiration_date*)

keystone.common.utils.**remove_standard_port**(*url*)

keystone.common.utils.**resource_uuid**(*value*)

Convert input to valid UUID hex digits.

keystone.common.utils.**setup_remote_pydev_debug**()

Module contents

keystone.conf package

Submodules

keystone.conf.application_credential module

keystone.conf.application_credential.**list_opts**()

keystone.conf.application_credential.**register_opts**(*conf*)

keystone.conf.assignment module

keystone.conf.assignment.**list_opts**()

keystone.conf.assignment.**register_opts**(*conf*)

keystone.conf.auth module

keystone.conf.auth.**list_opts**()

keystone.conf.auth.**register_opts**(*conf*)

keystone.conf.auth.**setup_authentication**(*conf=None*)

Register non-default auth methods (used by extensions, etc).

keystone.conf.catalog module

keystone.conf.catalog.**list_opts**()

keystone.conf.catalog.**register_opts**(*conf*)

keystone.conf.constants module

Constants for use in the keystone.conf package.

These constants are shared by more than one module in the keystone.conf package.

keystone.conf.credential module

keystone.conf.credential.**list_opts**()

keystone.conf.credential.**register_opts**(*conf*)

keystone.conf.default module

keystone.conf.default.**list_opts**()

keystone.conf.default.**register_opts**(*conf*)

keystone.conf.domain_config module

keystone.conf.domain_config.**list_opts**()

keystone.conf.domain_config.**register_opts**(*conf*)

keystone.conf.endpoint_filter module

keystone.conf.endpoint_filter.**list_opts**()

keystone.conf.endpoint_filter.**register_opts**(*conf*)

keystone.conf.endpoint_policy module

keystone.conf.endpoint_policy.**list_opts**()

keystone.conf.endpoint_policy.**register_opts**(*conf*)

keystone.conf.federation module

keystone.conf.federation.**list_opts**()

keystone.conf.federation.**register_opts**(*conf*)

keystone.conf.fernet_receipts module

keystone.conf.fernet_receipts.**list_opts**()

keystone.conf.fernet_receipts.**register_opts**(*conf*)

keystone.conf.fernet_tokens module

keystone.conf.fernet_tokens.**list_opts**()

keystone.conf.fernet_tokens.**register_opts**(*conf*)

keystone.conf.identity module

keystone.conf.identity.**list_opts**()

keystone.conf.identity.**register_opts**(*conf*)

keystone.conf.identity_mapping module

keystone.conf.identity_mapping.**list_opts**()

keystone.conf.identity_mapping.**register_opts**(*conf*)

keystone.conf.jwt_tokens module

keystone.conf.jwt_tokens.**list_opts**()

keystone.conf.jwt_tokens.**register_opts**(*conf*)

keystone.conf.ldap module

keystone.conf.ldap.**list_opts**()

keystone.conf.ldap.**register_opts**(*conf*)

keystone.conf.oauth1 module

keystone.conf.oauth1.**list_opts**()

keystone.conf.oauth1.**register_opts**(*conf*)

keystone.conf.oauth2 module

keystone.conf.oauth2.**list_opts**()

keystone.conf.oauth2.**register_opts**(*conf*)

keystone.conf.opts module

Single point of entry to generate the sample configuration file.

This module collects all the necessary info from the other modules in this package. It is assumed that:

- Every other module in this package has a list_opts function which returns a dict where:
  - The keys are strings which are the group names.
  - The value of each key is a list of config options for that group.
- The conf package doesnt have further packages with config options.
- This module is only used in the context of sample file generation.

keystone.conf.opts.**list_opts**()

keystone.conf.policy module

keystone.conf.policy.**list_opts**()

keystone.conf.policy.**register_opts**(*conf*)

keystone.conf.receipt module

keystone.conf.receipt.**list_opts**()

keystone.conf.receipt.**register_opts**(*conf*)

keystone.conf.resource module

keystone.conf.resource.**list_opts**()

keystone.conf.resource.**register_opts**(*conf*)

keystone.conf.revoke module

keystone.conf.revoke.**list_opts**()

keystone.conf.revoke.**register_opts**(*conf*)

keystone.conf.role module

keystone.conf.role.**list_opts**()

keystone.conf.role.**register_opts**(*conf*)

keystone.conf.saml module

keystone.conf.saml.**list_opts**()

keystone.conf.saml.**register_opts**(*conf*)

keystone.conf.security_compliance module

keystone.conf.security_compliance.**list_opts**()

keystone.conf.security_compliance.**register_opts**(*conf*)

keystone.conf.shadow_users module

keystone.conf.shadow_users.**list_opts**()

keystone.conf.shadow_users.**register_opts**(*conf*)

keystone.conf.token module

keystone.conf.token.**list_opts**()

keystone.conf.token.**register_opts**(*conf*)

keystone.conf.tokenless_auth module

keystone.conf.tokenless_auth.**list_opts**()

keystone.conf.tokenless_auth.**register_opts**(*conf*)

keystone.conf.totp module

keystone.conf.totp.**list_opts**()

keystone.conf.totp.**register_opts**(*conf*)

keystone.conf.trust module

keystone.conf.trust.**list_opts**()

keystone.conf.trust.**register_opts**(*conf*)

keystone.conf.unified_limit module

keystone.conf.unified_limit.**list_opts**()

keystone.conf.unified_limit.**register_opts**(*conf*)

keystone.conf.utils module

keystone.conf.utils.**fmt**(*docstr*)

Format a docstring for use as documentation in sample config.

keystone.conf.wsgi module

keystone.conf.wsgi.**list_opts**()

keystone.conf.wsgi.**register_opts**(*conf*)

Module contents

keystone.conf.**configure**(*conf=None*)

keystone.conf.**set_config_defaults**()

Override all configuration default values for keystone.

keystone.conf.**set_default_for_default_log_levels**()

Set the default for the default_log_levels option for keystone.

Keystone uses some packages that other OpenStack services dont use that do logging. This will set the default_log_levels default level for those packages.

This function needs to be called before CONF().

keystone.conf.**set_external_opts_defaults**()

Update default configuration options for oslo.middleware.

keystone.conf.**setup_logging**()

Set up logging for the keystone package.

keystone.credential package

Subpackages

keystone.credential.backends package

Submodules

keystone.credential.backends.base module

class keystone.credential.backends.base.**CredentialDriverBase**

Bases: object

abstract create_credential(*credential_id*, *credential*)

Create a new credential.

Raises

keystone.exception.Conflict If a duplicate credential exists.

abstract delete_credential(*credential_id*)

Delete an existing credential.

Raises

keystone.exception.CredentialNotFound If credential doesnt exist.

abstract delete_credentials_for_project(*project_id*)

Delete all credentials for a project.

abstract delete_credentials_for_user(*user_id*)

Delete all credentials for a user.

abstract get_credential(*credential_id*)

Get a credential by ID.

Returns

credential_ref

Raises

keystone.exception.CredentialNotFound If credential doesnt exist.

abstract list_credentials(*hints*)

List all credentials.

Parameters

hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns

a list of credential_refs or an empty list.

abstract list_credentials_for_user(*user_id*, *type=None*)

List credentials for a user.

Parameters

- **user_id** ID of a user to filter credentials by.
- **type** type of credentials to filter on.

Returns

a list of credential_refs or an empty list.

abstract update_credential(*credential_id*, *credential*)

Update an existing credential.

Raises

- *keystone.exception.CredentialNotFound* If credential doesnt exist.
- *keystone.exception.Conflict* If a duplicate credential exists.

keystone.credential.backends.sql module

class keystone.credential.backends.sql.**Credential**

Bases: *CredentialDriverBase*

create_credential(*credential_id*, *credential*)

Create a new credential.

Raises

keystone.exception.Conflict If a duplicate credential exists.

delete_credential(*credential_id*)

Delete an existing credential.

Raises

keystone.exception.CredentialNotFound If credential doesnt exist.

delete_credentials_for_project(*project_id*)

Delete all credentials for a project.

delete_credentials_for_user(*user_id*)

Delete all credentials for a user.

get_credential(*credential_id*)

Get a credential by ID.

Returns
credential_ref

Raises
keystone.exception.CredentialNotFound If credential doesnt exist.

`list_credentials`(*hints*)

List all credentials.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
a list of credential_refs or an empty list.

`list_credentials_for_user`(*user_id*, *type=None*)

List credentials for a user.

Parameters

- **user_id** ID of a user to filter credentials by.
- **type** type of credentials to filter on.

Returns
a list of credential_refs or an empty list.

`update_credential`(*credential_id*, *credential*)

Update an existing credential.

Raises

- *keystone.exception.CredentialNotFound* If credential doesnt exist.
- *keystone.exception.Conflict* If a duplicate credential exists.

class keystone.credential.backends.sql.**CredentialModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'user_id', 'project_id', 'encrypted_blob', 'type', 'key_hash']

encrypted_blob

extra

id

key_hash

project_id

type

user_id

Module contents

keystone.credential.providers package

Subpackages

keystone.credential.providers.fernet package

Submodules

keystone.credential.providers.fernet.core module

class keystone.credential.providers.fernet.core.**Provider**

Bases: *Provider*

decrypt(*credential*)

Attempt to decrypt a credential.

Parameters
credential an encrypted credential string

Returns
a decrypted credential

encrypt(*credential*)

Attempt to encrypt a plaintext credential.

Parameters
credential a plaintext representation of a credential

Returns
an encrypted credential

keystone.credential.providers.fernet.core.**get_multi_fernet_keys**()

keystone.credential.providers.fernet.core.**primary_key_hash**(*keys*)

Calculate a hash of the primary key used for encryption.

Module contents

Submodules

keystone.credential.providers.core module

class keystone.credential.providers.core.**Provider**

Bases: `object`

Interface for credential providers that support encryption.

abstract decrypt(*credential*)

Decrypt a credential.

Parameters
credential (*str*) credential to decrypt

Returns
credential str as plaintext

Raises
keystone.exception.CredentialEncryptionError

abstract encrypt(*credential*)

Encrypt a credential.

Parameters
credential (*str*) credential to encrypt

Returns
encrypted credential str

Raises
keystone.exception.CredentialEncryptionError

Module contents

Submodules

keystone.credential.core module

Main entry point into the Credential service.

class keystone.credential.core.**Manager**

Bases: *Manager*

Default pivot point for the Credential backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

create_credential(*credential_id*, *credential*, *initiator=None*)

Create a credential.

delete_credential(*credential_id*, *initiator=None*)

Delete a credential.

delete_credentials_for_project(*project_id*)

Delete all credentials for a project.

delete_credentials_for_user(*user_id*)

Delete all credentials for a user.

driver_namespace: str = 'keystone.credential'

get_credential(*credential_id*)

Return a credential reference.

list_credentials(*hints=None*)

list_credentials_for_user(*user_id*, *type=None*)

update_credential(*credential_id*, *credential*)

Update an existing credential.

keystone.credential.provider module

class keystone.credential.provider.**Manager**

Bases: *Manager*

driver_namespace: str = 'keystone.credential.provider'

keystone.credential.schema module

Module contents

keystone.endpoint_policy package

Subpackages

keystone.endpoint_policy.backends package

Submodules

keystone.endpoint_policy.backends.base module

class keystone.endpoint_policy.backends.base.**EndpointPolicyDriverBase**

Bases: object

Interface description for an Endpoint Policy driver.

abstract check_policy_association(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

Check existence of a policy association.

Parameters

- **policy_id** (*string*) identity of policy that is being associated
- **endpoint_id** (*string*) identity of endpoint to associate
- **service_id** (*string*) identity of the service to associate
- **region_id** (*string*) identity of the region to associate

Raises

keystone.exception.PolicyAssociationNotFound If there is no match for the specified association.

Returns
None

abstract create_policy_association(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

Create a policy association.

Parameters

- **policy_id** (*string*) identity of policy that is being associated
- **endpoint_id** (*string*) identity of endpoint to associate
- **service_id** (*string*) identity of the service to associate
- **region_id** (*string*) identity of the region to associate

Returns
None

There are three types of association permitted:

- Endpoint (in which case service and region must be None)
- Service and region (in which endpoint must be None)
- Service (in which case endpoint and region must be None)

abstract delete_association_by_endpoint(*endpoint_id*)

Remove all the policy associations with the specific endpoint.

Parameters
endpoint_id (*string*) identity of endpoint to check

Returns
None

abstract delete_association_by_policy(*policy_id*)

Remove all the policy associations with the specific policy.

Parameters
policy_id (*string*) identity of endpoint to check

Returns
None

abstract delete_association_by_region(*region_id*)

Remove all the policy associations with the specific region.

Parameters
region_id (*string*) identity of endpoint to check

Returns
None

abstract delete_association_by_service(*service_id*)

Remove all the policy associations with the specific service.

Parameters
service_id (*string*) identity of endpoint to check

Returns
None

abstract delete_policy_association(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

Delete a policy association.

Parameters

- **policy_id** (*string*) identity of policy that is being associated
- **endpoint_id** (*string*) identity of endpoint to associate
- **service_id** (*string*) identity of the service to associate
- **region_id** (*string*) identity of the region to associate

Returns
None

abstract get_policy_association(*endpoint_id=None*, *service_id=None*, *region_id=None*)

Get the policy for an explicit association.

This method is not exposed as a public API, but is used by get_policy_for_endpoint().

Parameters

- **endpoint_id** (*string*) identity of endpoint
- **service_id** (*string*) identity of the service
- **region_id** (*string*) identity of the region

Raises
keystone.exception.PolicyAssociationNotFound If there is no match for the specified association.

Returns
dict containing policy_id (value is a tuple containing only the policy_id)

abstract list_associations_for_policy(*policy_id*)

List the associations for a policy.

This method is not exposed as a public API, but is used by list_endpoints_for_policy().

Parameters
policy_id (*string*) identity of policy

Returns
List of association dicts

keystone.endpoint_policy.backends.sql module

class keystone.endpoint_policy.backends.sql.**EndpointPolicy**

Bases: *EndpointPolicyDriverBase*

check_policy_association(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

Check existence of a policy association.

Parameters

- **policy_id** (*string*) identity of policy that is being associated
- **endpoint_id** (*string*) identity of endpoint to associate
- **service_id** (*string*) identity of the service to associate
- **region_id** (*string*) identity of the region to associate

Raises

keystone.exception.PolicyAssociationNotFound If there is no match for the specified association.

Returns

None

create_policy_association(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

Create a policy association.

Parameters

- **policy_id** (*string*) identity of policy that is being associated
- **endpoint_id** (*string*) identity of endpoint to associate
- **service_id** (*string*) identity of the service to associate
- **region_id** (*string*) identity of the region to associate

Returns

None

There are three types of association permitted:

- Endpoint (in which case service and region must be None)
- Service and region (in which endpoint must be None)
- Service (in which case endpoint and region must be None)

delete_association_by_endpoint(*endpoint_id*)

Remove all the policy associations with the specific endpoint.

Parameters

endpoint_id (*string*) identity of endpoint to check

Returns

None

delete_association_by_policy(*policy_id*)

Remove all the policy associations with the specific policy.

Parameters
policy_id (*string*) identity of endpoint to check

Returns
None

delete_association_by_region(*region_id*)

Remove all the policy associations with the specific region.

Parameters
region_id (*string*) identity of endpoint to check

Returns
None

delete_association_by_service(*service_id*)

Remove all the policy associations with the specific service.

Parameters
service_id (*string*) identity of endpoint to check

Returns
None

delete_policy_association(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

Delete a policy association.

Parameters

- **policy_id** (*string*) identity of policy that is being associated
- **endpoint_id** (*string*) identity of endpoint to associate
- **service_id** (*string*) identity of the service to associate
- **region_id** (*string*) identity of the region to associate

Returns
None

get_policy_association(*endpoint_id=None*, *service_id=None*, *region_id=None*)

Get the policy for an explicit association.

This method is not exposed as a public API, but is used by get_policy_for_endpoint().

Parameters

- **endpoint_id** (*string*) identity of endpoint
- **service_id** (*string*) identity of the service
- **region_id** (*string*) identity of the region

Raises
keystone.exception.PolicyAssociationNotFound If there is no match for the specified association.

Returns
dict containing policy_id (value is a tuple containing only the policy_id)

`list_associations_for_policy`(*policy_id*)

List the associations for a policy.

This method is not exposed as a public API, but is used by list_endpoints_for_policy().

Parameters
policy_id (*string*) identity of policy

Returns
List of association dicts

`class keystone.endpoint_policy.backends.sql.PolicyAssociation`(**args*, ***kwargs*)

Bases: `Base`, `ModelDictMixin`

`attributes = ['policy_id', 'endpoint_id', 'region_id', 'service_id']`

`endpoint_id`

`id`

`policy_id`

`region_id`

`service_id`

`to_dict`()

Return the models attributes as a dictionary.

We override the standard method in order to hide the id column, since this only exists to provide the table with a primary key.

Module contents

Submodules

keystone.endpoint_policy.core module

`class keystone.endpoint_policy.core.Manager`

Bases: *Manager*

Default pivot point for the Endpoint Policy backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

`check_policy_association`(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

`create_policy_association`(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

delete_policy_association(*policy_id*, *endpoint_id=None*, *service_id=None*, *region_id=None*)

driver_namespace: str = 'keystone.endpoint_policy'

get_policy_for_endpoint(*endpoint_id*)

list_endpoints_for_policy(*policy_id*)

Module contents

keystone.federation package

Subpackages

keystone.federation.backends package

Submodules

keystone.federation.backends.base module

class keystone.federation.backends.base.**FederationDriverBase**

Bases: `object`

abstract create_idp(*idp_id*, *idp*)

Create an identity provider.

Parameters

- **idp_id** (*string*) ID of IdP object
- **idp** (*dict*) idp object

Returns
idp ref

Return type
dict

abstract create_mapping(*mapping_id*, *mapping*)

Create a mapping.

Parameters

- **mapping_id** (*string*) ID of mapping object
- **mapping** (*dict*) mapping ref with mapping name

Returns
mapping ref

Return type
dict

abstract create_protocol(*idp_id*, *protocol_id*, *protocol*)

Add an IdP-Protocol configuration.

Parameters

- **idp_id** (*string*) ID of IdP object
- **protocol_id** (*string*) ID of protocol object
- **protocol** (*dict*) protocol object

Raises

keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns

protocol ref

Return type

dict

abstract create_sp(*sp_id*, *sp*)

Create a service provider.

Parameters

- **sp_id** (*string*) id of the service provider
- **sp** (*dict*) service provider object

Returns

service provider ref

Return type

dict

abstract delete_idp(*idp_id*)

Delete an identity provider.

Parameters

idp_id (*string*) ID of IdP object

Raises

keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

abstract delete_mapping(*mapping_id*)

Delete a mapping.

Parameters

mapping_id id of mapping to delete

Returns

None

abstract delete_protocol(*idp_id*, *protocol_id*)

Delete an IdP-Protocol configuration.

Parameters

- **idp_id** (*string*) ID of IdP object

- **protocol_id** (*string*) ID of protocol object

Raises

- *keystone.exception.IdentityProviderNotFound* If the IdP doesnt exist.
- *keystone.exception.FederatedProtocolNotFound* If the federated protocol cannot be found.

abstract delete_sp(*sp_id*)

Delete a service provider.

Parameters
sp_id (*string*) id of the service provider

Raises
keystone.exception.ServiceProviderNotFound If the service provider doesnt exist.

abstract get_enabled_service_providers()

List enabled service providers for Service Catalog.

Service Provider in a catalog contains three attributes: `id`, `auth_url`, `sp_url`, where:

- id is a unique, user defined identifier for service provider object
- auth_url is an authentication URL of remote Keystone
- sp_url a URL accessible at the remote service provider where SAML assertion is transmitted.

Returns
list of dictionaries with enabled service providers

Return type
list of dicts

abstract get_idp(*idp_id*)

Get an identity provider by ID.

Parameters
idp_id (*string*) ID of IdP object

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
idp ref

Return type
dict

abstract get_idp_from_remote_id(*remote_id*)

Get an identity provider by remote ID.

Parameters
remote_id ID of remote IdP

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
idp ref

Return type
dict

abstract get_mapping(*mapping_id*)

Get a mapping, returns the mapping based on mapping_id.

Parameters
mapping_id id of mapping to get

Raises
keystone.exception.MappingNotFound If the mapping cannot be found.

Returns
mapping ref

Return type
dict

abstract get_mapping_from_idp_and_protocol(*idp_id*, *protocol_id*)

Get mapping based on idp_id and protocol_id.

Parameters

- **idp_id** (*string*) id of the identity provider
- **protocol_id** (*string*) id of the protocol

Raises

- *keystone.exception.IdentityProviderNotFound* If the IdP doesnt exist.
- *keystone.exception.FederatedProtocolNotFound* If the federated protocol cannot be found.

Returns
mapping ref

Return type
dict

abstract get_protocol(*idp_id*, *protocol_id*)

Get an IdP-Protocol configuration.

Parameters

- **idp_id** (*string*) ID of IdP object
- **protocol_id** (*string*) ID of protocol object

Raises

- *keystone.exception.IdentityProviderNotFound* If the IdP doesnt exist.

- *keystone.exception.FederatedProtocolNotFound* If the federated protocol cannot be found.

Returns
protocol ref

Return type
dict

abstract get_sp(*sp_id*)

Get a service provider.

Parameters
sp_id (*string*) id of the service provider

Returns
service provider ref

Return type
dict

Raises
keystone.exception.ServiceProviderNotFound If the service provider doesnt exist.

abstract list_idps(*hints*)

List all identity providers.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
list of idp refs

Return type
list of dicts

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

abstract list_mappings()

List all mappings.

Returns
list of mapping refs

Return type
list of dicts

abstract list_protocols(*idp_id*)

List an IdPs supported protocols.

Parameters
idp_id (*string*) ID of IdP object

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
list of protocol ref

Return type
list of dict

abstract list_sps(*hints*)

List all service providers.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
List of service provider ref objects

Return type
list of dicts

Raises
keystone.exception.ServiceProviderNotFound If the SP doesnt exist.

abstract update_idp(*idp_id*, *idp*)

Update an identity provider by ID.

Parameters

- **idp_id** (*string*) ID of IdP object
- **idp** (*dict*) idp object

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
idp ref

Return type
dict

abstract update_mapping(*mapping_id*, *mapping_ref*)

Update a mapping.

Parameters

- **mapping_id** (*string*) id of mapping to update
- **mapping_ref** (*dict*) new mapping ref

Returns
mapping ref

Return type
dict

abstract update_protocol(*idp_id*, *protocol_id*, *protocol*)

Change an IdP-Protocol configuration.

Parameters

- **idp_id** (*string*) ID of IdP object

- **protocol_id** (*string*) ID of protocol object
- **protocol** (*dict*) protocol object

Raises

- *keystone.exception.IdentityProviderNotFound* If the IdP doesnt exist.
- *keystone.exception.FederatedProtocolNotFound* If the federated protocol cannot be found.

Returns
protocol ref

Return type
dict

abstract update_sp(*sp_id*, *sp*)

Update a service provider.

Parameters

- **sp_id** (*string*) id of the service provider
- **sp** (*dict*) service prvider object

Returns
service provider ref

Return type
dict

Raises
keystone.exception.ServiceProviderNotFound If the service provider doesnt exist.

keystone.federation.backends.sql module

class keystone.federation.backends.sql.**Federation**

Bases: *FederationDriverBase*

create_idp(*idp_id*, *idp*)

Create an identity provider.

Parameters

- **idp_id** (*string*) ID of IdP object
- **idp** (*dict*) idp object

Returns
idp ref

Return type
dict

create_mapping(*mapping_id*, *mapping*)

Create a mapping.

Parameters

- **mapping_id** (*string*) ID of mapping object
- **mapping** (*dict*) mapping ref with mapping name

Returns
mapping ref

Return type
dict

create_protocol(*idp_id*, *protocol_id*, *protocol*)

Add an IdP-Protocol configuration.

Parameters

- **idp_id** (*string*) ID of IdP object
- **protocol_id** (*string*) ID of protocol object
- **protocol** (*dict*) protocol object

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
protocol ref

Return type
dict

create_sp(*sp_id*, *sp*)

Create a service provider.

Parameters

- **sp_id** (*string*) id of the service provider
- **sp** (*dict*) service provider object

Returns
service provider ref

Return type
dict

delete_idp(*idp_id*)

Delete an identity provider.

Parameters
idp_id (*string*) ID of IdP object

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

delete_mapping(*mapping_id*)

Delete a mapping.

Parameters
mapping_id id of mapping to delete

Returns
None

`delete_protocol`(*idp_id*, *protocol_id*)

Delete an IdP-Protocol configuration.

Parameters

- **idp_id** (*string*) ID of IdP object
- **protocol_id** (*string*) ID of protocol object

Raises

- *keystone.exception.IdentityProviderNotFound* If the IdP doesnt exist.
- *keystone.exception.FederatedProtocolNotFound* If the federated protocol cannot be found.

`delete_sp`(*sp_id*)

Delete a service provider.

Parameters
sp_id (*string*) id of the service provider

Raises
keystone.exception.ServiceProviderNotFound If the service provider doesnt exist.

`get_enabled_service_providers`()

List enabled service providers for Service Catalog.

Service Provider in a catalog contains three attributes: `id`, `auth_url`, `sp_url`, where:

- id is a unique, user defined identifier for service provider object
- auth_url is an authentication URL of remote Keystone
- sp_url a URL accessible at the remote service provider where SAML assertion is transmitted.

Returns
list of dictionaries with enabled service providers

Return type
list of dicts

`get_idp`(*idp_id*)

Get an identity provider by ID.

Parameters
idp_id (*string*) ID of IdP object

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
idp ref

Return type
dict

`get_idp_from_remote_id`(*remote_id*)

Get an identity provider by remote ID.

Parameters
remote_id ID of remote IdP

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
idp ref

Return type
dict

`get_mapping`(*mapping_id*)

Get a mapping, returns the mapping based on mapping_id.

Parameters
mapping_id id of mapping to get

Raises
keystone.exception.MappingNotFound If the mapping cannot be found.

Returns
mapping ref

Return type
dict

`get_mapping_from_idp_and_protocol`(*idp_id*, *protocol_id*)

Get mapping based on idp_id and protocol_id.

Parameters

- **idp_id** (*string*) id of the identity provider
- **protocol_id** (*string*) id of the protocol

Raises

- *keystone.exception.IdentityProviderNotFound* If the IdP doesnt exist.
- *keystone.exception.FederatedProtocolNotFound* If the federated protocol cannot be found.

Returns
mapping ref

Return type
dict

`get_protocol`(*idp_id*, *protocol_id*)

Get an IdP-Protocol configuration.

Parameters

- **idp_id** (*string*) ID of IdP object
- **protocol_id** (*string*) ID of protocol object

Raises

- *keystone.exception.IdentityProviderNotFound* If the IdP doesnt exist.
- *keystone.exception.FederatedProtocolNotFound* If the federated protocol cannot be found.

Returns
protocol ref

Return type
dict

`get_sp`(*sp_id*)

Get a service provider.

Parameters
sp_id (*string*) id of the service provider

Returns
service provider ref

Return type
dict

Raises
keystone.exception.ServiceProviderNotFound If the service provider doesnt exist.

`list_idps`(*hints=None*)

List all identity providers.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
list of idp refs

Return type
list of dicts

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

`list_mappings`()

List all mappings.

Returns
list of mapping refs

Return type
list of dicts

`list_protocols`(*idp_id*)

List an IdPs supported protocols.

Parameters
idp_id (*string*) ID of IdP object

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
list of protocol ref

Return type
list of dict

`list_sps`(*hints=None*)

List all service providers.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
List of service provider ref objects

Return type
list of dicts

Raises
keystone.exception.ServiceProviderNotFound If the SP doesnt exist.

`update_idp`(*idp_id*, *idp*)

Update an identity provider by ID.

Parameters

- **idp_id** (*string*) ID of IdP object
- **idp** (*dict*) idp object

Raises
keystone.exception.IdentityProviderNotFound If the IdP doesnt exist.

Returns
idp ref

Return type
dict

`update_mapping`(*mapping_id*, *mapping*)

Update a mapping.

Parameters

- **mapping_id** (*string*) id of mapping to update
- **mapping_ref** (*dict*) new mapping ref

Returns
mapping ref

Return type
dict

update_protocol(*idp_id*, *protocol_id*, *protocol*)

Change an IdP-Protocol configuration.

Parameters

- **idp_id** (*string*) ID of IdP object
- **protocol_id** (*string*) ID of protocol object
- **protocol** (*dict*) protocol object

Raises

- *keystone.exception.IdentityProviderNotFound* If the IdP doesnt exist.
- *keystone.exception.FederatedProtocolNotFound* If the federated protocol cannot be found.

Returns
protocol ref

Return type
dict

update_sp(*sp_id*, *sp*)

Update a service provider.

Parameters

- **sp_id** (*string*) id of the service provider
- **sp** (*dict*) service prvider object

Returns
service provider ref

Return type
dict

Raises
keystone.exception.ServiceProviderNotFound If the service provider doesnt exist.

class keystone.federation.backends.sql.**FederationProtocolModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['id', 'idp_id', 'mapping_id', 'remote_id_attribute']

classmethod from_dict(*dictionary*)

Return a model instance from a dictionary.

id

idp_id

mapping_id

mutable_attributes = frozenset({'mapping_id', 'remote_id_attribute'})

remote_id_attribute

to_dict()

Return a dictionary with models attributes.

class keystone.federation.backends.sql.**IdPRemoteIdsModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['idp_id', 'remote_id']

classmethod from_dict(*dictionary*)

Return a model instance from a dictionary.

idp_id

mutable_attributes = frozenset({'idp_id', 'remote_id'})

remote_id

to_dict()

Return a dictionary with models attributes.

class keystone.federation.backends.sql.**IdentityProviderModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['id', 'domain_id', 'enabled', 'description', 'remote_ids', 'authorization_ttl']

authorization_ttl

description

domain_id

enabled

expiring_user_group_memberships

classmethod from_dict(*dictionary*)

Return a model instance from a dictionary.

id

mutable_attributes = frozenset({'authorization_ttl', 'description', 'enabled', 'remote_ids'})

remote_ids

to_dict()

Return a dictionary with models attributes.

class keystone.federation.backends.sql.**MappingModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['id', 'rules', 'schema_version']

classmethod from_dict(*dictionary*)

Return a model instance from a dictionary.

id

rules

schema_version

to_dict()

Return a dictionary with models attributes.

class keystone.federation.backends.sql.ServiceProviderModel(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['auth_url', 'id', 'enabled', 'description', 'relay_state_prefix', 'sp_url']

auth_url

description

enabled

classmethod from_dict(*dictionary*)

Return a model instance from a dictionary.

id

mutable_attributes = frozenset({'auth_url', 'description', 'enabled', 'relay_state_prefix', 'sp_url'})

relay_state_prefix

sp_url

to_dict()

Return a dictionary with models attributes.

Module contents

Submodules

keystone.federation.constants module

keystone.federation.core module

Main entry point into the Federation service.

class keystone.federation.core.**Manager**

Bases: *Manager*

Default pivot point for the Federation backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

create_idp(*idp_id*, *idp*)

create_protocol(*idp_id*, *protocol_id*, *protocol*)

create_sp(*sp_id*, *service_provider*)

delete_idp(*idp_id*)

delete_protocol(*idp_id*, *protocol_id*)

delete_sp(*sp_id*)

driver_namespace: str = 'keystone.federation'

evaluate(*idp_id*, *protocol_id*, *assertion_data*)

get_enabled_service_providers()

List enabled service providers for Service Catalog.

Service Provider in a catalog contains three attributes: `id`, `auth_url`, `sp_url`, where:

- id is a unique, user defined identifier for service provider object
- auth_url is an authentication URL of remote Keystone
- sp_url a URL accessible at the remote service provider where SAML assertion is transmitted.

Returns
list of dictionaries with enabled service providers

Return type
list of dicts

update_protocol(*idp_id*, *protocol_id*, *protocol*)

update_sp(*sp_id*, *service_provider*)

keystone.federation.idp module

class keystone.federation.idp.**ECPGenerator**

Bases: `object`

A class for generating an ECP assertion.

static generate_ecp(*saml_assertion*, *relay_state_prefix*)

class keystone.federation.idp.**MetadataGenerator**

Bases: object

A class for generating SAML IdP Metadata.

generate_metadata()

Generate Identity Provider Metadata.

Generate and format metadata into XML that can be exposed and consumed by a federated Service Provider.

Returns
XML <EntityDescriptor> object.

Raises
keystone.exception.ValidationError If the required config options aren't set.

class keystone.federation.idp.**SAMLGenerator**

Bases: object

A class to generate SAML assertions.

samlize_token(*issuer*, *recipient*, *user*, *user_domain_name*, *roles*, *project*, *project_domain_name*, *groups*, *expires_in=None*)

Convert Keystone attributes to a SAML assertion.

Parameters

- **issuer** (*string*) URL of the issuing party
- **recipient** (*string*) URL of the recipient
- **user** (*string*) User name
- **user_domain_name** (*string*) User Domain name
- **roles** (*list*) List of role names
- **project** (*string*) Project name
- **project_domain_name** (*string*) Project Domain name
- **groups** (*list*) List of strings of user groups and domain name, where strings are serialized dictionaries.
- **expires_in** (*int*) Sets how long the assertion is valid for, in seconds

Returns
XML <Response> object

keystone.federation.schema module

keystone.federation.utils module

Utilities for Federation Extension.

class keystone.federation.utils.**DirectMaps**

Bases: object

An abstraction around the remote matches.

Each match is treated internally as a list.

add(*values*)

Add a matched value to the list of matches.

Parameters
value (*list*) the match to save

class keystone.federation.utils.**RuleProcessor**(*mapping_id*, *rules*)

Bases: object

A class to process assertions and mapping rules.

extract_groups(*groups_by_domain*)

extract_projects(*identity_value*)

normalize_user(*user*, *default_mapping_domain*)

Parse and validate user mapping.

process(*assertion_data*)

Transform assertion to a dictionary.

The dictionary contains mapping of user name and group ids based on mapping rules.

This function will iterate through the mapping rules to find assertions that are valid.

Parameters
assertion_data (*dict*) an assertion containing values from an IdP

Example assertion_data:

```
{
    'Email': 'testacct@example.com',
    'UserName': 'testacct',
    'FirstName': 'Test',
    'LastName': 'Account',
    'orgPersonType': 'Tester'
}
```

Returns
dictionary with user and group_ids

The expected return structure is:

```
{
    'name': 'foobar',
    'group_ids': ['abc123', 'def456'],
    'group_names': [
        {
            'name': 'group_name_1',
            'domain': {
                'name': 'domain1'
            }
        },
        {
            'name': 'group_name_1_1',
            'domain': {
                'name': 'domain1'
            }
        },
        {
            'name': 'group_name_2',
            'domain': {
                'id': 'xyz132'
            }
        }
    ]
}
```

process_group_by_name(*group*, *groups_by_domain*)

class keystone.federation.utils.**RuleProcessorToHonorDomainOption**(*mapping_id*, *rules*)

Bases: *RuleProcessor*

Handles the default domain configured in the attribute mapping.

This rule processor is designed to handle the *domain* attribute configured at the root of the attribute mapping. When this attribute is configured, we should take it as the default one for the attribute mapping, instead of the domain of the IdP. Moreover, we should respect the override to it that can take place at the *groups*, *user*, and *projects* attributes definition.

extract_projects(*identity_value*)

normalize_user(*user*, *default_mapping_domain*)

Parse and validate user mapping.

class keystone.federation.utils.**UserType**

Bases: object

User mapping type.

EPHEMERAL = 'ephemeral'

LOCAL = 'local'

keystone.federation.utils.**assert_enabled_identity_provider**(*federation_api*, *idp_id*)

keystone.federation.utils.**assert_enabled_service_provider_object**(*service_provider*)

keystone.federation.utils.**create_attribute_mapping_rules_processor**(*mapping*)

keystone.federation.utils.**get_assertion_params_from_env**()

keystone.federation.utils.**get_default_attribute_mapping_schema_version**()

keystone.federation.utils.**get_remote_id_parameter**(*idp*, *protocol*)

keystone.federation.utils.**transform_to_group_ids**(*group_names*, *mapping_id*, *identity_api*, *resource_api*)

Transform groups identified by name/domain to their ids.

Function accepts list of groups identified by a name and domain giving a list of group ids in return. A message is logged if the group doesnt exist in the backend.

Example of group_names parameter:

```
[
    {
        "name": "group_name",
        "domain": {
            "id": "domain_id"
        },
    },
    {
        "name": "group_name_2",
        "domain": {
            "name": "domain_name"
        }
    }
]
```

Parameters

- **group_names** (*list*) list of group identified by name and its domain.
- **mapping_id** (*str*) id of the mapping used for mapping assertion into local credentials
- **identity_api** identity_api object
- **resource_api** resource manager object

Returns

generator object with group ids

keystone.federation.utils.**validate_expiration**(*token*)

keystone.federation.utils.**validate_idp**(*idp*, *protocol*, *assertion*)

The IdP providing the assertion should be registered for the mapping.

keystone.federation.utils.**validate_mapped_group_ids**(*group_ids*, *mapping_id*, *identity_api*)

Iterate over group ids and make sure they are present in the backend.

This call is not transactional. :param group_ids: IDs of the groups to be checked :type group_ids: list of str

Parameters

- **mapping_id** (*str*) id of the mapping used for this operation
- **identity_api** (*identity.Manager*) Identity Manager object used for communication with backend

Raises

keystone.exception.MappedGroupNotFound If the group returned by mapping was not found in the backend.

keystone.federation.utils.**validate_mapping_structure**(*ref*)

Module contents

keystone.identity package

Subpackages

keystone.identity.backends package

Subpackages

keystone.identity.backends.ldap package

Submodules

keystone.identity.backends.ldap.common module

class keystone.identity.backends.ldap.common.**AsynchronousMessage**(*message_id*, *connection*, *context_manager*)

Bases: object

A container for handling asynchronous LDAP responses.

Some LDAP APIs, like *search_ext*, are asynchronous and return a message ID when the server successfully initiates the operation. Clients can use this message ID and the original connection to make the request to fetch the results using *result3*.

This object holds the message ID, the original connection, and a callable weak reference Finalizer that cleans up context managers specific to the connection associated to the message ID.

Parameters

- **message_id** The message identifier (str).

- **connection** The connection associated with the message identifier (ldappool.StateConnector).

The *clean* attribute is a callable that cleans up the context manager used to create or return the connection object (weakref.finalize).

class keystone.identity.backends.ldap.common.**BaseLdap**(*conf*)

Bases: object

DEFAULT_EXTRA_ATTR_MAPPING: list[str] = []

DEFAULT_FILTER: str | None = None

DEFAULT_ID_ATTR: str = 'cn'

DEFAULT_OBJECTCLASS: str

DEFAULT_OU: str

DEFAULT_STRUCTURAL_CLASSES: list[str]

NotFound: Type[*Error*]

add_member(*member_dn*, *member_list_dn*)

Add member to the member list.

Parameters

- **member_dn** DN of member to be added.
- **member_list_dn** DN of group to which the member will be added.

Raises

- ***keystone.exception.Conflict*** If the user was already a member.
- **self.NotFound** If the group entry didnt exist.

affirm_unique(*values*)

attribute_ignore: list[str] = []

attribute_options_names: dict[str, str] = {}

create(*values*)

filter_query(*hints*, *query=None*)

Apply filtering to a query.

Parameters

- **hints** contains the list of filters, which may be None, indicating that there are no filters to be applied. If its not None, then any filters satisfied here will be removed so that the caller will know if any filters remain to be applied.
- **query** LDAP query into which to include filters

Returns query

LDAP query, updated with any filters satisfied

get(*object_id*, *ldap_filter=None*)

get_all(*ldap_filter=None*, *hints=None*)

get_by_name(*name*, *ldap_filter=None*)

get_connection(*user=None*, *password=None*, *end_user_auth=False*)

immutable_attrs: list[str] = []

model: Type[Model]

notfound_arg: str | None = None

options_name: str | None = None

tree_dn: str | None = None

update(*object_id*, *values*, *old_obj=None*)

class keystone.identity.backends.ldap.common.**EnabledEmuMixIn**(*conf*)

Bases: *BaseLdap*

Emulates boolean enabled attribute if turned on.

Creates a group holding all enabled objects of this class, all missing objects are considered disabled.

Options:

- $name_enabled_emulation - boolean, on/off
- $name_enabled_emulation_dn - DN of that group, default is cn=enabled_${name}s,${tree_dn}
- $name_enabled_emulation_use_group_config - boolean, on/off

Where ${name}s is the plural of self.options_name (users or tenants), ${tree_dn} is self.tree_dn.

DEFAULT_GROUP_MEMBERS_ARE_IDS = False

DEFAULT_GROUP_OBJECTCLASS = 'groupOfNames'

DEFAULT_MEMBER_ATTRIBUTE = 'member'

create(*values*)

get(*object_id*, *ldap_filter=None*)

get_all(*ldap_filter=None*, *hints=None*)

update(*object_id*, *values*, *old_obj=None*)

class keystone.identity.backends.ldap.common.**KeystoneLDAPHandler**(*conn=None*)

Bases: *LDAPHandler*

Convert data types and perform logging.

This LDAP interface wraps the python-ldap based interfaces. The python-ldap interfaces require string values encoded in UTF-8 with the exception of [1]. The OpenStack logging framework at the time of this writing is not capable of accepting strings encoded in UTF-8, the log functions will throw decoding errors if a non-ascii character appears in a string.

[1] In python-ldap, some fields (DNs, RDNs, attribute names, queries) are represented as text (str on Python 3, unicode on Python 2 when bytes_mode=False). For more details see: http://www.python-ldap.org/en/latest/bytes_mode.html#bytes-mode

Prior to the call Python data types are converted to a string representation as required by the LDAP APIs.

Then logging is performed so we can track what is being sent/received from LDAP. Also the logging filters security sensitive items (i.e. passwords).

Then the string values are encoded into UTF-8.

Then the LDAP API entry point is invoked.

Data returned from the LDAP call is converted back from UTF-8 encoded strings into the Python data type used internally in OpenStack.

add_s(*dn*, *modlist*)

connect(*url*, *page_size=0*, *alias_dereferencing=None*, *use_tls=False*, *tls_cacertfile=None*, *tls_cacertdir=None*, *tls_req_cert=2*, *chase_referrals=None*, *debug_level=None*, *conn_timeout=None*, *use_pool=None*, *pool_size=None*, *pool_retry_max=None*, *pool_retry_delay=None*, *pool_conn_timeout=None*, *pool_conn_lifetime=None*)

get_option(*option*)

modify_s(*dn*, *modlist*)

result3(*msgid=-1*, *all=1*, *timeout=None*, *resp_ctrl_classes=None*)

search_ext(*base*, *scope*, *filterstr='(objectClass=*)'*, *attrlist=None*, *attrsonly=0*, *serverctrls=None*, *clientctrls=None*, *timeout=-1*, *sizelimit=0*)

search_s(*base*, *scope*, *filterstr='(objectClass=*)'*, *attrlist=None*, *attrsonly=0*)

set_option(*option*, *invalue*)

simple_bind_s(*who=''*, *cred=''*, *serverctrls=None*, *clientctrls=None*)

unbind_s()

class keystone.identity.backends.ldap.common.**LDAPHandler**(*conn=None*)

Bases: object

Abstract class which defines methods for a LDAP API provider.

Native Keystone values cannot be passed directly into and from the python-ldap API. Type conversion must occur at the LDAP API boundary, examples of type conversions are:

- booleans map to the strings TRUE and FALSE
- integer values map to their string representation.
- unicode strings are encoded in UTF-8

Note, in python-ldap some fields (DNs, RDNs, attribute names, queries) are represented as text (str on Python 3, unicode on Python 2 when bytes_mode=False). For more details see: http://www.python-ldap.org/en/latest/bytes_mode.html#bytes-mode

In addition to handling type conversions at the API boundary we have the requirement to support more than one LDAP API provider. Currently we have:

- python-ldap, this is the standard LDAP API for Python, it requires access to a live LDAP server.
- Fake LDAP which emulates python-ldap. This is used for testing without requiring a live LDAP server.

To support these requirements we need a layer that performs type conversions and then calls another LDAP API which is configurable (e.g. either python-ldap or the fake emulation).

We have an additional constraint at the time of this writing due to limitations in the logging module. The logging module is not capable of accepting UTF-8 encoded strings, it will throw an encoding exception. Therefore all logging MUST be performed prior to UTF-8 conversion. This means no logging can be performed in the ldap APIs that implement the python-ldap API because those APIs are defined to accept only UTF-8 strings. Thus the layer which performs type conversions must also do the logging. We do the type conversions in two steps, once to convert all Python types to unicode strings, then log, then convert the unicode strings to UTF-8.

There are a variety of ways one could accomplish this, we elect to use a chaining technique whereby instances of this class simply call the next member in the chain via the conn attribute. The chain is constructed by passing in an existing instance of this class as the conn attribute when the class is instantiated.

Here is a brief explanation of why other possible approaches were not used:

subclassing

To perform the wrapping operations in the correct order the type conversion class would have to subclass each of the API providers. This is awkward, doubles the number of classes, and does not scale well. It requires the type conversion class to be aware of all possible API providers.

decorators

Decorators provide an elegant solution to wrap methods and would be an ideal way to perform type conversions before calling the wrapped function and then converting the values returned from the wrapped function. However decorators need to be aware of the method signature, it has to know what input parameters need conversion and how to convert the result. For an API like python-ldap which has a large number of different method signatures it would require a large number of specialized decorators. Experience has shown its very easy to apply the wrong decorator due to the inherent complexity and tendency to cut-n-paste code. Another option is to parameterize the decorator to make it smart. Experience has shown such decorators become insanely complicated and difficult to understand and debug. Also decorators tend to hide whats really going on when a method is called, the operations being performed are not visible when looking at the implemation of a decorated method, this too experience has shown leads to mistakes.

Chaining simplifies both wrapping to perform type conversion as well as the substitution of alternative API providers. One simply creates a new instance of the API interface and insert it at the front of the chain. Type conversions are explicit and obvious.

If a new method needs to be added to the API interface one adds it to the abstract class definition. Should one miss adding the new method to any derivations of the abstract class the code will fail to load and run making it impossible to forget updating all the derived classes.

abstract add_s(*dn*, *modlist*)

abstract connect(*url*, *page_size=0*, *alias_dereferencing=None*, *use_tls=False*, *tls_cacertfile=None*, *tls_cacertdir=None*, *tls_req_cert=2*, *chase_referrals=None*, *debug_level=None*, *conn_timeout=None*, *use_pool=None*, *pool_size=None*, *pool_retry_max=None*, *pool_retry_delay=None*, *pool_conn_timeout=None*, *pool_conn_lifetime=None*)

abstract get_option(*option*)

abstract modify_s(*dn*, *modlist*)

abstract result3(*msgid=-1*, *all=1*, *timeout=None*, *resp_ctrl_classes=None*)

abstract search_ext(*base*, *scope*, *filterstr='(objectClass=*)'*, *attrlist=None*, *attrsonly=0*, *serverctrls=None*, *clientctrls=None*, *timeout=-1*, *sizelimit=0*)

abstract search_s(*base*, *scope*, *filterstr='(objectClass=*)'*, *attrlist=None*, *attrsonly=0*)

abstract set_option(*option*, *invalue*)

abstract simple_bind_s(*who=''*, *cred=''*, *serverctrls=None*, *clientctrls=None*)

abstract unbind_s()

class keystone.identity.backends.ldap.common.**PooledLDAPHandler**(*conn=None*, *use_auth_pool=False*)

Bases: *LDAPHandler*

LDAPHandler implementation which uses pooled connection manager.

Pool specific configuration is defined in [ldap] section. All other LDAP configuration is still used from [ldap] section

Keystone LDAP authentication logic authenticates an end user using its DN and password via LDAP bind to establish supplied password is correct. This can fill up the pool quickly (as pool re-uses existing connection based on its bind data) and would not leave space in pool for connection re-use for other LDAP operations. Now a separate pool can be established for those requests when related flag use_auth_pool is enabled. That pool can have its own size and connection lifetime. Other pool attributes are shared between those pools. If use_pool is disabled, then use_auth_pool does not matter. If use_auth_pool is not enabled, then connection pooling is not used for those LDAP operations.

Note, the python-ldap API requires all string attribute values to be UTF-8 encoded. The KeystoneLDAPHandler enforces this prior to invoking the methods in this class.

Note, in python-ldap some fields (DNs, RDNs, attribute names, queries) are represented as text (str on Python 3, unicode on Python 2 when bytes_mode=False). For more details see: http://www.python-ldap.org/en/latest/bytes_mode.html#bytes-mode

Connector

alias of `StateConnector`

add_s(**args*, ***kwargs*)

auth_pool_prefix = 'auth_pool_'

connect(*url*, *page_size=0*, *alias_dereferencing=None*, *use_tls=False*, *tls_cacertfile=None*, *tls_cacertdir=None*, *tls_req_cert=2*, *chase_referrals=None*, *debug_level=None*, *conn_timeout=None*, *use_pool=None*, *pool_size=None*, *pool_retry_max=None*, *pool_retry_delay=None*, *pool_conn_timeout=None*, *pool_conn_lifetime=None*)

connection_pools: dict = {}

get_option(*option*)

modify_s(**args*, ***kwargs*)

result3(*message*, *all=1*, *timeout=None*, *resp_ctrl_classes=None*)

Wait for and return the result to an asynchronous message.

This method returns the result of an operation previously initiated by one of the LDAP asynchronous operation routines (e.g., *search_ext()*). The *search_ext()* method in python-ldap returns an invocation identifier, or a message ID, upon successful initiation of the operation by the LDAP server.

The *message* is expected to be instance of class *AsynchronousMessage*, which contains the message ID and the connection used to make the original request.

The connection and context manager associated with *search_ext()* are cleaned up when message.clean() is called.

search_ext(*base*, *scope*, *filterstr='(objectClass=*)'*, *attrlist=None*, *attrsonly=0*, *serverctrls=None*, *clientctrls=None*, *timeout=-1*, *sizelimit=0*)

Return an AsynchronousMessage instance, it asynchronous API.

The AsynchronousMessage instance can be safely used in a call to *result3()*.

To work with *result3()* API in predictable manner, the same LDAP connection is needed which originally provided the *msgid*. So, this method wraps the existing connection and *msgid* in a new *AsynchronousMessage* instance. The connection associated with *search_ext()* is released after *result3()* fetches the data associated with *msgid*.

search_s(**args*, ***kwargs*)

set_option(*option*, *invalue*)

simple_bind_s(*who=''*, *cred=''*, *serverctrls=None*, *clientctrls=None*)

unbind_s()

class keystone.identity.backends.ldap.common.**PythonLDAPHandler**(*conn=None*)

Bases: *LDAPHandler*

LDAPHandler implementation which calls the python-ldap API.

Note, the python-ldap API requires all string attribute values to be UTF-8 encoded.

Note, in python-ldap some fields (DNs, RDNs, attribute names, queries) are represented as text (str on Python 3, unicode on Python 2 when bytes_mode=False). For more details see: http://www.python-ldap.org/en/latest/bytes_mode.html#bytes-mode

The KeystoneLDAPHandler enforces this prior to invoking the methods in this class.

add_s(*dn*, *modlist*)

connect(*url*, *page_size=0*, *alias_dereferencing=None*, *use_tls=False*, *tls_cacertfile=None*, *tls_cacertdir=None*, *tls_req_cert=2*, *chase_referrals=None*, *debug_level=None*, *conn_timeout=None*, *use_pool=None*, *pool_size=None*, *pool_retry_max=None*, *pool_retry_delay=None*, *pool_conn_timeout=None*, *pool_conn_lifetime=None*)

get_option(*option*)

modify_s(*dn*, *modlist*)

result3(*msgid=-1*, *all=1*, *timeout=None*, *resp_ctrl_classes=None*)

search_ext(*base*, *scope*, *filterstr='(objectClass=*)'*, *attrlist=None*, *attrsonly=0*, *serverctrls=None*, *clientctrls=None*, *timeout=-1*, *sizelimit=0*)

search_s(*base*, *scope*, *filterstr='(objectClass=*)'*, *attrlist=None*, *attrsonly=0*)

set_option(*option*, *invalue*)

simple_bind_s(*who=''*, *cred=''*, *serverctrls=None*, *clientctrls=None*)

unbind_s()

keystone.identity.backends.ldap.common.**convert_ldap_result**(*ldap_result*)

Convert LDAP search result to Python types used by OpenStack.

Each result tuple is of the form (dn, attrs), where dn is a string containing the DN (distinguished name) of the entry, and attrs is a dictionary containing the attributes associated with the entry. The keys of attrs are strings, and the associated values are lists of strings.

OpenStack wants to use Python types of its choosing. Strings will be unicode, truth values boolean, whole numbers ints, etc. DNs are represented as text in python-ldap by default for Python 3 and when bytes_mode=False for Python 2, and therefore do not require decoding.

Parameters

ldap_result LDAP search result

Returns

list of 2-tuples containing (dn, attrs) where dn is unicode and attrs is a dict whose values are type converted to OpenStack preferred types.

keystone.identity.backends.ldap.common.**dn_startswith**(*descendant_dn*, *dn*)

Return True if and only if the descendant_dn is under the dn.

Parameters

- **descendant_dn** Either a string DN or a DN parsed by ldap.dn.str2dn.
- **dn** Either a string DN or a DN parsed by ldap.dn.str2dn.

keystone.identity.backends.ldap.common.**enabled2py**(*val*)

Similar to ldap2py, only useful for the enabled attribute.

keystone.identity.backends.ldap.common.**filter_entity**(*entity_ref*)

Filter out private items in an entity dict.

Parameters

entity_ref the entity dictionary. The dn field will be removed. dn is used in LDAP, but should not be returned to the user. This value may be modified.

Returns
entity_ref

keystone.identity.backends.ldap.common.**is_ava_value_equal**(*attribute_type*, *val1*, *val2*)

Return True if and only if the AVAs are equal.

When comparing AVAs, the equality matching rule for the attribute type should be taken into consideration. For simplicity, this implementation does a case-insensitive comparison.

Note that this function uses prep_case_insenstive so the limitations of that function apply here.

keystone.identity.backends.ldap.common.**is_dn_equal**(*dn1*, *dn2*)

Return True if and only if the DNs are equal.

Two DNs are equal if theyve got the same number of RDNs and if the RDNs are the same at each position. See RFC4517.

Note that this function uses is_rdn_equal to compare RDNs so the limitations of that function apply here.

Parameters

- **dn1** Either a string DN or a DN parsed by ldap.dn.str2dn.
- **dn2** Either a string DN or a DN parsed by ldap.dn.str2dn.

keystone.identity.backends.ldap.common.**is_rdn_equal**(*rdn1*, *rdn2*)

Return True if and only if the RDNs are equal.

- RDNs must have the same number of AVAs.
- Each AVA of the RDNs must be the equal for the same attribute type. The order isnt significant. Note that an attribute type will only be in one AVA in an RDN, otherwise the DN wouldnt be valid.
- Attribute types arent case sensitive. Note that attribute type comparison is more complicated than implemented. This function only compares case-insentive. The code should handle multiple names for an attribute type (e.g., cn, commonName, and 2.5.4.3 are the same).

Note that this function uses is_ava_value_equal to compare AVAs so the limitations of that function apply here.

keystone.identity.backends.ldap.common.**ldap2py**(*val*)

Convert an LDAP formatted value to Python type used by OpenStack.

Virtually all LDAP values are stored as UTF-8 encoded strings. OpenStack prefers values which are unicode friendly.

Parameters
val LDAP formatted value

Returns
val converted to preferred Python type

keystone.identity.backends.ldap.common.**ldap_scope**(*scope*)

keystone.identity.backends.ldap.common.**parse_deref**(*opt*)

keystone.identity.backends.ldap.common.**parse_tls_cert**(*opt*)

keystone.identity.backends.ldap.common.**prep_case_insensitive**(*value*)

Prepare a string for case-insensitive comparison.

This is defined in RFC4518. For simplicity, all this function does is lowercase all the characters, strip leading and trailing whitespace, and compress sequences of spaces to a single space.

keystone.identity.backends.ldap.common.**py2ldap**(*val*)

Type convert a Python value to a type accepted by LDAP (unicode).

The LDAP API only accepts strings for values therefore convert the values type to a unicode string. A subsequent type conversion will encode the unicode as UTF-8 as required by the python-ldap API, but for now we just want a string representation of the value.

Parameters
val The value to convert to a LDAP string representation

Returns
unicode string representation of value.

keystone.identity.backends.ldap.common.**register_handler**(*prefix*, *handler*)

keystone.identity.backends.ldap.common.**safe_iter**(*attrs*)

keystone.identity.backends.ldap.common.**use_conn_pool**(*func*)

Use this only for connection pool specific ldap API.

This adds connection object to decorated API as next argument after self.

keystone.identity.backends.ldap.common.**utf8_decode**(*value*)

Decode a from UTF-8 into unicode.

If the value is a binary string assume its UTF-8 encoded and decode it into a unicode string. Otherwise convert the value from its type into a unicode string.

Parameters
value value to be returned as unicode

Returns
value as unicode

Raises
UnicodeDecodeError for invalid UTF-8 encoding

keystone.identity.backends.ldap.common.**utf8_encode**(*value*)

Encode a basestring to UTF-8.

If the string is unicode encode it to UTF-8, if the string is str then assume its already encoded. Otherwise raise a TypeError.

Parameters
value A basestring

Returns
UTF-8 encoded version of value

Raises
TypeError If value is not basestring

keystone.identity.backends.ldap.core module

class keystone.identity.backends.ldap.core.**GroupApi**(*conf*)

Bases: *BaseLdap*

DEFAULT_ID_ATTR: str = 'cn'

DEFAULT_MEMBER_ATTRIBUTE = 'member'

DEFAULT_OBJECTCLASS: str = 'groupOfNames'

DEFAULT_OU: str = 'ou=UserGroups'

DEFAULT_STRUCTURAL_CLASSES: list[str] = []

NotFound

alias of *GroupNotFound*

add_user(*user_dn*, *group_id*, *user_id*)

attribute_options_names: dict[str, str] = {'description': 'desc', 'name': 'name'}

create(*values*)

get_all_filtered(*hints*, *query=None*)

get_filtered(*group_id*)

get_filtered_by_name(*group_name*)

immutable_attrs: list[str] = ['name']

list_group_users(*group_id*)

Return a list of user dns which are members of a group.

list_user_groups(*user_dn*)

Return a list of groups for which the user is a member.

list_user_groups_filtered(*user_dn*, *hints*)

Return a filtered list of groups for which the user is a member.

model

alias of *Group*

options_name: ty.Optional[str] = 'group'

update(*group_id*, *values*)

class keystone.identity.backends.ldap.core.**Identity**(*conf=None*)

Bases: *IdentityDriverBase*

add_user_to_group(*user_id*, *group_id*)

Add a user to a group.

Parameters

- **user_id** (*str*) User ID.

- **group_id** (*str*) Group ID.

Raises

- *keystone.exception.UserNotFound* If the user doesnt exist.
- *keystone.exception.GroupNotFound* If the group doesnt exist.

authenticate(*user_id*, *password*)

Authenticate a given user and password.

Parameters

- **user_id** (*str*) User ID
- **password** (*str*) Password

Returns

user. See user schema in *IdentityDriverBase*.

Return type

dict

Raises

AssertionError If user or password is invalid.

change_password(*user_id*, *new_password*)

Self-service password change.

Parameters

- **user_id** (*str*) User ID.
- **new_password** (*str*) New password.

Raises

- *keystone.exception.UserNotFound* If the user doesnt exist.
- **keystone.exception.PasswordValidation** If password fails validation

check_user_in_group(*user_id*, *group_id*)

Check if a user is a member of a group.

Parameters

- **user_id** (*str*) User ID.
- **group_id** (*str*) Group ID.

Raises

- *keystone.exception.NotFound* If the user is not a member of the group.
- *keystone.exception.UserNotFound* If the user doesnt exist.
- *keystone.exception.GroupNotFound* If the group doesnt exist.

create_group(*group_id*, *group*)

Create a new group.

Parameters

- **group_id** (*str*) group ID. The driver can ignore this value.
- **group** (*dict*) group info. See group schema in *IdentityDriverBase*.

Returns
group, matching the group schema.

Return type
dict

Raises
keystone.exception.Conflict If a duplicate group exists.

create_user(*user_id*, *user*)

Create a new user.

Parameters

- **user_id** (*str*) user ID. The driver can ignore this value.
- **user** (*dict*) user info. See user schema in *IdentityDriverBase*.

Returns
user, matching the user schema. The driver should not return the password.

Return type
dict

Raises
keystone.exception.Conflict If a duplicate user exists.

delete_group(*group_id*)

Delete an existing group.

Parameters
group_id (*str*) Group ID.

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

delete_user(*user_id*)

Delete an existing user.

Raises
keystone.exception.UserNotFound If the user doesnt exist.

generates_uuids()

Indicate if Driver generates UUIDs as the local entity ID.

get_group(*group_id*)

Get a group by ID.

Parameters
group_id (*str*) group ID.

Returns
group info. See group schema in *IdentityDriverBase*

Return type
dict

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

get_group_by_name(*group_name*, *domain_id*)

Get a group by name.

Parameters

- **group_name** (*str*) group name.
- **domain_id** (*str*) domain ID.

Returns
group info. See group schema in *IdentityDriverBase*.

Return type
dict

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

get_user(*user_id*)

Get a user by ID.

Parameters
user_id (*str*) User ID.

Returns
user. See user schema in *IdentityDriverBase*.

Return type
dict

Raises
keystone.exception.UserNotFound If the user doesnt exist.

get_user_by_name(*user_name*, *domain_id*)

Get a user by name.

Returns
user_ref

Raises
keystone.exception.UserNotFound If the user doesnt exist.

is_domain_aware()

Indicate if the driver supports domains.

list_groups(*hints*)

List groups in the system.

Parameters
hints (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of group_refs or an empty list. See group schema in *IdentityDriverBase*.

list_groups_for_user(*user_id*, *hints*)

List groups a user is in.

Parameters

- **user_id** (*str*) the user in question
- **hints** (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns

a list of group_refs or an empty list. See group schema in *IdentityDriverBase*.

Raises

keystone.exception.UserNotFound If the user doesnt exist.

list_users(*hints*)

List users in the system.

Parameters

hints (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns

a list of users or an empty list. See user schema in *IdentityDriverBase*.

Return type

list of dict

list_users_in_group(*group_id*, *hints*)

List users in a group.

Parameters

- **group_id** (*str*) the group in question
- **hints** (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns

a list of users or an empty list. See user schema in *IdentityDriverBase*.

Return type

list of dict

Raises

keystone.exception.GroupNotFound If the group doesnt exist.

remove_user_from_group(*user_id*, *group_id*)

Remove a user from a group.

Parameters

- **user_id** (*str*) User ID.
- **group_id** (*str*) Group ID.

Raises

keystone.exception.NotFound If the user is not in the group.

reset_last_active()

Resets null last_active_at values.

This method looks for all users in the database that have a null value for last_updated_at and resets that value to the current time.

unset_default_project_id(*project_id*)

Unset a users default project given a specific project ID.

Parameters
project_id (*str*) project ID

update_group(*group_id*, *group*)

Update an existing group.

Parameters

- **group_id** (*str*) Group ID.
- **group** (*dict*) Group modification. See group schema in *IdentityDriverBase*. Required properties cannot be removed.

Returns
group, matching the group schema.

Return type
dict

Raises

- ***keystone.exception.GroupNotFound*** If the group doesnt exist.
- ***keystone.exception.Conflict*** If a duplicate group exists.

update_user(*user_id*, *user*)

Update an existing user.

Parameters

- **user_id** (*str*) User ID.
- **user** (*dict*) User modification. See user schema in *IdentityDriverBase*. Properties set to None will be removed. Required properties cannot be removed.

Returns
user. See user schema in *IdentityDriverBase*.

Raises

- ***keystone.exception.UserNotFound*** If the user doesnt exist.
- ***keystone.exception.Conflict*** If a duplicate user exists in the same domain.

class keystone.identity.backends.ldap.core.**UserApi**(*conf*)

Bases: *EnabledEmuMixIn*, *BaseLdap*

DEFAULT_ID_ATTR: str = 'cn'

DEFAULT_OBJECTCLASS: str = 'inetOrgPerson'

DEFAULT_OU: str = 'ou=Users'

DEFAULT_STRUCTURAL_CLASSES: list[str] = ['person']

NotFound

alias of *UserNotFound*

attribute_options_names: dict[str, str] = {'default_project_id': 'default_project_id', 'description': 'description', 'email': 'mail', 'enabled': 'enabled', 'name': 'name', 'password': 'pass'}

create(*values*)

filter_attributes(*user*)

get(*user_id*, *ldap_filter=None*)

get_all(*ldap_filter=None*, *hints=None*)

get_all_filtered(*hints*)

get_filtered(*user_id*)

immutable_attrs: list[str] = ['id']

is_user(*dn*)

Return True if the entry is a user.

mask_enabled_attribute(*values*)

model

alias of *User*

options_name: ty.Optional[str] = 'user'

update(*user_id*, *values*, *old_obj=None*)

keystone.identity.backends.ldap.models module

Base model for keystone internal services.

Unless marked otherwise, all fields are strings.

class keystone.identity.backends.ldap.models.**Group**

Bases: *Model*

Group object.

Required keys:

id name domain_id

Optional keys:

description

optional_keys = ('description',)

required_keys = ('id', 'name', 'domain_id')

class keystone.identity.backends.ldap.models.**Model**

Bases: dict

Base model class.

property known_keys

class keystone.identity.backends.ldap.models.**User**

Bases: *Model*

User object.

Required keys:
id name domain_id

Optional keys:
password description email enabled (bool, default True) default_project_id

optional_keys = ('password', 'description', 'email', 'enabled', 'default_project_id')

required_keys = ('id', 'name', 'domain_id')

Module contents

Submodules

keystone.identity.backends.base module

class keystone.identity.backends.base.**IdentityDriverBase**

Bases: object

Interface description for an Identity driver.

The schema for users and groups is different depending on whether the driver is domain aware or not (as returned by self.is_domain_aware()).

If the driver is not domain aware:

- domain_id will be not be included in the user / group passed in to create_user / create_group
- the domain_id should not be returned in user / group refs. Theyll be overwritten.

The password_expires_at in the user schema is a read-only attribute, meaning that it is expected in the response, but not in the request.

User schema (if driver is domain aware):

```
type: object
properties:
    id:
        type: string
    name:
        type: string
```

(continues on next page)

(continued from previous page)

```
    domain_id:
        type: string
    password:
        type: string
    password_expires_at:
        type: datetime
    enabled:
        type: boolean
    default_project_id:
        type: string
required: [id, name, domain_id, enabled]
additionalProperties: True
```

User schema (if driver is not domain aware):

```
type: object
properties:
    id:
        type: string
    name:
        type: string
    password:
        type: string
    password_expires_at:
        type: datetime
    enabled:
        type: boolean
    default_project_id:
        type: string
required: [id, name, enabled]
additionalProperties: True
# Note that domain_id is not allowed as a property
```

Group schema (if driver is domain aware):

```
type: object
properties:
    id:
        type: string
    name:
        type: string
    domain_id:
        type: string
    description:
        type: string
required: [id, name, domain_id]
additionalProperties: True
```

Group schema (if driver is not domain aware):

```
type: object
properties:
    id:
        type: string
    name:
        type: string
    description:
        type: string
required: [id, name]
additionalProperties: True
# Note that domain_id is not allowed as a property
```

abstract add_user_to_group(*user_id*, *group_id*)

Add a user to a group.

Parameters

- **user_id** (*str*) User ID.
- **group_id** (*str*) Group ID.

Raises

- *keystone.exception.UserNotFound* If the user doesnt exist.
- *keystone.exception.GroupNotFound* If the group doesnt exist.

abstract authenticate(*user_id*, *password*)

Authenticate a given user and password.

Parameters

- **user_id** (*str*) User ID
- **password** (*str*) Password

Returns

user. See user schema in *IdentityDriverBase*.

Return type

dict

Raises

AssertionError If user or password is invalid.

abstract change_password(*user_id*, *new_password*)

Self-service password change.

Parameters

- **user_id** (*str*) User ID.
- **new_password** (*str*) New password.

Raises

- *keystone.exception.UserNotFound* If the user doesnt exist.
- **keystone.exception.PasswordValidation** If password fails validation

abstract check_user_in_group(*user_id*, *group_id*)

Check if a user is a member of a group.

Parameters

- **user_id** (*str*) User ID.
- **group_id** (*str*) Group ID.

Raises

- *keystone.exception.NotFound* If the user is not a member of the group.
- *keystone.exception.UserNotFound* If the user doesnt exist.
- *keystone.exception.GroupNotFound* If the group doesnt exist.

abstract create_group(*group_id*, *group*)

Create a new group.

Parameters

- **group_id** (*str*) group ID. The driver can ignore this value.
- **group** (*dict*) group info. See group schema in *IdentityDriverBase*.

Returns

group, matching the group schema.

Return type

dict

Raises

keystone.exception.Conflict If a duplicate group exists.

abstract create_user(*user_id*, *user*)

Create a new user.

Parameters

- **user_id** (*str*) user ID. The driver can ignore this value.
- **user** (*dict*) user info. See user schema in *IdentityDriverBase*.

Returns

user, matching the user schema. The driver should not return the password.

Return type

dict

Raises

keystone.exception.Conflict If a duplicate user exists.

abstract delete_group(*group_id*)

Delete an existing group.

Parameters

group_id (*str*) Group ID.

Raises

keystone.exception.GroupNotFound If the group doesnt exist.

abstract delete_user(*user_id*)

Delete an existing user.

Raises
keystone.exception.UserNotFound If the user doesnt exist.

generates_uuids()

Indicate if Driver generates UUIDs as the local entity ID.

abstract get_group(*group_id*)

Get a group by ID.

Parameters
group_id (*str*) group ID.

Returns
group info. See group schema in *IdentityDriverBase*

Return type
dict

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

abstract get_group_by_name(*group_name*, *domain_id*)

Get a group by name.

Parameters

- **group_name** (*str*) group name.
- **domain_id** (*str*) domain ID.

Returns
group info. See group schema in *IdentityDriverBase*.

Return type
dict

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

abstract get_user(*user_id*)

Get a user by ID.

Parameters
user_id (*str*) User ID.

Returns
user. See user schema in *IdentityDriverBase*.

Return type
dict

Raises
keystone.exception.UserNotFound If the user doesnt exist.

abstract get_user_by_name(*user_name*, *domain_id*)

Get a user by name.

Returns
user_ref

Raises
keystone.exception.UserNotFound If the user doesnt exist.

`is_domain_aware()`

Indicate if the driver supports domains.

`property is_sql`

Indicate if this Driver uses SQL.

`abstract list_groups(hints)`

List groups in the system.

Parameters
hints (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of group_refs or an empty list. See group schema in *IdentityDriverBase*.

`abstract list_groups_for_user(user_id, hints)`

List groups a user is in.

Parameters

- **user_id** (*str*) the user in question
- **hints** (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of group_refs or an empty list. See group schema in *IdentityDriverBase*.

Raises
keystone.exception.UserNotFound If the user doesnt exist.

`abstract list_users(hints)`

List users in the system.

Parameters
hints (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of users or an empty list. See user schema in *IdentityDriverBase*.

Return type
list of dict

`abstract list_users_in_group(group_id, hints)`

List users in a group.

Parameters

- **group_id** (*str*) the group in question

- **hints** (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of users or an empty list. See user schema in *IdentityDriverBase*.

Return type
list of dict

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

property multiple_domains_supported

abstract remove_user_from_group(*user_id*, *group_id*)

Remove a user from a group.

Parameters

- **user_id** (*str*) User ID.
- **group_id** (*str*) Group ID.

Raises
keystone.exception.NotFound If the user is not in the group.

abstract reset_last_active()

Resets null last_active_at values.

This method looks for all users in the database that have a null value for last_updated_at and resets that value to the current time.

abstract unset_default_project_id(*project_id*)

Unset a users default project given a specific project ID.

Parameters
project_id (*str*) project ID

abstract update_group(*group_id*, *group*)

Update an existing group.

Parameters

- **group_id** (*str*) Group ID.
- **group** (*dict*) Group modification. See group schema in *IdentityDriverBase*. Required properties cannot be removed.

Returns
group, matching the group schema.

Return type
dict

Raises

- *keystone.exception.GroupNotFound* If the group doesnt exist.
- *keystone.exception.Conflict* If a duplicate group exists.

abstract update_user(*user_id*, *user*)

Update an existing user.

Parameters

- **user_id** (*str*) User ID.
- **user** (*dict*) User modification. See user schema in *IdentityDriverBase*. Properties set to None will be removed. Required properties cannot be removed.

Returns

user. See user schema in *IdentityDriverBase*.

Raises

- ***keystone.exception.UserNotFound*** If the user doesnt exist.
- ***keystone.exception.Conflict*** If a duplicate user exists in the same domain.

keystone.identity.backends.base.**filter_user**(*user_ref*)

Filter out private items in a user dict.

password, tenants and groups are never returned.

Returns

user_ref

keystone.identity.backends.resource_options module

keystone.identity.backends.resource_options.**register_user_options**()

keystone.identity.backends.sql module

class keystone.identity.backends.sql.**Identity**(*conf=None*)

Bases: *IdentityDriverBase*

add_user_to_group(*user_id*, *group_id*)

Add a user to a group.

Parameters

- **user_id** (*str*) User ID.
- **group_id** (*str*) Group ID.

Raises

- ***keystone.exception.UserNotFound*** If the user doesnt exist.
- ***keystone.exception.GroupNotFound*** If the group doesnt exist.

authenticate(*user_id*, *password*)

Authenticate a given user and password.

Parameters

- **user_id** (*str*) User ID
- **password** (*str*) Password

Returns
user. See user schema in *IdentityDriverBase*.

Return type
dict

Raises
AssertionError If user or password is invalid.

change_password(*user_id*, *new_password*)

Self-service password change.

Parameters

- **user_id** (*str*) User ID.
- **new_password** (*str*) New password.

Raises

- *keystone.exception.UserNotFound* If the user doesnt exist.
- **keystone.exception.PasswordValidation** If password fails validation

check_user_in_group(*user_id*, *group_id*)

Check if a user is a member of a group.

Parameters

- **user_id** (*str*) User ID.
- **group_id** (*str*) Group ID.

Raises

- *keystone.exception.NotFound* If the user is not a member of the group.
- *keystone.exception.UserNotFound* If the user doesnt exist.
- *keystone.exception.GroupNotFound* If the group doesnt exist.

create_group(*group_id*, *group*)

Create a new group.

Parameters

- **group_id** (*str*) group ID. The driver can ignore this value.
- **group** (*dict*) group info. See group schema in *IdentityDriverBase*.

Returns
group, matching the group schema.

Return type
dict

Raises
keystone.exception.Conflict If a duplicate group exists.

create_user(*user_id*, *user*)

Create a new user.

Parameters

- **user_id** (*str*) user ID. The driver can ignore this value.
- **user** (*dict*) user info. See user schema in *IdentityDriverBase*.

Returns
user, matching the user schema. The driver should not return the password.

Return type
dict

Raises
keystone.exception.Conflict If a duplicate user exists.

delete_group(*group_id*)

Delete an existing group.

Parameters
group_id (*str*) Group ID.

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

delete_user(*user_id*)

Delete an existing user.

Raises
keystone.exception.UserNotFound If the user doesnt exist.

get_group(*group_id*)

Get a group by ID.

Parameters
group_id (*str*) group ID.

Returns
group info. See group schema in *IdentityDriverBase*

Return type
dict

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

get_group_by_name(*group_name*, *domain_id*)

Get a group by name.

Parameters

- **group_name** (*str*) group name.
- **domain_id** (*str*) domain ID.

Returns
group info. See group schema in *IdentityDriverBase*.

Return type
dict

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

`get_user`(*user_id*)

Get a user by ID.

Parameters
user_id (*str*) User ID.

Returns
user. See user schema in *IdentityDriverBase*.

Return type
dict

Raises
keystone.exception.UserNotFound If the user doesnt exist.

`get_user_by_name`(*user_name*, *domain_id*)

Get a user by name.

Returns
user_ref

Raises
keystone.exception.UserNotFound If the user doesnt exist.

`property is_sql`

Indicate if this Driver uses SQL.

`list_groups`(*hints*)

List groups in the system.

Parameters
hints (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of group_refs or an empty list. See group schema in *IdentityDriverBase*.

`list_groups_for_user`(*user_id*, *hints*)

List groups a user is in.

Parameters

- **user_id** (*str*) the user in question
- **hints** (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of group_refs or an empty list. See group schema in *IdentityDriverBase*.

Raises
keystone.exception.UserNotFound If the user doesnt exist.

list_users(*hints*)

List users in the system.

Parameters
hints (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of users or an empty list. See user schema in *IdentityDriverBase*.

Return type
list of dict

list_users_in_group(*group_id*, *hints*)

List users in a group.

Parameters

- **group_id** (*str*) the group in question
- **hints** (keystone.common.driver_hints.Hints) filter hints which the driver should implement if at all possible.

Returns
a list of users or an empty list. See user schema in *IdentityDriverBase*.

Return type
list of dict

Raises
keystone.exception.GroupNotFound If the group doesnt exist.

remove_user_from_group(*user_id*, *group_id*)

Remove a user from a group.

Parameters

- **user_id** (*str*) User ID.
- **group_id** (*str*) Group ID.

Raises
keystone.exception.NotFound If the user is not in the group.

reset_last_active()

Resets null last_active_at values.

This method looks for all users in the database that have a null value for last_updated_at and resets that value to the current time.

unset_default_project_id(*project_id*)

Unset a users default project given a specific project ID.

Parameters
project_id (*str*) project ID

update_group(*group_id*, *group*)

Update an existing group.

Parameters

- **group_id** (*str*) Group ID.
- **group** (*dict*) Group modification. See group schema in *IdentityDriverBase*. Required properties cannot be removed.

Returns
group, matching the group schema.

Return type
dict

Raises
- *keystone.exception.GroupNotFound* If the group doesnt exist.
- *keystone.exception.Conflict* If a duplicate group exists.

update_user(*user_id*, *user*)

Update an existing user.

Parameters
- **user_id** (*str*) User ID.
- **user** (*dict*) User modification. See user schema in *IdentityDriverBase*. Properties set to None will be removed. Required properties cannot be removed.

Returns
user. See user schema in *IdentityDriverBase*.

Raises
- *keystone.exception.UserNotFound* If the user doesnt exist.
- *keystone.exception.Conflict* If a duplicate user exists in the same domain.

keystone.identity.backends.sql_model module

class keystone.identity.backends.sql_model.**ExpiringUserGroupMembership**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

Expiring group membership through federation mapping rules.

expired

expires

group_id

idp_id

last_verified

user_id

class keystone.identity.backends.sql_model.**FederatedUser**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['id', 'user_id', 'idp_id', 'protocol_id', 'unique_id', 'display_name']

display_name

id

idp_id

protocol_id

unique_id

user_id

class keystone.identity.backends.sql_model.**Group**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'name', 'domain_id', 'description']

description

domain_id

expiring_user_group_memberships

extra

id

name

class keystone.identity.backends.sql_model.**LocalUser**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['id', 'user_id', 'domain_id', 'name']

domain_id

failed_auth_at

failed_auth_count

id

name

passwords

user_id

class keystone.identity.backends.sql_model.**NonLocalUser**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

SQL data model for nonlocal users (LDAP and custom).

attributes = ['domain_id', 'name', 'user_id']

domain_id

name

user_id

class keystone.identity.backends.sql_model.Password(*args, **kwargs)

Bases: Base, ModelDictMixin

attributes = ['id', 'local_user_id', 'password_hash', 'created_at', 'expires_at']

created_at

created_at_int

expires_at

expires_at_int

id

local_user_id

password_hash

self_service

class keystone.identity.backends.sql_model.User(*args, **kwargs)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'name', 'domain_id', 'password', 'enabled', 'default_project_id', 'password_expires_at']

created_at

default_project_id

domain_id

enabled

Return whether user is enabled or not.

expiring_user_group_memberships

extra

federated_users

classmethod from_dict(*user_dict*)

Override from_dict to remove password_expires_at attribute.

Overriding this method to remove password_expires_at attribute to support update_user and unit tests where password_expires_at inadvertently gets added by calling to_dict followed by from_dict.

Parameters
user_dict User entity dictionary

Returns User
User object

get_resource_option(*option_id*)

id

last_active_at

local_user

name
Return the current user name.

nonlocal_user

password
Return the current password.

property password_created_at
Return when password was created at.

property password_expires_at
Return when password expires at.

property password_is_expired
Return whether password is expired or not.

property password_ref
Return the current password ref.

readonly_attributes = ['id', 'password_expires_at', 'password']

resource_options_registry =
<keystone.common.resource_options.core.ResourceOptionRegistry object>

to_dict(*include_extra_dict=False*)
Return the models attributes as a dictionary.

If include_extra_dict is True, extra attributes are literally included in the resulting dictionary twice, for backwards-compatibility with a broken implementation.

class keystone.identity.backends.sql_model.**UserGroupMembership**(**args*, ***kwargs*)
Bases: Base, ModelDictMixin

Group membership join table.

group_id

user_id

class keystone.identity.backends.sql_model.**UserOption**(*option_id*, *option_value*)
Bases: Base

option_id

option_value

user_id

Module contents

keystone.identity.id_generators package

Submodules

keystone.identity.id_generators.sha256 module

class keystone.identity.id_generators.sha256.**Generator**

Bases: *IDGenerator*

generate_public_ID(*mapping*)

Return a Public ID for the given mapping dict.

Parameters
mapping (*dict*) The items to be hashed.

The ID must be reproducible and no more than 64 chars in length. The ID generated should be independent of the order of the items in the mapping dict.

Module contents

keystone.identity.mapping_backends package

Submodules

keystone.identity.mapping_backends.base module

class keystone.identity.mapping_backends.base.**MappingDriverBase**

Bases: *ProviderAPIMixin*

Interface description for an ID Mapping driver.

abstract create_id_mapping(*local_entity*, *public_id=None*)

Create and store a mapping to a public_id.

Parameters

- **local_entity** (*dict*) Containing the entity domain, local ID and type (user or group).
- **public_id** If specified, this will be the public ID. If this is not specified, a public ID will be generated.

Returns
public ID

abstract delete_id_mapping(*public_id*)

Delete an entry for the given public_id.

Parameters

public_id The public ID for the mapping to be deleted.

The method is silent if no mapping is found.

abstract get_domain_mapping_list(*domain_id*, *entity_type=None*)

Return mappings for the domain.

Parameters

- **domain_id** Domain ID to get mappings for.
- **entity_type** (*String, one of mappings defined in keystone.identity.mapping_backends.mapping.EntityType*) Optional entity_type to get mappings for.

Returns

list of mappings.

abstract get_id_mapping(*public_id*)

Return the local mapping.

Parameters

public_id The public ID for the mapping required.

Returns dict

Containing the entity domain, local ID and type. If no mapping is found, it returns None.

abstract get_public_id(*local_entity*)

Return the public ID for the given local entity.

Parameters

local_entity (*dict*) Containing the entity domain, local ID and type (user or group).

Returns

public ID, or None if no mapping is found.

abstract purge_mappings(*purge_filter*)

Purge selected identity mappings.

Parameters

purge_filter (*dict*) Containing the attributes of the filter that defines which entries to purge. An empty filter means purge all mappings.

keystone.identity.mapping_backends.mapping module

class keystone.identity.mapping_backends.mapping.**EntityType**

Bases: object

GROUP = 'group'

USER = 'user'

keystone.identity.mapping_backends.sql module

class keystone.identity.mapping_backends.sql.**IDMapping**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

domain_id

entity_type

local_id

public_id

class keystone.identity.mapping_backends.sql.**Mapping**

Bases: *MappingDriverBase*

create_id_mapping(*local_entity*, *public_id=None*)

Create and store a mapping to a public_id.

Parameters

- **local_entity** (*dict*) Containing the entity domain, local ID and type (user or group).
- **public_id** If specified, this will be the public ID. If this is not specified, a public ID will be generated.

Returns

public ID

delete_id_mapping(*public_id*)

Delete an entry for the given public_id.

Parameters

public_id The public ID for the mapping to be deleted.

The method is silent if no mapping is found.

get_domain_mapping_list(*domain_id*, *entity_type=None*)

Return mappings for the domain.

Parameters

- **domain_id** Domain ID to get mappings for.
- **entity_type** (*String, one of mappings defined in keystone.identity.mapping_backends.mapping.EntityType*) Optional entity_type to get mappings for.

Returns
list of mappings.

`get_id_mapping`(*public_id*)

Return the local mapping.

Parameters
public_id The public ID for the mapping required.

Returns dict
Containing the entity domain, local ID and type. If no mapping is found, it returns None.

`get_public_id`(*local_entity*)

Return the public ID for the given local entity.

Parameters
local_entity (*dict*) Containing the entity domain, local ID and type (user or group).

Returns
public ID, or None if no mapping is found.

`purge_mappings`(*purge_filter*)

Purge selected identity mappings.

Parameters
purge_filter (*dict*) Containing the attributes of the filter that defines which entries to purge. An empty filter means purge all mappings.

Module contents

keystone.identity.shadow_backends package

Submodules

keystone.identity.shadow_backends.base module

`class keystone.identity.shadow_backends.base.ShadowUsersDriverBase`

Bases: `object`

Interface description for an Shadow Users driver.

`abstract create_federated_object`(*fed_dict*)

Create a new federated object.

Parameters
federated_dict (*dict*) Reference to the federated user

`abstract create_federated_user`(*domain_id*, *federated_dict*, *email=None*)

Create a new user with the federated identity.

Parameters

- **domain_id** The domain ID of the IdP used for the federated user

- **federated_dict** (*dict*) Reference to the federated user
- **email** Federated users email

Returns dict
Containing the user reference

abstract create_nonlocal_user(*user_dict*)

Create a new non-local user.

Parameters
user_dict (*dict*) Reference to the non-local user

Returns dict
Containing the user reference

delete_federated_object(*user_id*)

Delete a users federated objects.

Parameters
user_id Unique identifier of the user

abstract get_federated_objects(*user_id*)

Get all federated objects for a user.

Parameters
user_id Unique identifier of the user

Returns list
Containing the users federated objects

abstract get_federated_user(*idp_id*, *protocol_id*, *unique_id*)

Return the found user for the federated identity.

Parameters

- **idp_id** The identity provider ID
- **protocol_id** The federation protocol ID
- **unique_id** The unique ID for the user

Returns dict
Containing the user reference

abstract get_user(*user_id*)

Return the found user.

Parameters
user_id Unique identifier of the user

Returns dict
Containing the user reference

abstract list_federated_users_info(*hints=None*)

Get the shadow users info with the specified filters.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns list
A list of objects that containing the shadow users reference.

abstract set_last_active_at(*user_id*)

Set the last active at date for the user.

Parameters
user_id Unique identifier of the user

abstract update_federated_user_display_name(*idp_id*, *protocol_id*, *unique_id*, *display_name*)

Update federated users display name if changed.

Parameters

- **idp_id** The identity provider ID
- **protocol_id** The federation protocol ID
- **unique_id** The unique ID for the user
- **display_name** The users display name

keystone.identity.shadow_backends.base.**federated_objects_to_list**(*fed_ref*)

Create a new reformatted federated object list using the one passed in.

When returning federated objects with a user we only need the attributes idp_id, protocol_id, and unique_id. Therefore, we pull these elements out of the fed_ref and create a newly formatted list with the needed information. We simply group each federated objects protocol_ids and unique_ids under the corresponding idp_id.

Returns list
Containing the users federated objects

keystone.identity.shadow_backends.sql module

class keystone.identity.shadow_backends.sql.**ShadowUsers**

Bases: *ShadowUsersDriverBase*

add_user_to_group_expires(*user_id*, *group_id*)

create_federated_object(*fed_dict*)

Create a new federated object.

Parameters
federated_dict (*dict*) Reference to the federated user

create_federated_user(*domain_id*, *federated_dict*, *email=None*)

Create a new user with the federated identity.

Parameters

- **domain_id** The domain ID of the IdP used for the federated user
- **federated_dict** (*dict*) Reference to the federated user
- **email** Federated users email

Returns dict
Containing the user reference

`create_nonlocal_user`(*user_dict*)

Create a new non-local user.

Parameters
user_dict (*dict*) Reference to the non-local user

Returns dict
Containing the user reference

`delete_federated_object`(*user_id*)

Delete a users federated objects.

Parameters
user_id Unique identifier of the user

`delete_user`(*user_id*)

`get_federated_objects`(*user_id*)

Get all federated objects for a user.

Parameters
user_id Unique identifier of the user

Returns list
Containing the users federated objects

`get_federated_user`(*idp_id*, *protocol_id*, *unique_id*)

Return the found user for the federated identity.

Parameters

- **idp_id** The identity provider ID
- **protocol_id** The federation protocol ID
- **unique_id** The unique ID for the user

Returns dict
Containing the user reference

`get_federated_users`(*hints*)

`get_user`(*user_id*)

Return the found user.

Parameters
user_id Unique identifier of the user

Returns dict
Containing the user reference

`list_federated_users_info`(*hints=None*)

Get the shadow users info with the specified filters.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns list
A list of objects that containing the shadow users reference.

`set_last_active_at`(*user_id*)
Set the last active at date for the user.

Parameters
user_id Unique identifier of the user

`update_federated_user_display_name`(*idp_id*, *protocol_id*, *unique_id*, *display_name*)
Update federated users display name if changed.

Parameters

- **idp_id** The identity provider ID
- **protocol_id** The federation protocol ID
- **unique_id** The unique ID for the user
- **display_name** The users display name

Module contents

Submodules

keystone.identity.core module

Main entry point into the Identity service.

`class keystone.identity.core.DomainConfigs`
Bases: *ProviderAPIMixin*, `dict`

Discover, store and provide access to domain specific configs.

The setup_domain_drivers() call will be made via the wrapper from the first call to any driver function handled by this manager.

Domain specific configurations are only supported for the identity backend and the individual configurations are either specified in the resource database or in individual domain configuration files, depending on the setting of the domain_configurations_from_database config option.

The result will be that for each domain with a specific configuration, this class will hold a reference to a ConfigOpts and driver object that the identity manager and driver can use.

`check_config_and_reload_domain_driver_if_required`(*domain_id*)
Check for, and load, any new domain specific config for this domain.

This is only supported for the database-stored domain specific configuration.

When the domain specific drivers were set up, we stored away the specific config for this domain that was available at that time. So we now read the current version and compare. While this might seem somewhat inefficient, the sensitive config call is cached, so should be light weight. More importantly, when the cache timeout is reached, we will get any config that has been updated from any other keystone process.

This cache-timeout approach works for both multi-process and multi-threaded keystone configurations. In multi-threaded configurations, even though we might remove a driver object

(that could be in use by another thread), this wont actually be thrown away until all references to it have been broken. When that other thread is released back and is restarted with another command to process, next time it accesses the driver it will pickup the new one.

configured = False

driver = None

get_domain_conf(*domain_id*)

get_domain_driver(*domain_id*)

lock = <unlocked _thread.lock object>

reload_domain_driver(*domain_id*)

setup_domain_drivers(*standard_driver*, *resource_api*)

class keystone.identity.core.**Manager**

Bases: *Manager*

Default pivot point for the Identity backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

This class also handles the support of domain specific backends, by using the DomainConfigs class. The setup call for DomainConfigs is called from with the @domains_configured wrapper in a lazy loading fashion to get around the fact that we cant satisfy the assignment api it needs from within our __init__() function since the assignment driver is not itself yet initialized.

Each of the identity calls are pre-processed here to choose, based on domain, which of the drivers should be called. The non-domain-specific driver is still in place, and is used if there is no specific driver for the domain in question (or we are not using multiple domain drivers).

Starting with Juno, in order to be able to obtain the domain from just an ID being presented as part of an API call, a public ID to domain and local ID mapping is maintained. This mapping also allows for the local ID of drivers that do not provide simple UUIDs (such as LDAP) to be referenced via a public facing ID. The mapping itself is automatically generated as entities are accessed via the driver.

This mapping is only used when: - the entity is being handled by anything other than the default driver, or - the entity is being handled by the default LDAP driver and backward compatible IDs are not required.

This means that in the standard case of a single SQL backend or the default settings of a single LDAP backend (since backward compatible IDs is set to True by default), no mapping is used. An alternative approach would be to always use the mapping table, but in the cases where we dont need it to make the public and local IDs the same. It is felt that not using the mapping by default is a more prudent way to introduce this functionality.

add_user_to_group(*user_id*, *group_id*, *initiator=None*)

assert_user_enabled(*user_id*, *user=None*)

Assert the user and the users domain are enabled.

:raise AssertionError if the user or the users domain is disabled.

authenticate(*user_id*, *password*)

change_password(*user_id*, *original_password*, *new_password*, *initiator=None*)

check_user_in_group(*user_id*, *group_id*)

create_group(*group_ref*, *initiator=None*)

create_user(*user_ref*, *initiator=None*)

delete_group(*group_id*, *initiator=None*)

delete_user(*user_id*, *initiator=None*)

driver_namespace: str = 'keystone.identity'

get_group(*group_id*)

get_group_by_name(*group_name*, *domain_id*)

get_user(*user_id*)

get_user_by_name(*user_name*, *domain_id*)

list_groups(*domain_scope=None*, *hints=None*)

list_groups_for_user(*user_id*, *hints=None*)

list_users(*domain_scope=None*, *hints=None*)

list_users_in_group(*group_id*, *hints=None*)

remove_user_from_group(*user_id*, *group_id*, *initiator=None*)

shadow_federated_user(*idp_id*, *protocol_id*, *user*, *group_ids=None*)

Map a federated user to a user.

Parameters

- **idp_id** identity provider id
- **protocol_id** protocol id
- **user** User dictionary
- **group_ids** list of group ids to add the user to

Returns

dictionary of the mapped User entity

update_group(*group_id*, *group*, *initiator=None*)

update_user(*user_id*, *user_ref*, *initiator=None*)

class keystone.identity.core.**MappingManager**

Bases: *Manager*

Default pivot point for the ID Mapping backend.

create_id_mapping(*local_entity*, *public_id=None*)

delete_id_mapping(*public_id*)

driver_namespace: str = 'keystone.identity.id_mapping'

get_id_mapping(*public_id*)

get_public_id(*local_entity*)

purge_mappings(*purge_filter*)

class keystone.identity.core.**ShadowUsersManager**

Bases: *Manager*

Default pivot point for the Shadow Users backend.

driver_namespace: str = 'keystone.identity.shadow_users'

keystone.identity.core.**domains_configured**(*f*)

Wrap API calls to lazy load domain configs after init.

This is required since the assignment manager needs to be initialized before this manager, and yet this managers init wants to be able to make assignment calls (to build the domain configs). So instead, we check if the domains have been initialized on entry to each call, and if requires load them,

keystone.identity.core.**exception_translated**(*exception_type*)

Wrap API calls to map to correct exception.

keystone.identity.core.**get_driver**(*namespace*, *driver_name*, **args*)

Get identity driver without initializing.

The method is invoked to be able to introspect domain specific driver looking for additional configuration options required by the driver.

keystone.identity.generator module

ID Generator provider interface.

class keystone.identity.generator.**IDGenerator**

Bases: object

Interface description for an ID Generator provider.

abstract generate_public_ID(*mapping*)

Return a Public ID for the given mapping dict.

Parameters

mapping (*dict*) The items to be hashed.

The ID must be reproducible and no more than 64 chars in length. The ID generated should be independent of the order of the items in the mapping dict.

class keystone.identity.generator.**Manager**

Bases: *Manager*

Default pivot point for the identifier generator backend.

driver_namespace: str = 'keystone.identity.id_generator'

keystone.identity.schema module

Module contents

keystone.limit package

Subpackages

keystone.limit.backends package

Submodules

keystone.limit.backends.base module

class keystone.limit.backends.base.**UnifiedLimitDriverBase**

Bases: object

abstract create_limits(*limits*)

Create new limits.

Parameters
limits a list of dictionaries representing limits to create.

Returns
all the newly created limits.

Raises

- *keystone.exception.Conflict* If a duplicate limit exists.
- *keystone.exception.NoLimitReference* If no reference registered limit exists.

abstract create_registered_limits(*registered_limits*)

Create new registered limits.

Parameters
registered_limits a list of dictionaries representing limits to create.

Returns
all the newly created registered limits.

Raises
keystone.exception.Conflict If a duplicate registered limit exists.

abstract delete_limit(*limit_id*)

Delete an existing limit.

Parameters
limit_id the limit id to delete.

Raises
keystone.exception.LimitNotFound If limit doesnt exist.

abstract delete_limits_for_project(*project_id*)

Delete the existing limits which belong to the specified project.

Parameters
project_id the limits project id.

Returns
a dictionary representing the deleted limits id. Used for cache invalidating.

abstract delete_registered_limit(*registered_limit_id*)

Delete an existing registered limit.

Parameters
registered_limit_id the registered limit id to delete.

Raises
keystone.exception.RegisteredLimitNotFound If registered limit doesnt exist.

abstract get_limit(*limit_id*)

Get a limit.

Parameters
limit_id the limit id to get.

Returns
a dictionary representing a limit reference.

Raises
keystone.exception.LimitNotFound If limit doesnt exist.

abstract get_registered_limit(*registered_limit_id*)

Get a registered limit.

Parameters
registered_limit_id the registered limit id to get.

Returns
a dictionary representing a registered limit reference.

Raises
keystone.exception.RegisteredLimitNotFound If registered limit doesnt exist.

abstract list_limits(*hints*)

List all limits.

Parameters
hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns
a list of dictionaries or an empty list.

abstract list_registered_limits(*hints*)

List all registered limits.

Parameters

hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns

a list of dictionaries or an empty registered limit.

abstract update_limit(*limit_id*, *limit*)

Update existing limits.

Parameters

- **limit_id** the id of the limit.
- **limit** a dict containing the limit attributes to update.

Returns

the updated limit.

Raises

- *keystone.exception.LimitNotFound* If limit doesnt exist.
- *keystone.exception.Conflict* If update to a duplicate limit.

abstract update_registered_limit(*registered_limit_id*, *registered_limit*)

Update existing registered limits.

Parameters

- **registered_limit_id** the id of the registered limit.
- **registered_limit** a dict containing the registered limit attributes to update.

Returns

the updated registered limit.

Raises

- *keystone.exception.RegisteredLimitNotFound* If registered limit doesnt exist.
- *keystone.exception.Conflict* If update to a duplicate registered limit.

keystone.limit.backends.sql module

class keystone.limit.backends.sql.**LimitModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['internal_id', 'id', 'project_id', 'domain_id', 'service_id', 'region_id', 'resource_name', 'resource_limit', 'description', 'registered_limit_id']

description

domain_id

id

internal_id

project_id

region_id

registered_limit

registered_limit_id

resource_limit

resource_name

service_id

to_dict()

Return the models attributes as a dictionary.

class keystone.limit.backends.sql.**RegisteredLimitModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['internal_id', 'id', 'service_id', 'region_id', 'resource_name', 'default_limit', 'description']

default_limit

description

id

internal_id

region_id

resource_name

service_id

to_dict()

Return the models attributes as a dictionary.

class keystone.limit.backends.sql.**UnifiedLimit**

Bases: *UnifiedLimitDriverBase*

create_limits(*limits*)

Create new limits.

Parameters

limits a list of dictionaries representing limits to create.

Returns

all the newly created limits.

Raises

- *keystone.exception.Conflict* If a duplicate limit exists.
- *keystone.exception.NoLimitReference* If no reference registered limit exists.

`create_registered_limits`(*registered_limits*)

Create new registered limits.

Parameters
`registered_limits` a list of dictionaries representing limits to create.

Returns
all the newly created registered limits.

Raises
keystone.exception.Conflict If a duplicate registered limit exists.

`delete_limit`(*limit_id*)

Delete an existing limit.

Parameters
`limit_id` the limit id to delete.

Raises
keystone.exception.LimitNotFound If limit doesnt exist.

`delete_limits_for_project`(*project_id*)

Delete the existing limits which belong to the specified project.

Parameters
`project_id` the limits project id.

Returns
a dictionary representing the deleted limits id. Used for cache invalidating.

`delete_registered_limit`(*registered_limit_id*)

Delete an existing registered limit.

Parameters
`registered_limit_id` the registered limit id to delete.

Raises
keystone.exception.RegisteredLimitNotFound If registered limit doesnt exist.

`get_limit`(*limit_id*)

Get a limit.

Parameters
`limit_id` the limit id to get.

Returns
a dictionary representing a limit reference.

Raises
keystone.exception.LimitNotFound If limit doesnt exist.

`get_registered_limit`(*registered_limit_id*)

Get a registered limit.

Parameters
`registered_limit_id` the registered limit id to get.

Returns
a dictionary representing a registered limit reference.

Raises

keystone.exception.RegisteredLimitNotFound If registered limit doesnt exist.

`list_limits`(*hints*)

List all limits.

Parameters

hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns

a list of dictionaries or an empty list.

`list_registered_limits`(*hints*)

List all registered limits.

Parameters

hints contains the list of filters yet to be satisfied. Any filters satisfied here will be removed so that the caller will know if any filters remain.

Returns

a list of dictionaries or an empty registered limit.

`update_limit`(*limit_id*, *limit*)

Update existing limits.

Parameters

- **limit_id** the id of the limit.
- **limit** a dict containing the limit attributes to update.

Returns

the updated limit.

Raises

- *keystone.exception.LimitNotFound* If limit doesnt exist.
- *keystone.exception.Conflict* If update to a duplicate limit.

`update_registered_limit`(*registered_limit_id*, *registered_limit*)

Update existing registered limits.

Parameters

- **registered_limit_id** the id of the registered limit.
- **registered_limit** a dict containing the registered limit attributes to update.

Returns

the updated registered limit.

Raises

- *keystone.exception.RegisteredLimitNotFound* If registered limit doesnt exist.
- *keystone.exception.Conflict* If update to a duplicate registered limit.

Module contents

keystone.limit.models package

Submodules

keystone.limit.models.base module

class keystone.limit.models.base.**ModelBase**

Bases: object

Interface for a limit model driver.

DESCRIPTION: str

MAX_PROJECT_TREE_DEPTH: int | None = None

NAME: str

check_limit(*limits*)

Check the new creating or updating limits if satisfy the model.

Parameters

limits (*A list of the limits. Each limit is a dictionary reference containing all limit attributes.*) A list of the limit references to be checked.

Raises

keystone.exception.InvalidLimit If any of the input limits doesnt satisfy the limit model.

keystone.limit.models.base.**load_driver**(*driver_name*, **args*)

keystone.limit.models.flat module

class keystone.limit.models.flat.**FlatModel**

Bases: *ModelBase*

DESCRIPTION: str = 'Limit enforcement and validation does not take project hierarchy into consideration.'

MAX_PROJECT_TREE_DEPTH: ty.Optional[int] = None

NAME: str = 'flat'

check_limit(*limits*)

Check the new creating or updating limits if satisfy the model.

Parameters

limits (*A list of the limits. Each limit is a dictionary reference containing all limit attributes.*) A list of the limit references to be checked.

Raises

keystone.exception.InvalidLimit If any of the input limits doesnt satisfy the limit model.

keystone.limit.models.strict_two_level module

class keystone.limit.models.strict_two_level.**StrictTwoLevelModel**

Bases: *ModelBase*

DESCRIPTION: str = 'This model requires project hierarchy never exceeds a depth of two'

MAX_PROJECT_TREE_DEPTH: ty.Optional[int] = 2

NAME: str = 'strict_two_level'

check_limit(*limits*)

Check the input limits satisfy the related project tree or not.

1. Ensure the input is legal.
2. Ensure the input will not break the exist limit tree.

Module contents

Submodules

keystone.limit.core module

class keystone.limit.core.**Manager**

Bases: *Manager*

check_project_depth()

Check if project depth satisfies current enforcement model.

create_limits(*limits*)

create_registered_limits(*registered_limits*)

delete_limit(*limit_id*)

delete_limits_for_project(*project_id*)

delete_registered_limit(*registered_limit_id*)

driver_namespace: str = 'keystone.unified_limit'

get_limit(*limit_id*)

get_model()

Return information of the configured enforcement model.

get_registered_limit(*registered_limit_id*)

list_limits(*hints=None*)

list_registered_limits(*hints=None*)

update_limit(*limit_id*, *limit*)

update_registered_limit(*registered_limit_id*, *registered_limit*)

keystone.limit.schema module

Module contents

keystone.models package

Submodules

keystone.models.receipt_model module

Unified in-memory receipt model.

class keystone.models.receipt_model.**ReceiptModel**

Bases: object

An object that represents a receipt emitted by keystone.

This is a queryable object that other parts of keystone can use to reason about a users receipt.

property expires_at

property issued_at

mint(*receipt_id*, *issued_at*)

Set the id and issued_at attributes of a receipt.

The process of building a Receipt requires setting attributes about the partial authentication context, like user_id and methods for example. Once a Receipt object accurately represents this information it should be minted. Receipt are minted when they get an id attribute and their creation time is recorded.

property required_methods

property user

property user_domain

keystone.models.revoke_model module

class keystone.models.revoke_model.**RevokeEvent**(***kwargs*)

Bases: object

to_dict()

keystone.models.revoke_model.**blank_token_data**(*issued_at*)

keystone.models.revoke_model.**build_token_values**(*token*)

keystone.models.revoke_model.**is_revoked**(*events*, *token_data*)

Check if a token matches a revocation event.

Compare a token against every revocation event. If the token matches an event in the *events* list, the token is revoked. If the token is compared against every item in the list without a match, it is not considered revoked from the *revoke_api*.

Parameters

- **events** a list of RevokeEvent instances
- **token_data** map based on a flattened view of the token. The required fields are *expires_at*,'user_id', *project_id*, *identity_domain_id*, *assignment_domain_id*, *trust_id*, *trustor_id*, *trustee_id* *consumer_id* and *access_token_id*

Returns

True if the token matches an existing revocation event, meaning the token is revoked. False is returned if the token does not match any revocation events, meaning the token is considered valid by the revocation API.

keystone.models.revoke_model.**matches**(*event*, *token_values*)

See if the token matches the revocation event.

A brute force approach to checking. Compare each attribute from the event with the corresponding value from the token. If the event does not have a value for the attribute, a match is still possible. If the event has a value for the attribute, and it does not match the token, no match is possible, so skip the remaining checks.

Parameters

- **event** a RevokeEvent instance
- **token_values** dictionary with set of values taken from the token

Returns

True if the token matches the revocation event, indicating the token has been revoked

keystone.models.token_model module

Unified in-memory token model.

class keystone.models.token_model.**TokenModel**

Bases: object

An object that represents a token emitted by keystone.

This is a queryable object that other parts of keystone can use to reason about a users authentication or authorization.

property access_token

property application_credential

property audit_ids

property domain

property domain_scoped

property expires_at

property issued_at

mint(*token_id*, *issued_at*)

Set the id and issued_at attributes of a token.

The process of building a token requires setting attributes about the authentication and authorization context, like user_id and project_id for example. Once a Token object accurately represents this information it should be minted. Tokens are minted when they get an id attribute and their creation time is recorded.

property oauth_scoped

property project

property project_domain

property project_scoped

property roles

property system_scoped

property trust

property trust_project

property trust_project_domain

property trust_scoped

property trustee

property trustor

property unscoped

property user

property user_domain

Module contents

keystone.oauth1 package

Subpackages

keystone.oauth1.backends package

Submodules

keystone.oauth1.backends.base module

class keystone.oauth1.backends.base.Oauth1DriverBase

Bases: object

Interface description for an OAuth1 driver.

abstract authorize_request_token(*request_token_id*, *user_id*, *role_ids*)

Authorize request token.

Parameters

- **request_token_id** (*string*) the id of the request token, to be authorized
- **user_id** (*string*) the id of the authorizing user
- **role_ids** (*list*) list of role ids to authorize

Returns

verifier

abstract create_access_token(*request_id*, *access_token_duration*)

Create access token.

Parameters

- **request_id** (*string*) the id of the request token, to be deleted
- **access_token_duration** (*string*) duration of an access token

Returns

access_token_ref

abstract create_consumer(*consumer_ref*)

Create consumer.

Parameters

consumer_ref (*dict*) consumer ref with consumer name

Returns
consumer_ref

abstract create_request_token(*consumer_id*, *requested_project*, *request_token_duration*)

Create request token.

Parameters

- **consumer_id** (*string*) the id of the consumer
- **requested_project_id** (*string*) requested project id
- **request_token_duration** (*string*) duration of request token

Returns
request_token_ref

abstract delete_access_token(*user_id*, *access_token_id*)

Delete access token.

Parameters

- **user_id** (*string*) authorizing user id
- **access_token_id** (*string*) access token to delete

Returns
None

abstract delete_consumer(*consumer_id*)

Delete consumer.

Parameters
consumer_id (*string*) id of consumer to get

Returns
None.

abstract get_access_token(*access_token_id*)

Get access token.

Parameters
access_token_id (*string*) the id of the access token

Returns
access_token_ref

abstract get_consumer(*consumer_id*)

Get consumer, returns the consumer id (key) and description.

Parameters
consumer_id (*string*) id of consumer to get

Returns
consumer_ref

abstract get_consumer_with_secret(*consumer_id*)

Like get_consumer(), but also returns consumer secret.

Returned dictionary consumer_ref includes consumer secret. Secrets should only be shared upon consumer creation; the consumer secret is required to verify incoming OAuth requests.

Parameters
consumer_id (*string*) id of consumer to get

Returns
consumer_ref containing consumer secret

abstract get_request_token(*request_token_id*)

Get request token.

Parameters
request_token_id (*string*) the id of the request token

Returns
request_token_ref

abstract list_access_tokens(*user_id*)

List access tokens.

Parameters
user_id (*string*) search for access tokens authorized by given user id

Returns
list of access tokens the user has authorized

abstract list_consumers()

List consumers.

Returns
list of consumers

abstract update_consumer(*consumer_id*, *consumer_ref*)

Update consumer.

Parameters

- **consumer_id** (*string*) id of consumer to update
- **consumer_ref** (*dict*) new consumer ref with consumer name

Returns
consumer_ref

keystone.oauth1.backends.base.**filter_consumer**(*consumer_ref*)

Filter out private items in a consumer dict.

secret is never returned.

Returns
consumer_ref

keystone.oauth1.backends.base.**filter_token**(*access_token_ref*)

Filter out private items in an access token dict.

access_secret is never returned.

Returns
access_token_ref

keystone.oauth1.backends.sql module

class keystone.oauth1.backends.sql.**AccessToken**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

access_secret

attributes = ['id', 'access_secret', 'authorizing_user_id', 'project_id', 'role_ids', 'consumer_id', 'expires_at']

authorizing_user_id

consumer_id

expires_at

classmethod from_dict(*user_dict*)

Return a model instance from a dictionary.

id

project_id

role_ids

to_dict()

Return the models attributes as a dictionary.

class keystone.oauth1.backends.sql.**Consumer**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'description', 'secret']

description

extra

id

secret

class keystone.oauth1.backends.sql.**OAuth1**

Bases: *Oauth1DriverBase*

authorize_request_token(*request_token_id*, *user_id*, *role_ids*)

Authorize request token.

Parameters

- **request_token_id** (*string*) the id of the request token, to be authorized
- **user_id** (*string*) the id of the authorizing user
- **role_ids** (*list*) list of role ids to authorize

Returns

verifier

create_access_token(*request_id*, *access_token_duration*)

Create access token.

Parameters

- **request_id** (*string*) the id of the request token, to be deleted
- **access_token_duration** (*string*) duration of an access token

Returns
access_token_ref

create_consumer(*consumer_ref*)

Create consumer.

Parameters
consumer_ref (*dict*) consumer ref with consumer name

Returns
consumer_ref

create_request_token(*consumer_id*, *requested_project*, *request_token_duration*)

Create request token.

Parameters

- **consumer_id** (*string*) the id of the consumer
- **requested_project_id** (*string*) requested project id
- **request_token_duration** (*string*) duration of request token

Returns
request_token_ref

delete_access_token(*user_id*, *access_token_id*)

Delete access token.

Parameters

- **user_id** (*string*) authorizing user id
- **access_token_id** (*string*) access token to delete

Returns
None

delete_consumer(*consumer_id*)

Delete consumer.

Parameters
consumer_id (*string*) id of consumer to get

Returns
None.

get_access_token(*access_token_id*)

Get access token.

Parameters
access_token_id (*string*) the id of the access token

Returns
access_token_ref

`get_consumer`(*consumer_id*)

Get consumer, returns the consumer id (key) and description.

Parameters
consumer_id (*string*) id of consumer to get

Returns
consumer_ref

`get_consumer_with_secret`(*consumer_id*)

Like get_consumer(), but also returns consumer secret.

Returned dictionary consumer_ref includes consumer secret. Secrets should only be shared upon consumer creation; the consumer secret is required to verify incoming OAuth requests.

Parameters
consumer_id (*string*) id of consumer to get

Returns
consumer_ref containing consumer secret

`get_request_token`(*request_token_id*)

Get request token.

Parameters
request_token_id (*string*) the id of the request token

Returns
request_token_ref

`list_access_tokens`(*user_id*)

List access tokens.

Parameters
user_id (*string*) search for access tokens authorized by given user id

Returns
list of access tokens the user has authorized

`list_consumers`()

List consumers.

Returns
list of consumers

`update_consumer`(*consumer_id*, *consumer_ref*)

Update consumer.

Parameters

- **consumer_id** (*string*) id of consumer to update
- **consumer_ref** (*dict*) new consumer ref with consumer name

Returns
consumer_ref

class keystone.oauth1.backends.sql.**RequestToken**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['id', 'request_secret', 'verifier', 'authorizing_user_id', 'requested_project_id', 'role_ids', 'consumer_id', 'expires_at']

authorizing_user_id

consumer_id

expires_at

classmethod **from_dict**(*user_dict*)

Return a model instance from a dictionary.

id

request_secret

requested_project_id

role_ids

to_dict()

Return the models attributes as a dictionary.

verifier

Module contents

Submodules

keystone.oauth1.core module

Main entry point into the OAuth1 service.

class keystone.oauth1.core.**Manager**

Bases: *Manager*

Default pivot point for the OAuth1 backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

create_access_token(*request_id*, *access_token_duration*, *initiator=None*)

create_consumer(*consumer_ref*, *initiator=None*)

create_request_token(*consumer_id*, *requested_project*, *request_token_duration*, *initiator=None*)

delete_access_token(*user_id*, *access_token_id*, *initiator=None*)

delete_consumer(*consumer_id*, *initiator=None*)

driver_namespace: str = 'keystone.oauth1'

update_consumer(*consumer_id*, *consumer_ref*, *initiator=None*)

class keystone.oauth1.core.**Token**(*key*, *secret*)

Bases: object

set_verifier(*verifier*)

keystone.oauth1.core.**get_oauth_headers**(*headers*)

keystone.oauth1.core.**token_generator**(**args*, ***kwargs*)

keystone.oauth1.core.**validate_oauth_params**(*query_string*)

keystone.oauth1.schema module

keystone.oauth1.validator module

oAuthlib request validator.

class keystone.oauth1.validator.**OAuthValidator**

Bases: *ProviderAPIMixin*, RequestValidator

check_access_token(*access_token*)

Checks that the token contains only safe characters and is no shorter than lower and no longer than upper.

check_client_key(*client_key*)

Check that the client key only contains safe characters and is no shorter than lower and no longer than upper.

check_nonce(*nonce*)

Checks that the nonce only contains only safe characters and is no shorter than lower and no longer than upper.

check_request_token(*request_token*)

Checks that the request token contains only safe characters and is no shorter than lower and no longer than upper.

check_verifier(*verifier*)

Checks that the verifier contains only safe characters and is no shorter than lower and no longer than upper.

property enforce_ssl

get_access_token_secret(*client_key*, *token*, *request*)

Retrieves the shared secret associated with the access token.

Parameters

- **client_key** The client/consumer key.
- **token** The access token string.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns
The token secret as a string.

This method must allow the use of a dummy values and the running time must be roughly equivalent to that of the running time of valid values:

```
# Unlikely to be near constant time as it uses two database
# lookups for a valid client, and only one for an invalid.
from your_datastore import AccessTokenSecret
if AccessTokenSecret.has(client_key):
    return AccessTokenSecret.get((client_key, request_token))
else:
    return 'dummy'

# Aim to mimic number of latency inducing operations no matter
# whether the client is valid or not.
from your_datastore import AccessTokenSecret
return ClientSecret.get((client_key, request_token), 'dummy')
```

Note that the returned key must be in plaintext.

This method is used by

- ResourceEndpoint

get_client_secret(*client_key*, *request*)

Retrieves the client secret associated with the client key.

Parameters

- **client_key** The client/consumer key.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns
The client secret as a string.

This method must allow the use of a dummy client_key value. Fetching the secret using the dummy key must take the same amount of time as fetching a secret for a valid client:

```
# Unlikely to be near constant time as it uses two database
# lookups for a valid client, and only one for an invalid.
from your_datastore import ClientSecret
if ClientSecret.has(client_key):
    return ClientSecret.get(client_key)
else:
    return 'dummy'

# Aim to mimic number of latency inducing operations no matter
# whether the client is valid or not.
from your_datastore import ClientSecret
return ClientSecret.get(client_key, 'dummy')
```

Note that the returned key must be in plaintext.

This method is used by

- AccessTokenEndpoint
- RequestTokenEndpoint
- ResourceEndpoint
- SignatureOnlyEndpoint

get_default_realms(*client_key*, *request*)

Get the default realms for a client.

Parameters

- **client_key** The client/consumer key.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

The list of default realms associated with the client.

The list of default realms will be set during client registration and is outside the scope of OAuthLib.

This method is used by

- RequestTokenEndpoint

get_realms(*token*, *request*)

Get realms associated with a request token.

Parameters

- **token** The request token string.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

The list of realms associated with the request token.

This method is used by

- AuthorizationEndpoint
- AccessTokenEndpoint

get_redirect_uri(*token*, *request*)

Get the redirect URI associated with a request token.

Parameters

- **token** The request token string.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

The redirect URI associated with the request token.

It may be desirable to return a custom URI if the redirect is set to oob. In this case, the user will be redirected to the returned URI and at that endpoint the verifier can be displayed.

This method is used by

- AuthorizationEndpoint

get_request_token_secret(*client_key*, *token*, *request*)

Retrieves the shared secret associated with the request token.

Parameters

- **client_key** The client/consumer key.
- **token** The request token string.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

The token secret as a string.

This method must allow the use of a dummy values and the running time must be roughly equivalent to that of the running time of valid values:

```
# Unlikely to be near constant time as it uses two database
# lookups for a valid client, and only one for an invalid.
from your_datastore import RequestTokenSecret
if RequestTokenSecret.has(client_key):
    return RequestTokenSecret.get((client_key, request_token))
else:
    return 'dummy'

# Aim to mimic number of latency inducing operations no matter
# whether the client is valid or not.
from your_datastore import RequestTokenSecret
return ClientSecret.get((client_key, request_token), 'dummy')
```

Note that the returned key must be in plaintext.

This method is used by

- AccessTokenEndpoint

get_rsa_key(*client_key*, *request*)

Retrieves a previously stored client provided RSA key.

Parameters

- **client_key** The client/consumer key.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

The rsa public key as a string.

This method must allow the use of a dummy client_key value. Fetching the rsa key using the dummy key must take the same amount of time as fetching a key for a valid client. The dummy key must also be of the same bit length as client keys.

Note that the key must be returned in plaintext.

This method is used by

- AccessTokenEndpoint
- RequestTokenEndpoint
- ResourceEndpoint

- SignatureOnlyEndpoint

invalidate_request_token(*client_key*, *request_token*, *request*)

Invalidate a used request token.

Parameters

- **client_key** The client/consumer key.
- **request_token** The request token string.
- **request** An oauthlib.common.Request object.

Returns
None

Per Section 2.3 of the spec:

The server MUST () ensure that the temporary credentials have not expired or been used before.

This method should ensure that provided token wont validate anymore. It can be simply removing RequestToken from storage or setting specific flag that makes it invalid (note that such flag should be also validated during request token validation).

This method is used by

- AccessTokenEndpoint

property safe_characters

save_access_token(*token*, *request*)

Save an OAuth1 access token.

Parameters

- **token** A dict with token credentials.
- **request** (*oauthlib.common.Request*) OAuthlib request.

The token dictionary will at minimum include

- `oauth_token` the access token string.
- `oauth_token_secret` the token specific secret used in signing.
- `oauth_authorized_realms` a space separated list of realms.

Client key can be obtained from `request.client_key`.

The list of realms (not joined string) can be obtained from `request.realm`.

This method is used by

- AccessTokenEndpoint

save_request_token(*token*, *request*)

Save an OAuth1 request token.

Parameters

- **token** A dict with token credentials.
- **request** (*oauthlib.common.Request*) OAuthlib request.

The token dictionary will at minimum include

- `oauth_token` the request token string.
- `oauth_token_secret` the token specific secret used in signing.
- `oauth_callback_confirmed` the string `true`.

Client key can be obtained from `request.client_key`.

This method is used by

- RequestTokenEndpoint

save_verifier(*token*, *verifier*, *request*)

Associate an authorization verifier with a request token.

Parameters

- **token** A request token string.
- **verifier** A dictionary containing the oauth_verifier and oauth_token
- **request** An oauthlib.common.Request object.

We need to associate verifiers with tokens for validation during the access token request.

Note that unlike save_x_token token here is the `oauth_token` token string from the request token saved previously.

This method is used by

- AuthorizationEndpoint

validate_access_token(*client_key*, *token*, *request*)

Validates that supplied access token is registered and valid.

Parameters

- **client_key** The client/consumer key.
- **token** The access token string.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

True or False

Note that if the dummy access token is supplied it should validate in the same or nearly the same amount of time as a valid one.

Ensure latency inducing tasks are mimiced even for dummy clients. For example, use:

```
from your_datastore import AccessToken
try:
    return AccessToken.exists(client_key, access_token)
except DoesNotExist:
    return False
```

Rather than:

```
from your_datastore import AccessToken
if access_token == self.dummy_access_token:
    return False
else:
    return AccessToken.exists(client_key, access_token)
```

This method is used by

- ResourceEndpoint

validate_client_key(*client_key*, *request*)

Validates that supplied client key is a registered and valid client.

Parameters

- **client_key** The client/consumer key.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

True or False

Note that if the dummy client is supplied it should validate in same or nearly the same amount of time as a valid one.

Ensure latency inducing tasks are mimiced even for dummy clients. For example, use:

```
from your_datastore import Client
try:
    return Client.exists(client_key, access_token)
except DoesNotExist:
    return False
```

Rather than:

```
from your_datastore import Client
if access_token == self.dummy_access_token:
    return False
else:
    return Client.exists(client_key, access_token)
```

This method is used by

- AccessTokenEndpoint
- RequestTokenEndpoint
- ResourceEndpoint
- SignatureOnlyEndpoint

validate_realms(*client_key*, *token*, *request*, *uri=None*, *realms=None*)

Validates access to the request realm.

Parameters

- **client_key** The client/consumer key.
- **token** A request token string.

- **request** (*oauthlib.common.Request*) OAuthlib request.
- **uri** The URI the realms is protecting.
- **realms** A list of realms that must have been granted to the access token.

Returns
True or False

How providers choose to use the realm parameter is outside the OAuth specification but it is commonly used to restrict access to a subset of protected resources such as photos.

realms is a convenience parameter which can be used to provide a per view method pre-defined list of allowed realms.

Can be as simple as:

```
from your_datastore import RequestToken
request_token = RequestToken.get(token, None)

if not request_token:
    return False
return set(request_token.realms).issuperset(set(realms))
```

This method is used by

- ResourceEndpoint

validate_redirect_uri(*client_key*, *redirect_uri*, *request*)

Validates the client supplied redirection URI.

Parameters

- **client_key** The client/consumer key.
- **redirect_uri** The URI the client which to redirect back to after authorization is successful.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns
True or False

It is highly recommended that OAuth providers require their clients to register all redirection URIs prior to using them in requests and register them as absolute URIs. See CWE-601 for more information about open redirection attacks.

By requiring registration of all redirection URIs it should be straightforward for the provider to verify whether the supplied redirect_uri is valid or not.

Alternatively per Section 2.1 of the spec:

If the client is unable to receive callbacks or a callback URI has been established via other means, the parameter value MUST be set to oob (case sensitive), to indicate an out-of-band configuration.

This method is used by

- RequestTokenEndpoint

validate_request_token(*client_key*, *token*, *request*)

Validates that supplied request token is registered and valid.

Parameters

- **client_key** The client/consumer key.
- **token** The request token string.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

True or False

Note that if the dummy request_token is supplied it should validate in the same nearly the same amount of time as a valid one.

Ensure latency inducing tasks are mimiced even for dummy clients. For example, use:

```
from your_datastore import RequestToken
try:
    return RequestToken.exists(client_key, access_token)
except DoesNotExist:
    return False
```

Rather than:

```
from your_datastore import RequestToken
if access_token == self.dummy_access_token:
    return False
else:
    return RequestToken.exists(client_key, access_token)
```

This method is used by

- AccessTokenEndpoint

validate_requested_realms(*client_key*, *realms*, *request*)

Validates that the client may request access to the realm.

Parameters

- **client_key** The client/consumer key.
- **realms** The list of realms that client is requesting access to.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

True or False

This method is invoked when obtaining a request token and should tie a realm to the request token and after user authorization this realm restriction should transfer to the access token.

This method is used by

- RequestTokenEndpoint

validate_timestamp_and_nonce(*client_key*, *timestamp*, *nonce*, *request*, *request_token=None*, *access_token=None*)

Validates that the nonce has not been used before.

Parameters

- **client_key** The client/consumer key.
- **timestamp** The `oauth_timestamp` parameter.
- **nonce** The `oauth_nonce` parameter.
- **request_token** Request token string, if any.
- **access_token** Access token string, if any.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns

True or False

Per Section 3.3 of the spec.

A nonce is a random string, uniquely generated by the client to allow the server to verify that a request has never been made before and helps prevent replay attacks when requests are made over a non-secure channel. The nonce value MUST be unique across all requests with the same timestamp, client credentials, and token combinations.

One of the first validation checks that will be made is for the validity of the nonce and timestamp, which are associated with a client key and possibly a token. If invalid then immediately fail the request by returning False. If the nonce/timestamp pair has been used before and you may just have detected a replay attack. Therefore it is an essential part of OAuth security that you not allow nonce/timestamp reuse. Note that this validation check is done before checking the validity of the client and token.:

```
nonces_and_timestamps_database = [
   (u'foo', 1234567890, u'rannoMstrInghere', u'bar')
]

def validate_timestamp_and_nonce(self, client_key, timestamp, nonce,
   request_token=None, access_token=None):

   return ((client_key, timestamp, nonce, request_token or access_
↪token)
           not in self.nonces_and_timestamps_database)
```

This method is used by

- AccessTokenEndpoint
- RequestTokenEndpoint
- ResourceEndpoint
- SignatureOnlyEndpoint

validate_verifier(*client_key*, *token*, *verifier*, *request*)

Validates a verification code.

Parameters

- **client_key** The client/consumer key.
- **token** A request token string.
- **verifier** The authorization verifier string.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns
True or False

OAuth providers issue a verification code to clients after the resource owner authorizes access. This code is used by the client to obtain token credentials and the provider must verify that the verifier is valid and associated with the client as well as the resource owner.

Verifier validation should be done in near constant time (to avoid verifier enumeration). To achieve this we need a constant time string comparison which is provided by OAuthLib in `oauthlib.common.safe_string_equals`:

```
from your_datastore import Verifier
correct_verifier = Verifier.get(client_key, request_token)
from oauthlib.common import safe_string_equals
return safe_string_equals(verifier, correct_verifier)
```

This method is used by

- AccessTokenEndpoint

verify_realms(*token*, *realms*, *request*)

Verify authorized realms to see if they match those given to token.

Parameters

- **token** An access token string.
- **realms** A list of realms the client attempts to access.
- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns
True or False

This prevents the list of authorized realms sent by the client during the authorization step to be altered to include realms outside what was bound with the request token.

Can be as simple as:

```
valid_realms = self.get_realms(token)
return all((r in valid_realms for r in realms))
```

This method is used by

- AuthorizationEndpoint

verify_request_token(*token*, *request*)

Verify that the given OAuth1 request token is valid.

Parameters

- **token** A request token string.

- **request** (*oauthlib.common.Request*) OAuthlib request.

Returns
True or False

This method is used only in AuthorizationEndpoint to check whether the oauth_token given in the authorization URL is valid or not. This request is not signed and thus similar `validate_request_token` method can not be used.

This method is used by

- AuthorizationEndpoint

Module contents

keystone.oauth2 package

Submodules

keystone.oauth2.handlers module

keystone.oauth2.handlers.**build_response**(*error*)

Module contents

keystone.policy package

Subpackages

keystone.policy.backends package

Submodules

keystone.policy.backends.base module

class keystone.policy.backends.base.**PolicyDriverBase**

Bases: object

abstract create_policy(*policy_id*, *policy*)

Store a policy blob.

Raises
keystone.exception.Conflict If a duplicate policy exists.

abstract delete_policy(*policy_id*)

Remove a policy blob.

Raises
keystone.exception.PolicyNotFound If the policy doesnt exist.

abstract enforce(*context*, *credentials*, *action*, *target*)

Verify that a user is authorized to perform action.

For more information on a full implementation of this see: *keystone.policy.backends.rules.Policy.enforce*

abstract get_policy(*policy_id*)

Retrieve a specific policy blob.

Raises

keystone.exception.PolicyNotFound If the policy doesnt exist.

abstract list_policies()

List all policies.

abstract update_policy(*policy_id*, *policy*)

Update a policy blob.

Raises

keystone.exception.PolicyNotFound If the policy doesnt exist.

keystone.policy.backends.rules module

Policy engine for keystone.

class keystone.policy.backends.rules.**Policy**

Bases: *PolicyDriverBase*

create_policy(*policy_id*, *policy*)

Store a policy blob.

Raises

keystone.exception.Conflict If a duplicate policy exists.

delete_policy(*policy_id*)

Remove a policy blob.

Raises

keystone.exception.PolicyNotFound If the policy doesnt exist.

enforce(*credentials*, *action*, *target*)

Verify that a user is authorized to perform action.

For more information on a full implementation of this see: *keystone.policy.backends.rules.Policy.enforce*

get_policy(*policy_id*)

Retrieve a specific policy blob.

Raises

keystone.exception.PolicyNotFound If the policy doesnt exist.

list_policies()

List all policies.

update_policy(*policy_id*, *policy*)

Update a policy blob.

Raises

keystone.exception.PolicyNotFound If the policy doesnt exist.

keystone.policy.backends.sql module

class keystone.policy.backends.sql.**Policy**

Bases: *Policy*

create_policy(*policy_id*, *policy*)

Store a policy blob.

Raises

keystone.exception.Conflict If a duplicate policy exists.

delete_policy(*policy_id*)

Remove a policy blob.

Raises

keystone.exception.PolicyNotFound If the policy doesnt exist.

get_policy(*policy_id*)

Retrieve a specific policy blob.

Raises

keystone.exception.PolicyNotFound If the policy doesnt exist.

list_policies()

List all policies.

update_policy(*policy_id*, *policy*)

Update a policy blob.

Raises

keystone.exception.PolicyNotFound If the policy doesnt exist.

class keystone.policy.backends.sql.**PolicyModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'blob', 'type']

blob

extra

id

type

Module contents

Submodules

keystone.policy.core module

Main entry point into the Policy service.

class keystone.policy.core.**Manager**

Bases: *Manager*

Default pivot point for the Policy backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

create_policy(*policy_id*, *policy*, *initiator=None*)

delete_policy(*policy_id*, *initiator=None*)

driver_namespace: str = 'keystone.policy'

get_policy(*policy_id*)

list_policies(*hints=None*)

update_policy(*policy_id*, *policy*, *initiator=None*)

keystone.policy.schema module

Module contents

keystone.receipt package

Subpackages

keystone.receipt.providers package

Subpackages

keystone.receipt.providers.fernet package

Submodules

keystone.receipt.providers.fernet.core module

class keystone.receipt.providers.fernet.core.**Provider**(**args*, ***kwargs*)

Bases: *Provider*

generate_id_and_issued_at(*receipt*)

Generate a receipt based on the information provided.

Parameters
receipt (*keystone.models.receipt.ReceiptModel*) A receipt object containing information about the authorization context of the request.

Returns
tuple containing an ID for the receipt and the issued at time of the receipt (receipt_id, issued_at).

validate_receipt(*receipt_id*)

Validate a given receipt by its ID and return the receipt_data.

Parameters
receipt_id (*str*) the unique ID of the receipt

Returns
receipt data as a tuple in the form of:

(user_id, methods, issued_at, expires_at)

`user_id` is the unique ID of the user as a string `methods` a list of authentication methods used to obtain the receipt `issued_at` a datetime object of when the receipt was minted `expires_at` a datetime object of when the receipt expires

Raises
keystone.exception.ReceiptNotFound when receipt doesnt exist.

Module contents

class keystone.receipt.providers.fernet.**Provider**(**args*, ***kwargs*)

Bases: *Provider*

generate_id_and_issued_at(*receipt*)

Generate a receipt based on the information provided.

Parameters
receipt (*keystone.models.receipt.ReceiptModel*) A receipt object containing information about the authorization context of the request.

Returns
tuple containing an ID for the receipt and the issued at time of the receipt (receipt_id, issued_at).

validate_receipt(*receipt_id*)

Validate a given receipt by its ID and return the receipt_data.

Parameters
receipt_id (*str*) the unique ID of the receipt

Returns
receipt data as a tuple in the form of:

(user_id, methods, issued_at, expires_at)

user_id is the unique ID of the user as a string methods a list of authentication methods used to obtain the receipt issued_at a datetime object of when the receipt was minted expires_at a datetime object of when the receipt expires

Raises
keystone.exception.ReceiptNotFound when receipt doesnt exist.

Submodules

keystone.receipt.providers.base module

class keystone.receipt.providers.base.**Provider**

Bases: object

Interface description for a Receipt provider.

abstract generate_id_and_issued_at(*receipt*)

Generate a receipt based on the information provided.

Parameters
receipt (*keystone.models.receipt.ReceiptModel*) A receipt object containing information about the authorization context of the request.

Returns
tuple containing an ID for the receipt and the issued at time of the receipt (receipt_id, issued_at).

abstract validate_receipt(*receipt_id*)

Validate a given receipt by its ID and return the receipt_data.

Parameters
receipt_id (*str*) the unique ID of the receipt

Returns
receipt data as a tuple in the form of:

(user_id, methods, issued_at, expires_at)

user_id is the unique ID of the user as a string methods a list of authentication methods used to obtain the receipt issued_at a datetime object of when the receipt was minted expires_at a datetime object of when the receipt expires

Raises
keystone.exception.ReceiptNotFound when receipt doesnt exist.

Module contents

Submodules

keystone.receipt.handlers module

keystone.receipt.handlers.**build_receipt**(*mfa_error*)

keystone.receipt.handlers.**extract_receipt**(*auth_context*)

keystone.receipt.provider module

Receipt provider interface.

class keystone.receipt.provider.**Manager**

Bases: *Manager*

Default pivot point for the receipt provider backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

driver_namespace: str = 'keystone.receipt.provider'

issue_receipt(*user_id*, *method_names*, *expires_at=None*)

validate_receipt(*receipt_id*, *window_seconds=0*)

keystone.receipt.provider.**default_expire_time**()

Determine when a fresh receipt should expire.

Expiration time varies based on configuration (see `[receipt] expiration`).

Returns
a naive UTC datetime.datetime object

keystone.receipt.receipt_formatters module

class keystone.receipt.receipt_formatters.**ReceiptFormatter**

Bases: `object`

Packs and unpacks payloads into receipts for transport.

create_receipt(*user_id*, *methods*, *expires_at*)

Given a set of payload attributes, generate a Fernet receipt.

classmethod creation_time(*fernet_receipt*)

Return the creation time of a valid Fernet receipt.

property crypto

Return a cryptography instance.

You can extend this class with a custom crypto @property to provide your own receipt encoding / decoding. For example, using a different cryptography library (e.g. `python-keyczar`) or to meet arbitrary security requirements.

This @property just needs to return an object that implements `encrypt(plaintext)` and `decrypt(ciphertext)`.

pack(*payload*)

Pack a payload for transport as a receipt.

Return type
str

classmethod restore_padding(*receipt*)

Restore padding based on receipt size.

Parameters
receipt (*str*) receipt to restore padding on

Returns
receipt with correct padding

unpack(*receipt*)

Unpack a receipt, and validate the payload.

Return type
bytes

validate_receipt(*receipt*)

Validate a Fernet receipt and returns the payload attributes.

class keystone.receipt.receipt_formatters.**ReceiptPayload**

Bases: object

classmethod assemble(*user_id*, *methods*, *expires_at*)

Assemble the payload of a receipt.

Parameters

- **user_id** identifier of the user in the receipt request
- **methods** list of authentication methods used
- **expires_at** datetime of the receipts expiration

Returns
the payload of a receipt

classmethod attempt_convert_uuid_hex_to_bytes(*value*)

Attempt to convert value to bytes or return value.

Parameters
value value to attempt to convert to bytes

Returns
tuple containing boolean indicating whether user_id was stored as bytes and uuid value as bytes or the original value

classmethod base64_encode(*s*)

Encode a URL-safe string.

Return type
str

classmethod convert_uuid_bytes_to_hex(*uuid_byte_string*)

Generate uuid.hex format based on byte string.

Parameters
uuid_byte_string uuid string to generate from

Returns
uuid hex formatted string

classmethod convert_uuid_hex_to_bytes(*uuid_string*)

Compress UUID formatted strings to bytes.

Parameters
uuid_string uuid string to compress to bytes

Returns
a byte representation of the uuid

classmethod disassemble(*payload*)

Disassemble a payload into the component data.

The tuple consists of:

```
(user_id, methods, expires_at_str)
```

- `methods` are the auth methods.

Parameters
payload this variant of payload

Returns
a tuple of the payloads component data

classmethod random_urlsafe_str_to_bytes(*s*)

Convert string from `random_urlsafe_str()` to bytes.

Return type
bytes

Module contents

keystone.resource package

Subpackages

keystone.resource.backends package

Submodules

keystone.resource.backends.base module

class keystone.resource.backends.base.**ResourceDriverBase**

Bases: `object`

check_project_depth(*max_depth*)

Check the projects depth in the backend whether exceed the limit.

Parameters
max_depth (*integer*) the limit depth that project depth should not exceed.

Returns
the exceeded projects id or None if no exceeding.

abstract create_project(*project_id*, *project*)

Create a new project.

Parameters

- **project_id** This parameter can be ignored.
- **project** (*dict*) The new project

Project schema:

```
type: object
properties:
    id:
        type: string
    name:
        type: string
    domain_id:
        type: [string, null]
    description:
        type: string
    enabled:
        type: boolean
    parent_id:
        type: string
    is_domain:
        type: boolean
required: [id, name, domain_id]
additionalProperties: true
```

If the project doesnt match the schema the behavior is undefined.

The driver can impose requirements such as the maximum length of a field. If these requirements are not met the behavior is undefined.

Raises
keystone.exception.Conflict if the project id already exists or the name already exists for the domain_id.

abstract delete_project(*project_id*)

Delete an existing project.

Raises
keystone.exception.ProjectNotFound if project_id does not exist

abstract delete_projects_from_ids(*project_ids*)

Delete a given list of projects.

Deletes a list of projects. Ensures no project on the list exists after it is successfully called. If an empty list is provided, the it is silently ignored. In addition, if a project ID in the list of project_ids is not found in the backend, no exception is raised, but a message is logged.

abstract get_project(*project_id*)

Get a project by ID.

Returns
project_ref

Raises
keystone.exception.ProjectNotFound if project_id does not exist

abstract get_project_by_name(*project_name*, *domain_id*)

Get a project by name.

Returns
project_ref

Raises
keystone.exception.ProjectNotFound if a project with the project_name does not exist within the domain

abstract is_leaf_project(*project_id*)

Check if a project is a leaf in the hierarchy.

Parameters
project_id the driver will check if this project is a leaf in the hierarchy.

Raises
keystone.exception.ProjectNotFound if project_id does not exist

abstract list_project_ids_from_domain_ids(*domain_ids*)

List project ids for the provided list of domain ids.

Parameters
domain_ids list of domain ids

Returns
a list of project ids owned by the specified domain ids.

This method is used internally by the assignment manager to bulk read a set of project ids given a list of domain ids.

abstract list_project_parents(*project_id*)

List all parents from a project by its ID.

Parameters
project_id the driver will list the parents of this project.

Returns
a list of project_refs or an empty list.

Raises
keystone.exception.ProjectNotFound if project_id does not exist

abstract list_projects(*hints*)

List projects in the system.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
a list of project_refs or an empty list.

abstract list_projects_acting_as_domain(*hints*)

List all projects acting as domains.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
a list of project_refs or an empty list.

abstract list_projects_from_ids(*project_ids*)

List projects for the provided list of ids.

Parameters
project_ids list of ids

Returns
a list of project_refs.

This method is used internally by the assignment manager to bulk read a set of projects given their ids.

abstract list_projects_in_domain(*domain_id*)

List projects in the domain.

Parameters
domain_id the driver MUST only return projects within this domain.

Returns
a list of project_refs or an empty list.

abstract list_projects_in_subtree(*project_id*)

List all projects in the subtree of a given project.

Parameters
project_id the driver will get the subtree under this project.

Returns
a list of project_refs or an empty list

Raises
keystone.exception.ProjectNotFound if project_id does not exist

abstract update_project(*project_id*, *project*)

Update an existing project.

Raises

- *keystone.exception.ProjectNotFound* if project_id does not exist
- *keystone.exception.Conflict* if project name already exists

keystone.resource.backends.base.**get_project_from_domain**(*domain_ref*)

Create a project ref from the provided domain ref.

keystone.resource.backends.resource_options module

keystone.resource.backends.resource_options.**register_role_options**()

keystone.resource.backends.sql module

class keystone.resource.backends.sql.**Resource**

Bases: *ResourceDriverBase*

check_project_depth(*max_depth*)

Check the projects depth in the backend whether exceed the limit.

Parameters

max_depth (*integer*) the limit depth that project depth should not exceed.

Returns

the exceeded projects id or None if no exceeding.

create_project(*project_id*, *project*)

Create a new project.

Parameters

- **project_id** This parameter can be ignored.
- **project** (*dict*) The new project

Project schema:

```
type: object
properties:
    id:
        type: string
    name:
        type: string
    domain_id:
        type: [string, null]
    description:
        type: string
    enabled:
        type: boolean
    parent_id:
        type: string
    is_domain:
        type: boolean
required: [id, name, domain_id]
additionalProperties: true
```

If the project doesnt match the schema the behavior is undefined.

The driver can impose requirements such as the maximum length of a field. If these requirements are not met the behavior is undefined.

Raises
keystone.exception.Conflict if the project id already exists or the name already exists for the domain_id.

`delete_project`(*project_id*)

Delete an existing project.

Raises
keystone.exception.ProjectNotFound if project_id does not exist

`delete_projects_from_ids`(*project_ids*)

Delete a given list of projects.

Deletes a list of projects. Ensures no project on the list exists after it is successfully called. If an empty list is provided, the it is silently ignored. In addition, if a project ID in the list of project_ids is not found in the backend, no exception is raised, but a message is logged.

`get_project`(*project_id*)

Get a project by ID.

Returns
project_ref

Raises
keystone.exception.ProjectNotFound if project_id does not exist

`get_project_by_name`(*project_name*, *domain_id*)

Get a project by name.

Returns
project_ref

Raises
keystone.exception.ProjectNotFound if a project with the project_name does not exist within the domain

`is_leaf_project`(*project_id*)

Check if a project is a leaf in the hierarchy.

Parameters
project_id the driver will check if this project is a leaf in the hierarchy.

Raises
keystone.exception.ProjectNotFound if project_id does not exist

`list_project_ids_from_domain_ids`(*domain_ids*)

List project ids for the provided list of domain ids.

Parameters
domain_ids list of domain ids

Returns
a list of project ids owned by the specified domain ids.

This method is used internally by the assignment manager to bulk read a set of project ids given a list of domain ids.

list_project_parents(*project_id*)

List all parents from a project by its ID.

Parameters
project_id the driver will list the parents of this project.

Returns
a list of project_refs or an empty list.

Raises
keystone.exception.ProjectNotFound if project_id does not exist

list_projects(*hints*)

List projects in the system.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
a list of project_refs or an empty list.

list_projects_acting_as_domain(*hints*)

List all projects acting as domains.

Parameters
hints filter hints which the driver should implement if at all possible.

Returns
a list of project_refs or an empty list.

list_projects_by_tags(*filters*)

list_projects_from_ids(*ids*)

List projects for the provided list of ids.

Parameters
project_ids list of ids

Returns
a list of project_refs.

This method is used internally by the assignment manager to bulk read a set of projects given their ids.

list_projects_in_domain(*domain_id*)

List projects in the domain.

Parameters
domain_id the driver MUST only return projects within this domain.

Returns
a list of project_refs or an empty list.

list_projects_in_subtree(*project_id*)

List all projects in the subtree of a given project.

Parameters
project_id the driver will get the subtree under this project.

Returns
a list of project_refs or an empty list

Raises
keystone.exception.ProjectNotFound if project_id does not exist

update_project(*project_id*, *project*)

Update an existing project.

Raises

- *keystone.exception.ProjectNotFound* if project_id does not exist
- *keystone.exception.Conflict* if project name already exists

keystone.resource.backends.sql_model module

class keystone.resource.backends.sql_model.**Project**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'name', 'domain_id', 'description', 'enabled', 'parent_id', 'is_domain', 'tags']

description

domain_id

enabled

extra

classmethod from_dict(*project_dict*)

id

is_domain

name

parent_id

resource_options_registry = <keystone.common.resource_options.core.ResourceOptionRegistry object>

property tags

to_dict(*include_extra_dict=False*)

Return the models attributes as a dictionary.

If include_extra_dict is True, extra attributes are literally included in the resulting dictionary twice, for backwards-compatibility with a broken implementation.

class keystone.resource.backends.sql_model.**ProjectOption**(*option_id*, *option_value*)

Bases: Base

option_id

option_value

project_id

class keystone.resource.backends.sql_model.**ProjectTag**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

attributes = ['project_id', 'name']

name

project_id

to_dict()

Return the models attributes as a dictionary.

Module contents

keystone.resource.config_backends package

Submodules

keystone.resource.config_backends.base module

class keystone.resource.config_backends.base.**DomainConfigDriverBase**

Bases: object

Interface description for a Domain Config driver.

abstract create_config_options(*domain_id*, *option_list*)

Create config options for a domain.

Any existing config options will first be deleted.

Parameters

- **domain_id** the domain for this option
- **option_list** a list of dicts, each one specifying an option

Option schema:

```
type: dict
properties:
    group:
        type: string
    option:
        type: string
    value:
        type: depends on the option
    sensitive:
        type: boolean
required: [group, option, value, sensitive]
additionalProperties: false
```

abstract delete_config_options(*domain_id*, *group=None*, *option=None*)

Delete config options for a domain.

Allows deletion of all options for a domain, all options in a group or a specific option. The driver is silent if there are no options to delete.

Parameters

- **domain_id** the domain for this option
- **group** optional group option name
- **option** optional option name. If group is None, then this parameter is ignored

The option is uniquely defined by domain_id, group and option, irrespective of whether it is sensitive ot not.

abstract get_config_option(*domain_id*, *group*, *option*, *sensitive=False*)

Get the config option for a domain.

Parameters

- **domain_id** the domain for this option
- **group** the group name
- **option** the option name
- **sensitive** whether the option is sensitive

Returns

dict containing group, option and value

Raises

keystone.exception.DomainConfigNotFound the option doesnt exist.

abstract list_config_options(*domain_id*, *group=None*, *option=False*, *sensitive=False*)

Get a config options for a domain.

Parameters

- **domain_id** the domain for this option
- **group** optional group option name
- **option** optional option name. If group is None, then this parameter is ignored
- **sensitive** whether the option is sensitive

Returns

list of dicts containing group, option and value

abstract obtain_registration(*domain_id*, *type*)

Try and register this domain to use the type specified.

Parameters

- **domain_id** the domain required
- **type** type of registration

Returns
True if the domain was registered, False otherwise. Failing to register means that someone already has it (which could even be the domain being requested).

abstract read_registration(*type*)
Get the domain ID of who is registered to use this type.

Parameters
type type of registration

Returns
domain_id of who is registered.

Raises
keystone.exception.ConfigRegistrationNotFound If nobody is registered.

abstract release_registration(*domain_id*, *type=None*)
Release registration if it is held by the domain specified.

If the specified domain is registered for this domain then free it, if it is not then do nothing - no exception is raised.

Parameters
- **domain_id** the domain in question
- **type** type of registration, if None then all registrations for this domain will be freed

abstract update_config_options(*domain_id*, *option_list*)
Update config options for a domain.

Parameters
- **domain_id** the domain for this option
- **option_list** a list of dicts, each one specifying an option

keystone.resource.config_backends.sql module

class keystone.resource.config_backends.sql.**ConfigRegister**(**args*, ***kwargs*)
Bases: Base, ModelDictMixin

domain_id

type

class keystone.resource.config_backends.sql.**DomainConfig**
Bases: *DomainConfigDriverBase*

choose_table(*sensitive*)

create_config_options(*domain_id*, *option_list*)
Create config options for a domain.

Any existing config options will first be deleted.

Parameters

- **domain_id** the domain for this option
- **option_list** a list of dicts, each one specifying an option

Option schema:

```
type: dict
properties:
    group:
        type: string
    option:
        type: string
    value:
        type: depends on the option
    sensitive:
        type: boolean
required: [group, option, value, sensitive]
additionalProperties: false
```

delete_config_options(*domain_id*, *group=None*, *option=None*)

Delete config options for a domain.

Allows deletion of all options for a domain, all options in a group or a specific option. The driver is silent if there are no options to delete.

Parameters

- **domain_id** the domain for this option
- **group** optional group option name
- **option** optional option name. If group is None, then this parameter is ignored

The option is uniquely defined by domain_id, group and option, irrespective of whether it is sensitive ot not.

get_config_option(*domain_id*, *group*, *option*, *sensitive=False*)

Get the config option for a domain.

Parameters

- **domain_id** the domain for this option
- **group** the group name
- **option** the option name
- **sensitive** whether the option is sensitive

Returns

dict containing group, option and value

Raises

keystone.exception.DomainConfigNotFound the option doesnt exist.

list_config_options(*domain_id*, *group=None*, *option=None*, *sensitive=False*)

Get a config options for a domain.

Parameters

- **domain_id** the domain for this option
- **group** optional group option name
- **option** optional option name. If group is None, then this parameter is ignored
- **sensitive** whether the option is sensitive

Returns
list of dicts containing group, option and value

obtain_registration(*domain_id*, *type*)

Try and register this domain to use the type specified.

Parameters

- **domain_id** the domain required
- **type** type of registration

Returns
True if the domain was registered, False otherwise. Failing to register means that someone already has it (which could even be the domain being requested).

read_registration(*type*)

Get the domain ID of who is registered to use this type.

Parameters
type type of registration

Returns
domain_id of who is registered.

Raises
keystone.exception.ConfigRegistrationNotFound If nobody is registered.

release_registration(*domain_id*, *type=None*)

Silently delete anything registered for the domain specified.

update_config_options(*domain_id*, *option_list*)

Update config options for a domain.

Parameters

- **domain_id** the domain for this option
- **option_list** a list of dicts, each one specifying an option

class keystone.resource.config_backends.sql.**SensitiveConfig**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

domain_id

group

option

to_dict()

Return the models attributes as a dictionary.

value

class keystone.resource.config_backends.sql.**WhiteListedConfig**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

domain_id

group

option

to_dict()

Return the models attributes as a dictionary.

value

Module contents

Submodules

keystone.resource.core module

Main entry point into the Resource service.

class keystone.resource.core.**DomainConfigManager**

Bases: *Manager*

Default pivot point for the Domain Config backend.

create_config(*domain_id*, *config*)

Create config for a domain.

Parameters

- **domain_id** the domain in question
- **config** the dict of config groups/options to assign to the domain

Creates a new config, overwriting any previous config (no Conflict error will be generated).

Returns

a dict of group dicts containing the options, with any that are sensitive removed

Raises

keystone.exception.InvalidDomainConfig when the config contains options we do not support

delete_config(*domain_id*, *group=None*, *option=None*)

Delete config, or partial config, for the domain.

Parameters

- **domain_id** the domain in question
- **group** an optional specific group of options

- **option** an optional specific option within the group

If group and option are None, then the entire config for the domain is deleted. If group is not None, then just that group of options will be deleted. If group and option are both specified, then just that option is deleted.

Raises

keystone.exception.InvalidDomainConfig when group/option parameters specify an option we do not support or one that does not exist in the original config.

driver_namespace: str = 'keystone.resource.domain_config'

get_config(*domain_id*, *group=None*, *option=None*)

Get config, or partial config, for a domain.

Parameters

- **domain_id** the domain in question
- **group** an optional specific group of options
- **option** an optional specific option within the group

Returns

a dict of group dicts containing the whitelisted options, filtered by group and option specified

Raises

- *keystone.exception.DomainConfigNotFound* when no config found that matches domain_id, group and option specified
- *keystone.exception.InvalidDomainConfig* when the config and group/option parameters specify an option we do not support

An example response:

```
{
    'ldap': {
        'url': 'myurl'
        'user_tree_dn': 'OU=myou'},
    'identity': {
        'driver': 'ldap'}

}
```

get_config_default(*group=None*, *option=None*)

Get default config, or partial default config.

Parameters

- **group** an optional specific group of options
- **option** an optional specific option within the group

Returns

a dict of group dicts containing the default options, filtered by group and option if specified

Raises

keystone.exception.InvalidDomainConfig when the config and group/option parameters specify an option we do not support (or one that is not whitelisted).

An example response:

```
{
    'ldap': {
        'url': 'myurl',
        'user_tree_dn': 'OU=myou',
        ...},
    'identity': {
        'driver': 'ldap'}

}
```

get_config_with_sensitive_info(*domain_id*)

Get config for a domain with sensitive info included.

This method is not exposed via the public API, but is used by the identity manager to initialize a domain with the fully formed config options.

get_security_compliance_config(*domain_id*, *group*, *option=None*)

Get full or partial security compliance config from configuration.

Parameters

- **domain_id** the domain in question
- **group** a specific group of options
- **option** an optional specific option within the group

Returns

a dict of group dicts containing the whitelisted options, filtered by group and option specified

Raises

keystone.exception.InvalidDomainConfig when the config and group/option parameters specify an option we do not support

An example response:

```
{
    'security_compliance': {
        'password_regex': '^(?=.*\d)(?=.*[a-zA-Z]).{7,}$'
        'password_regex_description':
            'A password must consist of at least 1 letter, '
            '1 digit, and have a minimum length of 7 characters'
        }
}
```

sensitive_options = {'identity': [], 'ldap': ['password']}

update_config(*domain_id*, *config*, *group=None*, *option=None*)

Update config, or partial config, for a domain.

Parameters

- **domain_id** the domain in question
- **config** the config dict containing and groups/options being updated
- **group** an optional specific group of options, which if specified must appear in config, with no other groups
- **option** an optional specific option within the group, which if specified must appear in config, with no other options

The contents of the supplied config will be merged with the existing config for this domain, updating or creating new options if these did not previously exist. If group or option is specified, then the update will be limited to those specified items and the inclusion of other options in the supplied config will raise an exception, as will the situation when those options do not already exist in the current config.

Returns

a dict of groups containing all whitelisted options

Raises

keystone.exception.InvalidDomainConfig when the config and group/option parameters specify an option we do not support or one that does not exist in the original config

whitelisted_options = {'identity': ['driver', 'list_limit'], 'ldap': ['url', 'user', 'suffix', 'query_scope', 'page_size', 'alias_dereferencing', 'debug_level', 'chase_referrals', 'user_tree_dn', 'user_filter', 'user_objectclass', 'user_id_attribute', 'user_name_attribute', 'user_mail_attribute', 'user_description_attribute', 'user_pass_attribute', 'user_enabled_attribute', 'user_enabled_invert', 'user_enabled_mask', 'user_enabled_default', 'user_attribute_ignore', 'user_default_project_id_attribute', 'user_enabled_emulation', 'user_enabled_emulation_dn', 'user_enabled_emulation_use_group_config', 'user_additional_attribute_mapping', 'group_tree_dn', 'group_filter', 'group_objectclass', 'group_id_attribute', 'group_name_attribute', 'group_members_are_ids', 'group_member_attribute', 'group_desc_attribute', 'group_attribute_ignore', 'group_additional_attribute_mapping', 'tls_cacertfile', 'tls_cacertdir', 'use_tls', 'tls_req_cert', 'use_pool', 'pool_size', 'pool_retry_max', 'pool_retry_delay', 'pool_connection_timeout', 'pool_connection_lifetime', 'use_auth_pool', 'auth_pool_size', 'auth_pool_connection_lifetime']}

class keystone.resource.core.**Manager**

Bases: *Manager*

Default pivot point for the Resource backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

assert_domain_enabled(*domain_id*, *domain=None*)

Assert the Domain is enabled.

Raises
AssertionError if domain is disabled.

assert_domain_not_federated(*domain_id*, *domain*)

Assert the Domains name and id do not match the reserved keyword.

Note that the reserved keyword is defined in the configuration file, by default, it is Federated, it is also case insensitive. If configs option is empty the default hardcoded value Federated will be used.

Raises
AssertionError if domain named match the value in the config.

assert_project_enabled(*project_id*, *project=None*)

Assert the project is enabled and its associated domain is enabled.

Raises
AssertionError if the project or domain is disabled.

check_project_depth(*max_depth=None*)

Check project depth whether greater than input or not.

create_domain(*domain_id*, *domain*, *initiator=None*)

create_project(*project_id*, *project*, *initiator=None*)

create_project_tag(*project_id*, *tag*, *initiator=None*)

Create a new tag on project.

Parameters

- **project_id** ID of a project to create a tag for
- **tag** The string value of a tag to add

Returns
The value of the created tag

delete_domain(*domain_id*, *initiator=None*)

delete_project(*project_id*, *initiator=None*, *cascade=False*)

Delete one project or a subtree.

Parameters
cascade (*boolean*) If true, the specified project and all its sub-projects are deleted. Otherwise, only the specified project is deleted.

Raises

- *keystone.exception.ValidationError* if project is a domain
- *keystone.exception.Forbidden* if project is not a leaf

delete_project_tag(*project_id*, *tag*)

Delete single tag from project.

Parameters

- **project_id** The ID of the project
- **tag** The tag value to delete

Raises

keystone.exception.ProjectTagNotFound If the tag name does not exist on the project

driver_namespace: str = 'keystone.resource'

get_domain(*domain_id*)

get_domain_by_name(*domain_name*)

get_project(*project_id*)

get_project_by_name(*project_name*, *domain_id*)

get_project_parents_as_ids(*project*)

Get the IDs from the parents from a given project.

The project IDs are returned as a structured dictionary traversing up the hierarchy to the top level project. For example, considering the following project hierarchy:

```
  A
  |
+-B-+
|   |
C   D
```

If we query for project C parents, the expected return is the following dictionary:

```
'parents': {
    B['id']: {
        A['id']: None
    }
}
```

get_project_tag(*project_id*, *tag_name*)

Return information for a single tag on a project.

Parameters

- **project_id** ID of a project to retrive a tag from
- **tag_name** Name of a tag to return

Raises

keystone.exception.ProjectTagNotFound If the tag name does not exist on the project

Returns

The tag value

get_projects_in_subtree_as_ids(*project_id*)

Get the IDs from the projects in the subtree from a given project.

The project IDs are returned as a structured dictionary representing their hierarchy. For example, considering the following project hierarchy:

```
  A
  |
+-B-+
|   |
C   D
```

If we query for project A subtree, the expected return is the following dictionary:

```
'subtree': {
    B['id']: {
        C['id']: None,
        D['id']: None
    }
}
```

list_domains(*hints=None*)

list_domains_from_ids(*domain_ids*)

List domains for the provided list of ids.

Parameters
domain_ids list of ids

Returns
a list of domain_refs.

This method is used internally by the assignment manager to bulk read a set of domains given their ids.

list_project_parents(*project_id*, *user_id=None*, *include_limits=False*)

list_project_tags(*project_id*)

List all tags on project.

Parameters
project_id The ID of a project

Returns
A list of tags from a project

list_projects(*hints=None*)

list_projects_acting_as_domain(*hints=None*)

list_projects_in_domain(*domain_id*)

list_projects_in_subtree(*project_id*, *user_id=None*, *include_limits=False*)

update_domain(*domain_id*, *domain*, *initiator=None*)

update_project(*project_id*, *project*, *initiator=None*, *cascade=False*)

update_project_tags(*project_id*, *tags*, *initiator=None*)

Update all tags on a project.

Parameters

- **project_id** The ID of the project to update
- **tags** A list of tags to update on the project

Returns
A list of tags

keystone.resource.schema module

Module contents

keystone.revoke package

Subpackages

keystone.revoke.backends package

Submodules

keystone.revoke.backends.base module

class keystone.revoke.backends.base.**RevokeDriverBase**

Bases: object

Interface for recording and reporting revocation events.

abstract list_events(*last_fetch=None*, *token=None*)

Return the revocation events, as a list of objects.

Parameters

- **last_fetch** Time of last fetch. Return all events newer.
- **token** dictionary of values from a token, normalized for differences between v2 and v3. The checked values are a subset of the attributes of model.TokenEvent

Returns
A list of keystone.revoke.model.RevokeEvent newer than *last_fetch.* If no last_fetch is specified, returns all events for tokens issued after the expiration cutoff.

abstract revoke(*event*)

Register a revocation event.

Parameters
event An instance of keystone.revoke.model.RevocationEvent

keystone.revoke.backends.base.**revoked_before_cutoff_time**()

keystone.revoke.backends.sql module

class keystone.revoke.backends.sql.**RevocationEvent**(**args*, ***kwargs*)

Bases: Base, ModelDictMixin

access_token_id

attributes = ['trust_id', 'consumer_id', 'access_token_id', 'audit_id', 'audit_chain_id', 'expires_at', 'domain_id', 'project_id', 'user_id', 'role_id', 'issued_before', 'revoked_at']

audit_chain_id

audit_id

consumer_id

domain_id

expires_at

id

issued_before

project_id

revoked_at

role_id

trust_id

user_id

class keystone.revoke.backends.sql.**Revoke**

Bases: *RevokeDriverBase*

list_events(*last_fetch=None*, *token=None*)

Return the revocation events, as a list of objects.

Parameters

- **last_fetch** Time of last fetch. Return all events newer.
- **token** dictionary of values from a token, normalized for differences between v2 and v3. The checked values are a subset of the attributes of model.TokenEvent

Returns

A list of keystone.revoke.model.RevokeEvent newer than *last_fetch*. If no last_fetch is specified, returns all events for tokens issued after the expiration cutoff.

revoke(*event*)

Register a revocation event.

Parameters

event An instance of keystone.revoke.model.RevocationEvent

Module contents

Submodules

keystone.revoke.core module

Main entry point into the Revoke service.

class keystone.revoke.core.**Manager**

Bases: *Manager*

Default pivot point for the Revoke backend.

Performs common logic for recording revocations.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

check_token(*token*)

Check the values from a token against the revocation list.

Parameters

token dictionary of values from a token, normalized for differences between v2 and v3. The checked values are a subset of the attributes of model.TokenEvent

Raises

keystone.exception.TokenNotFound If the token is invalid.

driver_namespace: str = 'keystone.revoke'

list_events(*last_fetch=None*)

revoke(*event*)

revoke_by_audit_chain_id(*audit_chain_id*, *project_id=None*, *domain_id=None*)

revoke_by_audit_id(*audit_id*)

revoke_by_user(*user_id*)

keystone.revoke.model module

Module contents

keystone.server package

Subpackages

keystone.server.flask package

Subpackages

keystone.server.flask.request_processing package

Subpackages

keystone.server.flask.request_processing.middleware package

Submodules

keystone.server.flask.request_processing.middleware.auth_context module

class keystone.server.flask.request_processing.middleware.auth_context.**AuthContextMiddleware**

Bases: *ProviderAPIMixin*, BaseAuthProtocol

Build the authentication context from the request auth token.

classmethod factory(*global_config*, ***local_config*)

Used for loading in middleware (holdover from paste.deploy).

fetch_token(*token*, ***kwargs*)

Fetch the token data based on the value in the header.

Retrieve the data associated with the token value that was in the header. This can be from PKI, contacting the identity server or whatever is required.

Parameters

- **token** (*str*) The token present in the request header.
- **kwargs** (*dict*) Additional keyword arguments may be passed through here to support new features. If an implementation is not aware of how to use these arguments it should ignore them.

Raises

exc.InvalidToken if token is invalid.

Returns

The token data

Return type

dict

fill_context(*request*)

kwargs_to_fetch_token = True

process_request(*request*)

Process request.

If this method returns a value then that value will be used as the response. The next application down the stack will not be executed and process_response will not be called.

Otherwise, the next application down the stack will be executed and process_response will be called with the generated response.

By default this method does not return a value.

Parameters
request (*_request.AuthTokenRequest*) Incoming request

keystone.server.flask.request_processing.middleware.url_normalize module

class keystone.server.flask.request_processing.middleware.url_normalize.**URLNormalizingMiddle**
Bases: object

Middleware filter to handle URL normalization.

Module contents

Submodules

keystone.server.flask.request_processing.json_body module

keystone.server.flask.request_processing.json_body.**json_body_before_request**()
Enforce JSON Request Body.

keystone.server.flask.request_processing.req_logging module

keystone.server.flask.request_processing.req_logging.**log_request_info**()

Module contents

Submodules

keystone.server.flask.application module

keystone.server.flask.application.**application_factory**(*name='public'*)

keystone.server.flask.application.**fail_gracefully**(*f*)
Log exceptions and aborts.

keystone.server.flask.common module

class keystone.server.flask.common.**APIBase**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)
Bases: object

property api

property blueprint

classmethod instantiate_and_register_to_app(*flask_app*)

Build the API object and register to the passed in flask_app.

This is a simplistic loader that makes assumptions about how the blueprint is loaded. Anything beyond defaults should be done explicitly via normal instantiation where more values may be passed via `__init__()`.

Returns
keystone.server.flask.common.APIBase

abstract property resource_mapping: list[*ResourceMap*]

An attr containing of an iterable of *ResourceMap*.

Each *ResourceMap* is a NamedTuple with the following elements:

- resource: a `flask_restful.Resource` class or subclass
- **url: a url route to match for the resource, standard flask**
  routing rules apply. Any url variables will be passed to the resource method as args. (str)
- **alternate_urls: an iterable of url routes to match for the**
  resource, standard flask routing rules apply. These rules are in addition (for API compat) to the primary url. Any url variables will be passed to the resource method as args. (iterable)
- **json_home_data: *JsonHomeData* populated with relevant**
  info for populated JSON Home Documents or None.
- **kwargs: a dict of optional value(s) that can further modify the**
  handling of the routing.
  - **endpoint: endpoint name (defaults to**
    `Resource.__name__.lower()` Can be used to reference this route in `fields.Url` fields (str)
  - **resource_class_args: args to be forwarded to the**
    constructor of the resource. (tuple)
  - **resource_class_kwargs: kwargs to be forwarded to the**
    constructor of the resource. (dict)

  Additional keyword arguments not specified above will be passed as-is to `flask.Flask.add_url_rule()`.

property resources: List[Type[*ResourceBase*]]

class keystone.server.flask.common.**JsonHomeData**(*rel*, *status*, *path_vars*)

Bases: `tuple`

path_vars

Alias for field number 2

rel

Alias for field number 0

status

Alias for field number 1

class keystone.server.flask.common.**ResourceBase**

Bases: Resource

api_prefix: str = ''

property audit_initiator

A pyCADF initiator describing the current authenticated context.

As a property.

property auth_context

static build_driver_hints(*supported_filters*)

Build list hints based on the context query string.

Parameters

supported_filters list of filters supported, so ignore any keys in query_dict that are not in this list.

collection_key: str

classmethod filter_by_attributes(*refs*, *hints*)

Filter a list of references by filter values.

classmethod filter_params(*ref*)

Remove unspecified parameters from the dictionary.

This function removes unspecified parameters from the dictionary. This method checks only root-level keys from a ref dictionary.

Parameters

ref a dictionary representing deserialized response to be serialized

classmethod get_token_ref()

Retrieve KeystoneToken object from the auth context and returns it.

Raises

keystone.exception.Unauthorized If auth context cannot be found.

Returns

The KeystoneToken object.

classmethod limit(*refs*, *hints*)

Limit a list of entities.

The underlying driver layer may have already truncated the collection for us, but in case it was unable to handle truncation we check here.

Parameters

- **refs** the list of members of the collection
- **hints** hints, containing, among other things, the limit requested

Returns

boolean indicating whether the list was truncated, as well as the list of (truncated if necessary) entities.

member_key: str

method_decorators: list[Callable] = []

property oslo_context

static query_filter_is_true(*filter_name*)

Determine if bool query param is True.

We treat this the same way as we do for policy enforcement:

{bool_param}=0 is treated as False

Any other value is considered to be equivalent to True, including the absence of a value (but existence as a parameter).

False Examples for param named *p*:

- http://host/url
- http://host/url?p=0

All other forms of the param p would be result in a True value including: *http://host/url?param*.

property request_body_json

classmethod wrap_collection(*refs*, *hints=None*, *collection_name=None*)

Wrap a collection, checking for filtering and pagination.

Returns the wrapped collection, which includes: - Executing any filtering not already carried out - Truncate to a set limit if necessary - Adds self links in every member - Adds next, self and prev links for the whole collection.

Parameters

- **refs** the list of members of the collection
- **hints** list hints, containing any relevant filters and limit. Any filters already satisfied by managers will have been removed
- **collection_name** optional override for the collection key class attribute. This is to be used when wrapping a collection for a different api, e.g. roles from the trust api.

classmethod wrap_member(*ref*, *collection_name=None*, *member_name=None*)

class keystone.server.flask.common.**ResourceMap**(*resource*, *url*, *alternate_urls*, *kwargs*, *json_home_data*)

Bases: tuple

alternate_urls

Alias for field number 2

json_home_data

Alias for field number 4

kwargs

Alias for field number 3

resource

Alias for field number 0

url

Alias for field number 1

keystone.server.flask.common.**base_url**(*path=''*)

keystone.server.flask.common.**construct_json_home_data**(*rel*, *status='stable'*, *path_vars=None*, *resource_relation_func=<function build_v3_resource_relation>*)

keystone.server.flask.common.**construct_resource_map**(*resource*, *url*, *resource_kwargs*, *alternate_urls=None*, *rel=None*, *status='stable'*, *path_vars=None*, *resource_relation_func=<function build_v3_resource_relation>*)

Construct the ResourceMap Named Tuple.

Parameters

- **resource** (*ResourceMap*) The flask-RESTful resource class implementing the methods for the API.
- **url** (*str*) Flask-standard url route, all flask url routing rules apply. url variables will be passed to the Resource methods as arguments.
- **resource_kwargs** a dict of optional value(s) that can further modify the handling of the routing.
  - **endpoint: endpoint name (defaults to** `Resource.__name__.lower()` Can be used to reference this route in `fields.Url` fields (str)
  - **resource_class_args: args to be forwarded to the** constructor of the resource. (tuple)
  - **resource_class_kwargs: kwargs to be forwarded to** the constructor of the resource. (dict)

  Additional keyword arguments not specified above will be passed as-is to `flask.Flask.add_url_rule()`.
- **alternate_urls** An iterable (list) of dictionaries containing urls and associated json home REL data. Each element is expected to be a dictionary with a url key and an optional json_home key for a JsonHomeData named tuple These urls will also map to the resource. These are used to ensure API compatibility when a new path is more correct for the API but old paths must continue to work. Example: */auth/domains* being the new path for */OS-FEDERATION/domains*. The *OS-FEDERATION* part would be listed as an alternate url. If a json_home key is provided, the original path with the new json_home data will be added to the JSON Home Document.
- **rel** (*str or None*)
- **status** (*str*) JSON Home API Status, e.g. STABLE
- **path_vars** (*dict or None*) JSON Home Path Var Data (arguments)

- **resource_relation_func** (*callable*) function to build expected resource rel data

Type
iterable or None

Returns

keystone.server.flask.common.**full_url**(*path=''*)

keystone.server.flask.common.**set_unenforced_ok**()

keystone.server.flask.common.**unenforced_api**(*f*)

Decorate a resource method to mark is as an unenforced API.

Explicitly exempts an API from receiving the enforced API check, specifically for cases such as user self-service password changes (or other APIs that must work without already having a token).

This decorator may also be used if the API has extended enforcement logic/varying enforcement logic (such as some of the AUTH paths) where the full enforcement will be implemented directly within the methods.

keystone.server.flask.core module

keystone.server.flask.core.**initialize_application**(*name*, *post_log_configured_function=<function <lambda>*, *config_files=None*)

keystone.server.flask.core.**setup_app_middleware**(*app*)

Module contents

class keystone.server.flask.**APIBase**(*blueprint_url_prefix=''*, *api_url_prefix=''*, *default_mediatype='application/json'*, *decorators=None*, *errors=None*)

Bases: object

property api

property blueprint

classmethod instantiate_and_register_to_app(*flask_app*)

Build the API object and register to the passed in flask_app.

This is a simplistic loader that makes assumptions about how the blueprint is loaded. Anything beyond defaults should be done explicitly via normal instantiation where more values may be passed via `__init__()`.

Returns
keystone.server.flask.common.APIBase

abstract property resource_mapping: list[*ResourceMap*]

An attr containing of an iterable of *ResourceMap*.

Each *ResourceMap* is a NamedTuple with the following elements:

- resource: a `flask_restful.Resource` class or subclass
- **url: a url route to match for the resource, standard flask**
  routing rules apply. Any url variables will be passed to the resource method as args. (str)
- **alternate_urls: an iterable of url routes to match for the**
  resource, standard flask routing rules apply. These rules are in addition (for API compat) to the primary url. Any url variables will be passed to the resource method as args. (iterable)
- **json_home_data: *JsonHomeData* populated with relevant**
  info for populated JSON Home Documents or None.
- **kwargs: a dict of optional value(s) that can further modify the**
  handling of the routing.
  - **endpoint: endpoint name (defaults to**
    `Resource.__name__.lower()` Can be used to reference this route in `fields.Url` fields (str)
  - **resource_class_args: args to be forwarded to the**
    constructor of the resource. (tuple)
  - **resource_class_kwargs: kwargs to be forwarded to the**
    constructor of the resource. (dict)

  Additional keyword arguments not specified above will be passed as-is to `flask.Flask.add_url_rule()`.

property resources: List[Type[*ResourceBase*]]

class keystone.server.flask.**JsonHomeData**(*rel*, *status*, *path_vars*)

Bases: `tuple`

path_vars

Alias for field number 2

rel

Alias for field number 0

status

Alias for field number 1

class keystone.server.flask.**ResourceBase**

Bases: `Resource`

api_prefix: str = ''

property audit_initiator

A pyCADF initiator describing the current authenticated context.

As a property.

property auth_context

static build_driver_hints(*supported_filters*)

Build list hints based on the context query string.

Parameters
supported_filters list of filters supported, so ignore any keys in query_dict that are not in this list.

collection_key: str

classmethod filter_by_attributes(*refs*, *hints*)

Filter a list of references by filter values.

classmethod filter_params(*ref*)

Remove unspecified parameters from the dictionary.

This function removes unspecified parameters from the dictionary. This method checks only root-level keys from a ref dictionary.

Parameters
ref a dictionary representing deserialized response to be serialized

classmethod get_token_ref()

Retrieve KeystoneToken object from the auth context and returns it.

Raises
keystone.exception.Unauthorized If auth context cannot be found.

Returns
The KeystoneToken object.

classmethod limit(*refs*, *hints*)

Limit a list of entities.

The underlying driver layer may have already truncated the collection for us, but in case it was unable to handle truncation we check here.

Parameters

- **refs** the list of members of the collection
- **hints** hints, containing, among other things, the limit requested

Returns
boolean indicating whether the list was truncated, as well as the list of (truncated if necessary) entities.

member_key: str

method_decorators: list[Callable] = []

property oslo_context

static query_filter_is_true(*filter_name*)

Determine if bool query param is True.

We treat this the same way as we do for policy enforcement:

{bool_param}=0 is treated as False

Any other value is considered to be equivalent to True, including the absence of a value (but existence as a parameter).

False Examples for param named *p*:

- http://host/url
- http://host/url?p=0

All other forms of the param p would be result in a True value including: *http://host/url?param*.

property request_body_json

classmethod wrap_collection(*refs*, *hints=None*, *collection_name=None*)

Wrap a collection, checking for filtering and pagination.

Returns the wrapped collection, which includes: - Executing any filtering not already carried out - Truncate to a set limit if necessary - Adds self links in every member - Adds next, self and prev links for the whole collection.

Parameters

- **refs** the list of members of the collection
- **hints** list hints, containing any relevant filters and limit. Any filters already satisfied by managers will have been removed
- **collection_name** optional override for the collection key class attribute. This is to be used when wrapping a collection for a different api, e.g. roles from the trust api.

classmethod wrap_member(*ref*, *collection_name=None*, *member_name=None*)

class keystone.server.flask.**ResourceMap**(*resource*, *url*, *alternate_urls*, *kwargs*, *json_home_data*)

Bases: tuple

alternate_urls

Alias for field number 2

json_home_data

Alias for field number 4

kwargs

Alias for field number 3

resource

Alias for field number 0

url

Alias for field number 1

keystone.server.flask.**base_url**(*path=''*)

keystone.server.flask.**construct_json_home_data**(*rel*, *status='stable'*, *path_vars=None*, *resource_relation_func=<function build_v3_resource_relation>*)

keystone.server.flask.**construct_resource_map**(*resource*, *url*, *resource_kwargs*, *alternate_urls=None*, *rel=None*, *status='stable'*, *path_vars=None*, *resource_relation_func=<function build_v3_resource_relation>*)

Construct the ResourceMap Named Tuple.

Parameters

- **resource** (*ResourceMap*) The flask-RESTful resource class implementing the methods for the API.
- **url** (*str*) Flask-standard url route, all flask url routing rules apply. url variables will be passed to the Resource methods as arguments.
- **resource_kwargs** a dict of optional value(s) that can further modify the handling of the routing.
  - **endpoint: endpoint name (defaults to** `Resource.__name__.lower()` Can be used to reference this route in `fields.Url` fields (str)
  - **resource_class_args: args to be forwarded to the** constructor of the resource. (tuple)
  - **resource_class_kwargs: kwargs to be forwarded to** the constructor of the resource. (dict)

  Additional keyword arguments not specified above will be passed as-is to `flask.Flask.add_url_rule()`.
- **alternate_urls** An iterable (list) of dictionaries containing urls and associated json home REL data. Each element is expected to be a dictionary with a url key and an optional json_home key for a JsonHomeData named tuple These urls will also map to the resource. These are used to ensure API compatibility when a new path is more correct for the API but old paths must continue to work. Example: */auth/domains* being the new path for */OS-FEDERATION/domains*. The *OS-FEDERATION* part would be listed as an alternate url. If a json_home key is provided, the original path with the new json_home data will be added to the JSON Home Document.
- **rel** (*str or None*)
- **status** (*str*) JSON Home API Status, e.g. STABLE
- **path_vars** (*dict or None*) JSON Home Path Var Data (arguments)
- **resource_relation_func** (*callable*) function to build expected resource rel data

Type
iterable or None

Returns

keystone.server.flask.**full_url**(*path=''*)

keystone.server.flask.**unenforced_api**(*f*)

Decorate a resource method to mark is as an unenforced API.

Explicitly exempts an API from receiving the enforced API check, specifically for cases such as user self-service password changes (or other APIs that must work without already having a token).

This decorator may also be used if the API has extended enforcement logic/varying enforcement logic (such as some of the AUTH paths) where the full enforcement will be implemented directly within the methods.

Submodules

keystone.server.backends module

keystone.server.backends.**load_backends**()

keystone.server.wsgi module

keystone.server.wsgi.**initialize_admin_application**()

keystone.server.wsgi.**initialize_public_application**()

Module contents

keystone.server.**configure**(*version=None*, *config_files=None*, *pre_setup_logging_fn=<function <lambda>*)

keystone.server.**setup_backends**(*load_extra_backends_fn=<function <lambda>*, *startup_application_fn=<function <lambda>*)

keystone.token package

Subpackages

keystone.token.providers package

Subpackages

keystone.token.providers.fernet package

Submodules

keystone.token.providers.fernet.core module

class keystone.token.providers.fernet.core.**Provider**(**args*, ***kwargs*)

Bases: *Provider*

generate_id_and_issued_at(*token*)

Generate a token based on the information provided.

Parameters
token (*keystone.models.token.TokenModel*) A token object containing information about the authorization context of the request.

Returns
tuple containing an ID for the token and the issued at time of the token (token_id, issued_at).

`validate_token`(*token_id*)

Validate a given token by its ID and return the token_data.

Parameters
token_id (*str*) the unique ID of the token

Returns
token data as a tuple in the form of:

(user_id, methods, audit_ids, system, domain_id, project_id,
trust_id, federated_group_ids, identity_provider_id, protocol_id, access_token_id, app_cred_id, issued_at, expires_at)

`user_id` is the unique ID of the user as a string `methods` a list of authentication methods used to obtain the token `audit_ids` a list of audit IDs for the token `system` a dictionary containing system scope if system-scoped `domain_id` the unique ID of the domain if domain-scoped `project_id` the unique ID of the project if project-scoped `trust_id` the unique identifier of the trust if trust-scoped `federated_group_ids` list of federated group IDs `identity_provider_id` unique ID of the users identity provider `protocol_id` unique ID of the protocol used to obtain the token `access_token_id` the unique ID of the access_token for OAuth1 tokens `app_cred_id` the unique ID of the application credential `param thumbprint` thumbprint of the certificate for OAuth2.0 mTLS `issued_at` a datetime object of when the token was minted `expires_at` a datetime object of when the token expires

Raises
keystone.exception.TokenNotFound If the token doesnt exist.

Module contents

keystone.token.providers.jws package

Submodules

keystone.token.providers.jws.core module

`class keystone.token.providers.jws.core.JWSFormatter`

Bases: `object`

`algorithm = 'ES256'`

`create_token`(*user_id*, *expires_at*, *audit_ids*, *methods*, *system=None*, *domain_id=None*, *project_id=None*, *trust_id=None*, *federated_group_ids=None*, *identity_provider_id=None*, *protocol_id=None*, *access_token_id=None*, *app_cred_id=None*, *thumbprint=None*)

property private_key

property public_keys

validate_token(*token_id*)

class keystone.token.providers.jws.core.**Provider**(**args*, ***kwargs*)

Bases: *Provider*

generate_id_and_issued_at(*token*)

Generate a token based on the information provided.

Parameters

token (*keystone.models.token.TokenModel*) A token object containing information about the authorization context of the request.

Returns

tuple containing an ID for the token and the issued at time of the token (token_id, issued_at).

validate_token(*token_id*)

Validate a given token by its ID and return the token_data.

Parameters

token_id (*str*) the unique ID of the token

Returns

token data as a tuple in the form of:

(user_id, methods, audit_ids, system, domain_id, project_id,
trust_id, federated_group_ids, identity_provider_id, protocol_id, access_token_id, app_cred_id, issued_at, expires_at)

`user_id` is the unique ID of the user as a string `methods` a list of authentication methods used to obtain the token `audit_ids` a list of audit IDs for the token `system` a dictionary containing system scope if system-scoped `domain_id` the unique ID of the domain if domain-scoped `project_id` the unique ID of the project if project-scoped `trust_id` the unique identifier of the trust if trust-scoped `federated_group_ids` list of federated group IDs `identity_provider_id` unique ID of the users identity provider `protocol_id` unique ID of the protocol used to obtain the token `access_token_id` the unique ID of the access_token for OAuth1 tokens `app_cred_id` the unique ID of the application credential `param thumbprint` thumbprint of the certificate for OAuth2.0 mTLS `issued_at` a datetime object of when the token was minted `expires_at` a datetime object of when the token expires

Raises

keystone.exception.TokenNotFound If the token doesnt exist.

Module contents

Submodules

keystone.token.providers.base module

class keystone.token.providers.base.**Provider**

Bases: object

Interface description for a Token provider.

abstract **generate_id_and_issued_at**(*token*)

Generate a token based on the information provided.

Parameters

token (*keystone.models.token.TokenModel*) A token object containing information about the authorization context of the request.

Returns

tuple containing an ID for the token and the issued at time of the token (token_id, issued_at).

abstract **validate_token**(*token_id*)

Validate a given token by its ID and return the token_data.

Parameters

token_id (*str*) the unique ID of the token

Returns

token data as a tuple in the form of:

(user_id, methods, audit_ids, system, domain_id, project_id,
trust_id, federated_group_ids, identity_provider_id, protocol_id, access_token_id, app_cred_id, issued_at, expires_at)

user_id is the unique ID of the user as a string methods a list of authentication methods used to obtain the token audit_ids a list of audit IDs for the token system a dictionary containing system scope if system-scoped domain_id the unique ID of the domain if domain-scoped project_id the unique ID of the project if project-scoped trust_id the unique identifier of the trust if trust-scoped federated_group_ids list of federated group IDs identity_provider_id unique ID of the users identity provider protocol_id unique ID of the protocol used to obtain the token access_token_id the unique ID of the access_token for OAuth1 tokens app_cred_id the unique ID of the application credential param thumbprint thumbprint of the certificate for OAuth2.0 mTLS issued_at a datetime object of when the token was minted expires_at a datetime object of when the token expires

Raises

keystone.exception.TokenNotFound If the token doesnt exist.

Module contents

Submodules

keystone.token.provider module

Token provider interface.

class keystone.token.provider.**Manager**

Bases: *Manager*

Default pivot point for the token provider backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

V3 = 'v3.0'

VERSIONS = frozenset({'v3.0'})

check_revocation(*token*)

check_revocation_v3(*token_values*)

driver_namespace: str = 'keystone.token.provider'

invalidate_individual_token_cache(*token*)

issue_token(*user_id*, *method_names*, *expires_at=None*, *system=None*, *project_id=None*, *domain_id=None*, *auth_context=None*, *trust_id=None*, *app_cred_id=None*, *thumbprint=None*, *parent_audit_id=None*)

revoke_token(*token_id*, *revoke_chain=False*)

validate_token(*token_id*, *window_seconds=0*, *access_rules_support=None*)

keystone.token.provider.**default_expire_time**()

Determine when a fresh token should expire.

Expiration time varies based on configuration (see `[token] expiration`).

Returns
a naive UTC datetime.datetime object

keystone.token.provider.**random_urlsafe_str**()

Generate a random URL-safe string.

Return type
str

keystone.token.token_formatters module

class keystone.token.token_formatters.**ApplicationCredentialScopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

version: int = 9

class keystone.token.token_formatters.**BasePayload**

Bases: object

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod attempt_convert_uuid_hex_to_bytes(*value*)

Attempt to convert value to bytes or return value.

Parameters

value value to attempt to convert to bytes

Returns

tuple containing boolean indicating whether user_id was stored as bytes and uuid value as bytes or the original value

classmethod base64_encode(*s*)

Encode a URL-safe string.

Return type

str

classmethod convert_uuid_bytes_to_hex(*uuid_byte_string*)

Generate uuid.hex format based on byte string.

Parameters
uuid_byte_string uuid string to generate from

Returns
uuid hex formatted string

classmethod convert_uuid_hex_to_bytes(*uuid_string*)

Compress UUID formatted strings to bytes.

Parameters
uuid_string uuid string to compress to bytes

Returns
a byte representation of the uuid

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters
payload this variant of payload

Returns
a tuple of the payloads component data

classmethod random_urlsafe_str_to_bytes(*s*)

Convert string from `random_urlsafe_str()` to bytes.

Return type
bytes

version: int

class keystone.token.token_formatters.**DomainScopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to

- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns
the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters
payload this variant of payload

Returns
a tuple of the payloads component data

version: int = 1

class keystone.token.token_formatters.**FederatedDomainScopedPayload**

Bases: *FederatedScopedPayload*

version: int = 6

class keystone.token.token_formatters.**FederatedProjectScopedPayload**

Bases: *FederatedScopedPayload*

version: int = 5

class keystone.token.token_formatters.**FederatedScopedPayload**

Bases: *FederatedUnscopedPayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

class keystone.token.token_formatters.**FederatedUnscopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

classmethod pack_group_id(*group_dict*)

classmethod unpack_group_id(*group_id_in_bytes*)

version: int = 4

class keystone.token.token_formatters.**Oauth2CredentialsScopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

version: int = 10

class keystone.token.token_formatters.**OauthScopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

version: int = 7

class keystone.token.token_formatters.**ProjectScopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

version: int = 2

class keystone.token.token_formatters.**SystemScopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

version: int = 8

class keystone.token.token_formatters.**TokenFormatter**

Bases: object

Packs and unpacks payloads into tokens for transport.

create_token(*user_id*, *expires_at*, *audit_ids*, *payload_class*, *methods=None*, *system=None*, *domain_id=None*, *project_id=None*, *trust_id=None*, *federated_group_ids=None*, *identity_provider_id=None*, *protocol_id=None*, *access_token_id=None*, *app_cred_id=None*, *thumbprint=None*)

Given a set of payload attributes, generate a Fernet token.

classmethod creation_time(*fernet_token*)

Return the creation time of a valid Fernet token.

property crypto

Return a cryptography instance.

You can extend this class with a custom crypto @property to provide your own token encoding / decoding. For example, using a different cryptography library (e.g. `python-keyczar`) or to meet arbitrary security requirements.

This @property just needs to return an object that implements `encrypt(plaintext)` and `decrypt(ciphertext)`.

pack(*payload*)

Pack a payload for transport as a token.

Return type
str

classmethod restore_padding(*token*)

Restore padding based on token size.

Parameters
token (*str*) token to restore padding on

Returns
token with correct padding

unpack(*token*)

Unpack a token, and validate the payload.

Return type
bytes

validate_token(*token*)

Validate a Fernet token and returns the payload attributes.

class keystone.token.token_formatters.**TrustScopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

version: int = 3

class keystone.token.token_formatters.**UnscopedPayload**

Bases: *BasePayload*

classmethod assemble(*user_id*, *methods*, *system*, *project_id*, *domain_id*, *expires_at*, *audit_ids*, *trust_id*, *federated_group_ids*, *identity_provider_id*, *protocol_id*, *access_token_id*, *app_cred_id*, *thumbprint*)

Assemble the payload of a token.

Parameters

- **user_id** identifier of the user in the token request
- **methods** list of authentication methods used
- **system** a string including system scope information
- **project_id** ID of the project to scope to
- **domain_id** ID of the domain to scope to
- **expires_at** datetime of the tokens expiration
- **audit_ids** list of the tokens audit IDs
- **trust_id** ID of the trust in effect
- **federated_group_ids** list of group IDs from SAML assertion
- **identity_provider_id** ID of the users identity provider
- **protocol_id** federated protocol used for authentication
- **access_token_id** ID of the secret in OAuth1 authentication
- **app_cred_id** ID of the application credential in effect
- **thumbprint** thumbprint of the certificate in OAuth2 mTLS

Returns

the payload of a token

classmethod disassemble(*payload*)

Disassemble an unscoped payload into the component data.

The tuple consists of:

```
(user_id, methods, system, project_id, domain_id,
 expires_at_str, audit_ids, trust_id, federated_group_ids,
 identity_provider_id, protocol_id,` access_token_id, app_cred_id)
```

- `methods` are the auth methods.

Fields will be set to None if they didnt apply to this payload type.

Parameters

payload this variant of payload

Returns

a tuple of the payloads component data

version: int = 0

Module contents

keystone.trust package

Subpackages

keystone.trust.backends package

Submodules

keystone.trust.backends.base module

class keystone.trust.backends.base.**TrustDriverBase**

Bases: object

abstract consume_use(*trust_id*)

Consume one use of a trust.

One use of a trust is consumed when the trust was created with a limitation on its uses, provided there are still uses available.

Raises

- *keystone.exception.TrustUseLimitReached* If no remaining uses for trust.
- *keystone.exception.TrustNotFound* If the trust doesnt exist.

abstract create_trust(*trust_id*, *trust*, *roles*)

Create a new trust.

Returns
a new trust

abstract delete_trust(*trust_id*)

abstract delete_trusts_for_project(*project_id*)

Delete all trusts for a project.

Parameters
project_id ID of a project to filter trusts by.

abstract flush_expired_and_soft_deleted_trusts(*project_id=None*, *trustor_user_id=None*, *trustee_user_id=None*, *date=None*)

Flush expired and non-expired soft deleted trusts from the backend.

Parameters

- **project_id** ID of a project to filter trusts by.
- **trustor_user_id** ID of a trustor_user_id to filter trusts by.
- **trustee_user_id** ID of a trustee_user_id to filter trusts by.

- **date** (*datetime*) date to filter trusts by.

abstract get_trust(*trust_id*, *deleted=False*)

Get a trust by the trust id.

Parameters

- **trust_id** (*string*) the trust identifier
- **deleted** (*bool*) return the trust even if it is deleted, expired, or has no consumptions left

abstract list_trusts()

abstract list_trusts_for_trustee(*trustee*)

abstract list_trusts_for_trustor(*trustor*, *redelegated_trust_id=None*)

keystone.trust.backends.sql module

class keystone.trust.backends.sql.**Trust**

Bases: *TrustDriverBase*

consume_use(*trust_id*)

Consume one use of a trust.

One use of a trust is consumed when the trust was created with a limitation on its uses, provided there are still uses available.

Raises

- *keystone.exception.TrustUseLimitReached* If no remaining uses for trust.
- *keystone.exception.TrustNotFound* If the trust doesnt exist.

create_trust(*trust_id*, *trust*, *roles*)

Create a new trust.

Returns
a new trust

delete_trust(*trust_id*)

delete_trusts_for_project(*project_id*)

Delete all trusts for a project.

Parameters
project_id ID of a project to filter trusts by.

flush_expired_and_soft_deleted_trusts(*project_id=None*, *trustor_user_id=None*, *trustee_user_id=None*, *date=None*)

Flush expired and non-expired soft deleted trusts from the backend.

Parameters

- **project_id** ID of a project to filter trusts by.
- **trustor_user_id** ID of a trustor_user_id to filter trusts by.

- **trustee_user_id** ID of a trustee_user_id to filter trusts by.
- **date** (*datetime*) date to filter trusts by.

get_trust(*trust_id*, *deleted=False*)

Get a trust by the trust id.

Parameters

- **trust_id** (*string*) the trust identifier
- **deleted** (*bool*) return the trust even if it is deleted, expired, or has no consumptions left

list_trusts()

list_trusts_for_trustee(*trustee_user_id*)

list_trusts_for_trustor(*trustor_user_id*, *redelegated_trust_id=None*)

class keystone.trust.backends.sql.**TrustModel**(**args*, ***kwargs*)

Bases: Base, ModelDictMixinWithExtras

attributes: list[str] = ['id', 'trustor_user_id', 'trustee_user_id', 'project_id', 'impersonation', 'expires_at', 'remaining_uses', 'deleted_at', 'redelegated_trust_id', 'redelegation_count']

deleted_at

expires_at

expires_at_int

extra

id

impersonation

project_id

redelegated_trust_id

redelegation_count

remaining_uses

trustee_user_id

trustor_user_id

class keystone.trust.backends.sql.**TrustRole**(**args*, ***kwargs*)

Bases: Base

attributes = ['trust_id', 'role_id']

role_id

trust_id

Module contents

Submodules

keystone.trust.core module

Main entry point into the Trust service.

class keystone.trust.core.**Manager**

Bases: *Manager*

Default pivot point for the Trust backend.

See *keystone.common.manager.Manager* for more details on how this dynamically calls the backend.

create_trust(*trust_id*, *trust*, *roles*, *redelegated_trust=None*, *initiator=None*)

Create a new trust.

Returns
a new trust

delete_trust(*trust_id*, *initiator=None*)

Remove a trust.

Raises
keystone.exception.TrustNotFound If the trust doesnt exist.

Recursively remove given and redelegated trusts

driver_namespace: str = 'keystone.trust'

get_trust(*trust_id*, *deleted=False*)

get_trust_pedigree(*trust_id*)

keystone.trust.schema module

Module contents

Submodules

keystone.exception module

exception keystone.exception.**AccessRuleNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find Access Rule: %(access_rule_id)s.'

exception keystone.exception.**AccountLocked**(*message=None*, ***kwargs*)

Bases: *Unauthorized*

message_format: str | None = 'The account is locked for user: %(user_id)s.'

exception keystone.exception.**AdditionalAuthRequired**(*auth_response=None*, ***kwargs*)

Bases: *AuthPluginException*

message_format: str | None = 'Additional authentications steps required.'

exception keystone.exception.**AmbiguityError**(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = "There are multiple %(resource)s entities named '%(name)s'. Please use ID instead of names to resolve the ambiguity."

exception keystone.exception.**ApplicationCredentialAuthError**(**args*, ***kwargs*)

Bases: *AuthPluginException*

message_format: str | None = 'Error authenticating with application credential: %(detail)s'

exception keystone.exception.**ApplicationCredentialLimitExceeded**(*message=None*, ***kwargs*)

Bases: *ForbiddenNotSecurity*

message_format: str | None = 'Unable to create additional application credentials, maximum of %(limit)d already exceeded for user.'

exception keystone.exception.**ApplicationCredentialNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find Application Credential: %(application_credential_id)s.'

exception keystone.exception.**ApplicationCredentialValidationError**(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = 'Invalid application credential: %(detail)s'

exception keystone.exception.**AssignmentTypeCalculationError**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'Unexpected combination of grant attributes - User: %(user_id)s, Group: %(group_id)s, Project: %(project_id)s, Domain: %(domain_id)s.'

exception keystone.exception.**AuthMethodNotSupported**(**args*, ***kwargs*)

Bases: *AuthPluginException*

message_format: str | None = 'Attempted to authenticate with an unsupported method.'

exception keystone.exception.**AuthPluginException**(**args*, ***kwargs*)

Bases: *Unauthorized*

message_format: str | None = 'Authentication plugin error.'

exception keystone.exception.**CacheDeserializationError**(*obj*, *data*)

Bases: Exception

exception keystone.exception.**CircularRegionHierarchyError**(*message=None*, ***kwargs*)

Bases: *Error*

code: int | None = 400

message_format: str | None = 'The specified parent region %(parent_region_id)s would create a circular region hierarchy.'

title: str | None = 'Bad Request'

exception keystone.exception.**ConfigFileNotFound**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'The Keystone configuration file %(config_file)s could not be found.'

exception keystone.exception.**ConfigRegistrationNotFound**

Bases: Exception

exception keystone.exception.**Conflict**(*message=None*, ***kwargs*)

Bases: *Error*

code: int | None = 409

message_format: str | None = 'Conflict occurred attempting to store %(type)s - %(details)s.'

title: str | None = 'Conflict'

exception keystone.exception.**CredentialEncryptionError**

Bases: Exception

message_format = 'An unexpected error prevented the server from accessing encrypted credentials.'

exception keystone.exception.**CredentialLimitExceeded**(*message=None*, ***kwargs*)

Bases: *ForbiddenNotSecurity*

message_format: str | None = 'Unable to create additional credentials, maximum of %(limit)d already exceeded for user.'

exception keystone.exception.**CredentialNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find credential: %(credential_id)s.'

exception keystone.exception.**CrossBackendNotAllowed**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'Group membership across backend boundaries is not allowed. Group in question is %(group_id)s, user is %(user_id)s.'

exception keystone.exception.**DirectMappingError**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = "Local section in mapping %(mapping_id)s refers to a remote match that doesn't exist (e.g. {0} in a local section)."

exception keystone.exception.**DomainConfigNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find %(group_or_option)s in domain configuration for domain %(domain_id)s.'

exception keystone.exception.**DomainIdInvalid**(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = 'Domain ID does not conform to required UUID format.'

exception keystone.exception.**DomainNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find domain: %(domain_id)s.'

exception keystone.exception.**DomainSpecificRoleMismatch**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'Project %(project_id)s must be in the same domain as the role %(role_id)s being assigned.'

exception keystone.exception.**DomainSpecificRoleNotWithinIdPDomain**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'role: %(role_name)s must be within the same domain as the identity provider: %(identity_provider)s.'

exception keystone.exception.**EndpointGroupNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find Endpoint Group: %(endpoint_group_id)s.'

exception keystone.exception.**EndpointNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find endpoint: %(endpoint_id)s.'

exception keystone.exception.**Error**(*message=None*, ***kwargs*)

Bases: Exception

Base error class.

Child classes should define an HTTP status code, title, and a message_format.

code: int | None = None

message_format: str | None = None

title: str | None = None

exception keystone.exception.**FederatedProtocolNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find federated protocol %(protocol_id)s for Identity Provider: %(idp_id)s.'

exception keystone.exception.**Forbidden**(*message=None*, ***kwargs*)

Bases: *SecurityError*

code: int | None = 403

message_format: str | None = 'You are not authorized to perform the requested action.'

title: str | None = 'Forbidden'

exception keystone.exception.**ForbiddenAction**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'You are not authorized to perform the requested action: %(action)s.'

exception keystone.exception.**ForbiddenNotSecurity**(*message=None*, ***kwargs*)

Bases: *Error*

When you want to return a 403 Forbidden response but not security.

Use this for errors where the message is always safe to present to the user and wont give away extra information.

code: int | None = 403

title: str | None = 'Forbidden'

exception keystone.exception.**Gone**(*message=None*, ***kwargs*)

Bases: *Error*

code: int | None = 410

message_format: str | None = 'The service you have requested is no longer available on this server.'

title: str | None = 'Gone'

exception keystone.exception.**GroupNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find group: %(group_id)s.'

exception keystone.exception.**IdentityProviderNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find Identity Provider: %(idp_id)s.'

exception keystone.exception.**ImpliedRoleNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = '%(prior_role_id)s does not imply %(implied_role_id)s.'

exception keystone.exception.**InsufficientAuthMethods**(*message=None*, *user_id=None*, *methods=None*)

Bases: *Error*

code: int | None = 401

message_format: str | None = 'Insufficient auth methods received for %(user_id)s. Auth Methods Provided: %(methods)s.'

title: str | None = 'Unauthorized'

exception keystone.exception.**InvalidDomainConfig**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'Invalid domain specific configuration: %(reason)s.'

exception keystone.exception.**InvalidImpliedRole**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = '%(role_id)s cannot be an implied roles.'

exception keystone.exception.**InvalidLimit**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'Invalid resource limit: %(reason)s.'

exception keystone.exception.**InvalidOperatorError**(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = "The given operator %(_op)s is not valid. It must be one of the following: 'eq', 'neq', 'lt', 'lte', 'gt', or 'gte'."

exception keystone.exception.**InvalidPolicyAssociation**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'Invalid mix of entities for policy association: only Endpoint, Service, or Region+Service allowed. Request was - Endpoint: %(endpoint_id)s, Service: %(service_id)s, Region: %(region_id)s.'

exception keystone.exception.**KeysNotFound**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'No encryption keys found; run keystone-manage fernet_setup to bootstrap one.'

exception keystone.exception.**LDAPInvalidCredentialsError**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

message_format: str | None = 'Unable to authenticate against Identity backend - Invalid username or password'

exception keystone.exception.**LDAPServerConnectionError**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'Unable to establish a connection to LDAP Server (%(url)s).'

exception keystone.exception.**LDAPSizeLimitExceeded**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

message_format: str | None = 'Number of User/Group entities returned by LDAP exceeded size limit. Contact your LDAP administrator.'

exception keystone.exception.**LimitNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find limit for %(id)s.'

exception keystone.exception.**LimitTreeExceedError**(*project_id*, *max_limit_depth*)

Bases: Exception

exception keystone.exception.**MalformedEndpoint**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'Malformed endpoint URL (%(endpoint)s), see ERROR log for details.'

exception keystone.exception.**MappedGroupNotFound**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'Group %(group_id)s returned by mapping %(mapping_id)s was not found in the backend.'

exception keystone.exception.**MappingNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find mapping: %(mapping_id)s.'

exception keystone.exception.**MetadataFileError**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'Error while reading metadata file: %(reason)s.'

exception keystone.exception.**MigrationNotProvided**(*mod_name*, *path*)

Bases: Exception

exception keystone.exception.**MultipleSQLDriversInConfig**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'The Keystone domain-specific configuration has specified more than one SQL driver (only one is permitted): %(source)s.'

exception keystone.exception.**NoLimitReference**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'Unable to create a limit that has no corresponding registered limit.'

exception keystone.exception.**NotFound**(*message=None*, ***kwargs*)

Bases: *Error*

code: int | None = 404

message_format: str | None = 'Could not find: %(target)s.'

title: str | None = 'Not Found'

exception keystone.exception.**NotImplemented**(*message=None*, ***kwargs*)

Bases: *Error*

code: int | None = 501

message_format: str | None = 'The action you have requested has not been implemented.'

title: str | None = 'Not Implemented'

exception keystone.exception.**OAuth2Error**(*code*, *title*, *error_title*, *message*)

Bases: *Error*

exception keystone.exception.**OAuth2InvalidClient**(*code*, *title*, *message*)

Bases: *OAuth2Error*

exception keystone.exception.**OAuth2InvalidRequest**(*code*, *title*, *message*)

Bases: *OAuth2Error*

exception keystone.exception.**OAuth2OtherError**(*code*, *title*, *message*)

Bases: *OAuth2Error*

exception keystone.exception.**OAuth2UnsupportedGrantType**(*code*, *title*, *message*)

Bases: *OAuth2Error*

exception keystone.exception.**OAuthHeadersMissingError**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'No Authorization headers found, cannot proceed with OAuth related calls. If running under HTTPd or Apache, ensure WSGIPassAuthorization is set to On.'

exception keystone.exception.**PasswordAgeValidationError**(*message=None*, ***kwargs*)

Bases: *PasswordValidationError*

message_format: str | None = 'You cannot change your password at this time due to the minimum password age. Once you change your password, it must be used for %(min_age_days)d day(s) before it can be changed. Please try again in %(days_left)d day(s) or contact your administrator to reset your password.'

exception keystone.exception.**PasswordExpired**(*message=None*, ***kwargs*)

Bases: *Unauthorized*

message_format: str | None = 'The password is expired and needs to be changed for user: %(user_id)s.'

exception keystone.exception.**PasswordHistoryValidationError**(*message=None*, ***kwargs*)

Bases: *PasswordValidationError*

message_format: str | None = 'The new password cannot be identical to a previous password. The total number which includes the new password must be unique is %(unique_count)s.'

exception keystone.exception.**PasswordRequirementsValidationError**(*message=None*, ***kwargs*)

Bases: *PasswordValidationError*

message_format: str | None = 'The password does not match the requirements: %(detail)s.'

exception keystone.exception.**PasswordSelfServiceDisabled**(*message=None*, ***kwargs*)

Bases: *PasswordValidationError*

message_format: str | None = 'You cannot change your password at this time due to password policy disallowing password changes. Please contact your administrator to reset your password.'

exception keystone.exception.**PasswordValidationError**(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = 'Password validation error: %(detail)s.'

exception keystone.exception.**PasswordVerificationError**(*message=None*, ***kwargs*)

Bases: *ForbiddenNotSecurity*

message_format: str | None = 'The password length must be less than or equal to %(size)i. The server could not comply with the request because the password is invalid.'

exception keystone.exception.**PolicyAssociationNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find policy association.'

exception keystone.exception.**PolicyNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find policy: %(policy_id)s.'

exception keystone.exception.**ProjectNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find project: %(project_id)s.'

exception keystone.exception.**ProjectTagNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find project tag: %(project_tag)s.'

exception keystone.exception.**PublicIDNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = '%(id)s'

exception keystone.exception.**ReceiptNotFound**(*message=None*, ***kwargs*)

Bases: *Unauthorized*

message_format: str | None = 'Could not find auth receipt: %(receipt_id)s.'

exception keystone.exception.**RedirectRequired**(*redirect_url*, ***kwargs*)

Bases: Exception

Error class for redirection.

Child classes should define an HTTP redirect url message_format.

code = 302

redirect_url = None

exception keystone.exception.**RegionDeletionError**(*message=None*, ***kwargs*)

Bases: *ForbiddenNotSecurity*

message_format: str | None = 'Unable to delete region %(region_id)s because it or its child regions have associated endpoints.'

exception keystone.exception.**RegionNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find region: %(region_id)s.'

exception keystone.exception.**RegisteredLimitError**(*message=None*, ***kwargs*)

Bases: *ForbiddenNotSecurity*

message_format: str | None = 'Unable to update or delete registered limit %(id)s because there are project limits associated with it.'

exception keystone.exception.**RegisteredLimitNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find registered limit for %(id)s.'

exception keystone.exception.ResourceDeleteForbidden(*message=None*, ***kwargs*)

Bases: *ForbiddenNotSecurity*

message_format: str | None = 'Unable to delete immutable %(type)s resource: `%(resource_id)s. Set resource option "immutable" to false first.'

exception keystone.exception.ResourceUpdateForbidden(*message=None*, ***kwargs*)

Bases: *ForbiddenNotSecurity*

message_format: str | None = 'Unable to update immutable %(type)s resource: `%(resource_id)s. Set resource option "immutable" to false first.'

exception keystone.exception.RoleAssignmentNotFound(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find role assignment with role: %(role_id)s, user or group: %(actor_id)s, project, domain, or system: %(target_id)s.'

exception keystone.exception.RoleNotFound(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find role: %(role_id)s.'

exception keystone.exception.SAMLSigningError(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'Unable to sign SAML assertion. It is likely that this server does not have xmlsec1 installed or this is the result of misconfiguration. Reason %(reason)s.'

exception keystone.exception.SchemaValidationError(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = '%(detail)s'

exception keystone.exception.SecurityError(*message=None*, ***kwargs*)

Bases: *Error*

Security error exception.

Avoids exposing details of security errors, unless in insecure_debug mode.

amendment = '(Disable insecure_debug mode to suppress these details.)'

exception keystone.exception.ServiceNotFound(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find service: %(service_id)s.'

exception keystone.exception.**ServiceProviderNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find Service Provider: %(sp_id)s.'

exception keystone.exception.**StringLengthExceeded**(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = "String length exceeded. The length of string '%(string)s' exceeds the limit of column %(type)s(CHAR(%(length)d))."

exception keystone.exception.**TokenNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find token: %(token_id)s.'

exception keystone.exception.**TokenlessAuthConfigError**(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = 'Could not determine Identity Provider ID. The configuration option %(issuer_attribute)s was not found in the request environment.'

exception keystone.exception.**TrustConsumeMaximumAttempt**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'Unable to consume trust %(trust_id)s. Unable to acquire lock.'

exception keystone.exception.**TrustNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find trust: %(trust_id)s.'

exception keystone.exception.**TrustUseLimitReached**(*message=None*, ***kwargs*)

Bases: *Forbidden*

message_format: str | None = 'No remaining uses for trust: %(trust_id)s.'

exception keystone.exception.**URLValidationError**(*message=None*, ***kwargs*)

Bases: *ValidationError*

message_format: str | None = 'Cannot create an endpoint with an invalid URL: %(url)s.'

exception keystone.exception.**Unauthorized**(*message=None*, ***kwargs*)

Bases: *SecurityError*

code: int | None = 401

message_format: str | None = 'The request you have made requires authentication.'

title: str | None = 'Unauthorized'

exception keystone.exception.**UnexpectedError**(*message=None*, ***kwargs*)

Bases: *SecurityError*

Avoids exposing details of failures, unless in insecure_debug mode.

code: int | None = 500

debug_message_format = 'An unexpected error prevented the server from fulfilling your request: %(exception)s.'

message_format: str | None = 'An unexpected error prevented the server from fulfilling your request.'

title: str | None = 'Internal Server Error'

exception keystone.exception.**UnsupportedTokenVersionException**(*message=None*, ***kwargs*)

Bases: *UnexpectedError*

debug_message_format = 'Token version is unrecognizable or unsupported.'

exception keystone.exception.**UserDisabled**(*message=None*, ***kwargs*)

Bases: *Unauthorized*

message_format: str | None = 'The account is disabled for user: %(user_id)s.'

exception keystone.exception.**UserNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find user: %(user_id)s.'

exception keystone.exception.**ValidationError**(*message=None*, ***kwargs*)

Bases: *Error*

code: int | None = 400

message_format: str | None = 'Expecting to find %(attribute)s in %(target)s. The server could not comply with the request since it is either malformed or otherwise incorrect. The client is assumed to be in error.'

title: str | None = 'Bad Request'

exception keystone.exception.**ValidationExpirationError**(*message=None*, ***kwargs*)

Bases: *Error*

code: int | None = 400

message_format: str | None = "The 'expires_at' must not be before now. The server could not comply with the request since it is either malformed or otherwise incorrect. The client is assumed to be in error."

title: str | None = 'Bad Request'

exception keystone.exception.**ValidationTimeStampError**(*message=None*, ***kwargs*)

Bases: *Error*

code: int | None = 400

message_format: str | None = 'Timestamp not in expected format. The server could not comply with the request since it is either malformed or otherwise incorrect. The client is assumed to be in error.'

title: str | None = 'Bad Request'

exception keystone.exception.**VersionNotFound**(*message=None*, ***kwargs*)

Bases: *NotFound*

message_format: str | None = 'Could not find version: %(version)s.'

keystone.i18n module

oslo.i18n integration module.

See https://docs.openstack.org/oslo.i18n/latest/user/usage.html .

keystone.notifications module

Notifications module for OpenStack Identity Service resources.

keystone.notifications.**ACTIONS = ('created', 'deleted', 'disabled', 'updated', 'internal')**

The actions on resources.

class keystone.notifications.**Audit**

Bases: object

Namespace for audit notification functions.

This is a namespace object to contain all of the direct notification functions utilized for Manager methods.

classmethod added_to(*target_type*, *target_id*, *actor_type*, *actor_id*, *initiator=None*, *public=True*, *reason=None*)

classmethod created(*resource_type*, *resource_id*, *initiator=None*, *public=True*, *reason=None*)

classmethod deleted(*resource_type*, *resource_id*, *initiator=None*, *public=True*, *reason=None*)

classmethod disabled(*resource_type*, *resource_id*, *initiator=None*, *public=True*, *reason=None*)

classmethod internal(*resource_type*, *resource_id*, *reason=None*)

classmethod removed_from(*target_type*, *target_id*, *actor_type*, *actor_id*, *initiator=None*, *public=True*, *reason=None*)

classmethod updated(*resource_type*, *resource_id*, *initiator=None*, *public=True*, *reason=None*)

class keystone.notifications.**CadfNotificationWrapper**(*operation*)

Bases: object

Send CADF event notifications for various methods.

This function is only used for Authentication events. Its action and event_type are dictated below.

- action: authenticate
- event_type: identity.authenticate

Sends CADF notifications for events such as whether an authentication was successful or not.

Parameters
operation The authentication related action being performed

class keystone.notifications.**CadfRoleAssignmentNotificationWrapper**(*operation*)

Bases: object

Send CADF notifications for role_assignment methods.

This function is only used for role assignment events. Its action and event_type are dictated below.

- action: created.role_assignment or deleted.role_assignment
- **event_type: identity.role_assignment.created or** identity.role_assignment.deleted

Sends a CADF notification if the wrapped method does not raise an Exception (such as *keystone.exception.NotFound*).

Parameters
operation one of the values from ACTIONS (created or deleted)

ROLE_ASSIGNMENT = 'role_assignment'

keystone.notifications.**build_audit_initiator**()

A pyCADF initiator describing the current authenticated context.

keystone.notifications.**clear_subscribers**()

Empty subscribers dictionary.

This effectively stops notifications since there will be no subscribers to publish to.

keystone.notifications.**emit_event**

alias of *CadfNotificationWrapper*

keystone.notifications.**invalidate_token_cache_notification**(*reason*)

A specific notification for invalidating the token cache.

Parameters
reason (*string*) The specific reason why the token cache is being invalidated.

keystone.notifications.**listener**(*cls*)

A class decorator to declare a class to be a notification listener.

A notification listener must specify the event(s) it is interested in by defining a `event_callbacks` attribute or property. `event_callbacks` is a dictionary where the key is the type of event and the value is a dictionary containing a mapping of resource types to callback(s).

ACTIONS contains constants for the currently supported events. There is currently no single place to find constants for the resource types.

Example:

```
@listener
class Something(object):

    def __init__(self):
        self.event_callbacks = {
            notifications.ACTIONS.created: {
                'user': self._user_created_callback,
            },
            notifications.ACTIONS.deleted: {
                'project': [
                    self._project_deleted_callback,
                    self._do_cleanup,
                ]
            },
        }
```

keystone.notifications.**notify_event_callbacks**(*service*, *resource_type*, *operation*, *payload*)

Send a notification to registered extensions.

keystone.notifications.**register_event_callback**(*event*, *resource_type*, *callbacks*)

Register each callback with the event.

Parameters

- **event** (*keystone.notifications.ACTIONS*) Action being registered
- **resource_type** (*str*) Type of resource being operated on
- **callbacks** (*list*) Callback items to be registered with event

Raises

- **ValueError** If event is not a valid ACTION
- **TypeError** If callback is not callable

keystone.notifications.**reset_notifier**()

Reset the notifications internal state.

This is used only for testing purposes.

keystone.notifications.**role_assignment**

alias of *CadfRoleAssignmentNotificationWrapper*

keystone.notifications.**send_saml_audit_notification**(*action*, *user_id*, *group_ids*, *identity_provider*, *protocol*, *token_id*, *outcome*)

Send notification to inform observers about SAML events.

Parameters

- **action** (*str*) Action being audited
- **user_id** (*str*) User ID from Keystone token
- **group_ids** (*list*) List of Group IDs from Keystone token
- **identity_provider** (*str or None*) ID of the IdP from the Keystone token
- **protocol** (*str*) Protocol ID for IdP from the Keystone token
- **token_id** (*str or None*) audit_id from Keystone token
- **outcome** (*str*) One of `pycadf.cadftaxonomy`

keystone.version module

keystone.version.**release_string**()

Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FIVE

CONTRIBUTOR DOCUMENTATION

5.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Keystone.

5.1.1 Communication

For communicating with Keystone Team, you can reach out to us through mailing lists and IRC channels.

5.1.2 Contacting the Core Team

For any help contact keystone maintainers , the core team of keystone.

5.1.3 New Feature Planning

If you are planning to propose a feature in keystone , check out Proposing Features

5.1.4 Task Tracking

We track our tasks in keystone bug tracker. You can also track the tasks of other keystone repositories also.

- keystonemiddleware
- keystoneauth
- python-keystoneclient

If youre looking for some smaller, easier work item to pick up and get started on, search for the low-hanging-fruit tag in bugs launchpad.

5.1.5 Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so in keystone bug tracker by following the bug triage procedure.

5.1.6 Getting Your Patch Merged

After submitting a Patch, anyone can cooperate by reviewing the patch on gerrit. Finally, the patch will be merged by the keystone maintainers.

Project Team Lead Duties

All common PTL duties are enumerated here in the PTL guide.

5.2 How Can I Help?

Are you interested in contributing to the keystone project? Whether youre a software developer, a technical writer, an OpenStack operator or an OpenStack user, there are many reasons to get involved with the keystone project:

- You can help shape the direction of the project, ensuring it meets your organizations needs in the future
- You can help maintain the projects health and get your bugs fixed faster
- You can collaborate with other people to find common solutions that will help you and your organization
- You can hack on a fun, security-related Python project with interesting challenges

Here are some easy ways to make a big difference to the keystone project and become part of the team:

- Read the documentation, starting with the rest of this contributor guide, and try to follow it to set up keystone and try out different features. Does it make sense? Is something out of date? Is something misleading or incorrect? Submit a patch or bug report to fix it.
- Monitor incoming bug reports, try to reproduce the bug in a test environment, ask the bug reporter for more information, answer support questions and close invalid bugs. Follow the bug triage guide. New bugs can be found with the New status:
  - keystone
  - keystonemiddleware
  - keystoneauth
  - python-keystoneclient

  You can also subscribe to email notifications for new bugs.
- Subscribe to the openstack-discuss@lists.openstack.org mailing list (filter on subject tag `[keystone]`) and join the #openstack-keystone IRC channel on OFTC. Help answer user support questions if you or your organization has faced and solved a similar problem, or chime in on design discussions that will affect you and your organization.
- Check out the low hanging fruit bugs, submit patches to fix them:

  - keystone
  - keystonemiddleware
  - keystoneauth
  - python-keystoneclient
- Look for deprecation warnings in the unit tests and in the keystone logs of a running keystone installation and submit patches to make them go away.
- Look at other projects, especially devstack, and submit patches to correct usage of options that keystone has deprecated. Make sure to let the keystone maintainers know youre looking at these so that its on their radar and they can help review.
- Check the test coverage report (`tox -ecover`) and try to add unit test coverage.
- Review new changes. Keep OpenStacks review guidelines in mind. Ask questions when you dont understand a change.

Need any help? *Reach out* to the keystone team.

5.2.1 The Meaning of Low Hanging Fruit

This section describes the intent behind bugs tagged as low hanging fruit. Current maintainers should apply the tag consistently while triaging bugs, using this document as a guide. This practice ensures newcomers to the project can expect each low hanging fruit bug to be of similar complexity.

Bugs fit for the low hanging fruit tag:

- Should require minimal python experience, someone new to OpenStack might also be new to python
- Should only require a basic understanding of the review workflow, complicated changesets with dependencies between repositories coupled with CI testing only raises the cognitive bar for new contributors
- Can include documentation fixes so long it doesnt require an in-depth understanding of complicated subsystems and features (e.g., overhauling the federated identity guide)
- Should be something a newcomer can progress through in a week or less, long wait times due to the discussion of complicated topics can deter new contributors from participating
- Shouldnt require a new contributor to understand copious amounts of historical context, newcomers should eventually understand this information but consuming that information is outside the scope of low hanging fruit

5.3 Setting up Keystone

5.3.1 Prerequisites

This document assumes you are using an Ubuntu, Fedora, or openSUSE platform and that you have the following tools pre-installed on your system:

- python 3.6, as the programming language;
- git, as the version control tool;

Note

Keystone dropped the support of python 2.7 in the Ussuri release of Openstack.

Reminder: If you are successfully using a different platform, or a different version of the above, please document your configuration here!

5.3.2 Installing from source

The source install instructions specifically avoid using platform specific packages. Instead, we recommend using the source for the code and the Python Package Index (PyPi) for development environment installations..

Its expected that your system already has python, pip, and git available.

Clone the keystone repository:

```
$ git clone https://opendev.org/openstack/keystone.git
$ cd keystone
```

Install the keystone web service:

```
$ pip install -e .
```

Note

This step is guaranteed to fail if you do not have the proper binary dependencies already installed on your development system. Maintaining a list of platform-specific dependencies is outside the scope of this documentation, but is within scope of DEVSTACK.

5.3.3 Development environment

For setting up the Python development environment and running *tox* testing environments, please refer to the Project Team Guide: Python Project Guide, the OpenStack guide on wide standard practices around the use of Python.

That documentation will help you configure your development environment and run keystone tests using *tox*, which uses virtualenv to isolate the Python environment. After running it, notice the existence of a *.tox* directory.

5.3.4 Deploying configuration files

You should be able to run keystone after installing via pip. Additional configuration files are required. The following file is required in order to run keystone:

- `keystone.conf`

Configuring Keystone with a sample file

Keystone requires a configuration file. Keystones sample configuration file `etc/keystone.conf.sample` is automatically generated based upon all of the options available within Keystone. These options are sourced from the many files around Keystone as well as some external libraries.

The sample configuration file will be updated as the end of the development cycle approaches. Developers should *NOT* generate the config file and propose it as part of their patches, this will cause unnecessary conflicts. You can generate one locally using the following command:

```
$ tox -e genconfig
```

The tox command will place an updated sample config in `etc/keystone.conf.sample`. The defaults are enough to get you going, but you can make any changes if needed.

If there is a new external library (e.g. `oslo.messaging`) that utilizes the `oslo.config` package for configuration, it can be added to the list of libraries found in `config-generator/keystone.conf`.

You can also generate sample policy files using `tox -e genpolicy`. Please refer to *API Configuration options* for guidance on specific configuration options or to view a sample paste file.

5.3.5 Bootstrapping a test deployment

You can use the `keystone-manage bootstrap` command to pre-populate the database with necessary data.

5.3.6 Verifying keystone is set up

Once set up, you should be able to invoke Python and import the libraries:

```
$ .tox/py36/bin/python -c "import keystone"
```

If you can import keystone without a traceback, you should be ready to move on to the next sections.

You can run keystone using a host of wsgi implementations or web servers. The following uses `uwsgi`:

```
$ uwsgi --http 127.0.0.1:5000 --wsgi-file $(which keystone-wsgi-public)
```

This runs Keystone with the configuration the etc/ directory of the project. See *API Configuration options* for details on how Keystone is configured. By default, Keystone is configured with SQL backends.

5.3.7 Database setup

The script `tools/test-setup.sh` sets up databases as used by the unit tests.

5.3.8 Initializing Keystone

Before using keystone, it is necessary to create the database tables and ensures the database schemas are up to date, perform the following:

```
$ keystone-manage db_sync
```

If the above commands result in a `KeyError`, or they fail on a `.pyc` file with the message, `You can only have one Python script per version`, then it is possible that there are out-of-date compiled Python bytecode files in the Keystone directory tree that are causing problems. This can occur if you have previously installed and ran older versions of Keystone. These out-of-date files can be easily removed by running a command like the following from the Keystone root project directory:

```
$ find . -name "*.pyc" -delete
```

Initial Sample Data

There is an included script which is helpful in setting up some initial sample data for use with keystone:

```
$ ADMIN_PASSWORD=s3cr3t tools/sample_data.sh
```

Once run, you can see the sample data that has been created by using the python-openstackclient command-line interface:

```
$ export OS_USERNAME=admin
$ export OS_PASSWORD=s3cr3t
$ export OS_PROJECT_NAME=admin
$ export OS_USER_DOMAIN_ID=default
$ export OS_PROJECT_DOMAIN_ID=default
$ export OS_IDENTITY_API_VERSION=3
$ export OS_AUTH_URL=http://localhost:5000/v3
$ openstack user list
```

The python-openstackclient can be installed using the following:

```
$ pip install python-openstackclient
```

5.3.9 Interacting with Keystone

You can also interact with keystone through its REST API. There is a Python keystone client library python-keystoneclient which interacts exclusively through the REST API, and a command-line interface python-openstackclient command-line interface.

5.4 Identity API v2.0 and v3 History

5.4.1 Specifications

As of the Queens release, Keystone solely implements the Identity API v3. Support for Identity API v2.0 has been removed in Queens in favor of the Identity API v3.

Identity API v3 is a superset of all the functionality available in the Identity API v2.0 and several of its extensions, and provides a much more consistent developer experience.

5.4.2 History

Youre probably wondering why Keystone does not implement a v1 API. As a matter of fact, one exists, but it actually predates OpenStack. The v1.x API was an extremely small API documented and implemented by Rackspace for their early public cloud products.

With the advent of OpenStack, Keystone served to provide a superset of the authentication and multi-tenant authorization models already implemented by Rackspaces public cloud, Nova, and Swift. Thus, Identity API v2.0 was introduced.

Identity API v3 was established to introduce namespacing for users and projects by using domains as a higher-level container for more flexible identity management and fixed a security issue in the v2.0 API (bearer tokens appearing in URLs).

5.4.3 How do I migrate from v2.0 to v3?

I am a deployer

You need to ensure that youve configured your service catalog in Keystone correctly. The simplest, and most ideal, configuration would expose one identity with unversioned endpoints (note the lack of `/v2.0/` or `/v3/` in these URLs):

- Service (type: `identity`)
  - Endpoint (interface: `public`, URL: `http://identity:5000/`)
  - Endpoint (interface: `admin`, URL: `http://identity:35357/`)

If you were to perform a `GET` against either of these endpoints, you would be greeted by an `HTTP/1.1 300 Multiple Choices` response, which newer Keystone clients can use to automatically detect available API versions.

Note

Deploying v3 only requires a single application since administrator and end-user operations are handled by the same process, and not separated into two different applications. Depending on how v2.0 was configured, you might be able to decommission one endpoint. Until users are educated about which endpoint to use, the former admin API (e.g. using port 35357) and the public API (e.g. using port 5000) can run the v3 API simulateously and serve both sets of users.

```
$ curl -i http://identity:35357/
HTTP/1.1 300 Multiple Choices
Vary: X-Auth-Token
Content-Type: application/json
Content-Length: 755
Date: Tue, 10 Jun 2014 14:22:26 GMT

{"versions": {"values": [ ... ]}}
```

With unversioned `identity` endpoints in the service catalog, you should be able to authenticate with keystoneclient successfully.

I have a Python client

The Keystone community provides first-class support for Python API consumers via our client library, python-keystoneclient. If youre not currently using this library, you should, as it is intended to expose all of our HTTP API functionality. If were missing something youre looking for, please contribute!

Adopting python-keystoneclient should be the easiest way to migrate to Identity API v3.

I have a non-Python client

Youll likely need to heavily reference our API documentation to port your application to Identity API v3.

The most common operation would be password-based authentication including a tenant name (i.e. project name) to specify an authorization scope. In Identity API v2.0, this would be a request to `POST /v2.0/tokens`:

```
{
    "auth": {
        "passwordCredentials": {
            "password": "my-password",
            "username": "my-username"
        },
        "tenantName": "project-x"
    }
}
```

And you would get back a JSON blob with an `access -> token -> id` that you could pass to another web service as your `X-Auth-Token` header value.

In Identity API v3, an equivalent request would be to `POST /v3/auth/tokens`:

```
{
    "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                    "domain": {
                        "id": "default"
                    },
                    "name": "my-username",
                    "password": "my-password"
                }
            }
        },
        "scope": {
            "project": {
                "domain": {
                    "id": "default"
                },
                "name": "project-x"
            }
        }
    }
}
```

Note a few key differences when compared to the v2.0 API:

- A tenant in v2.0 became a project in v3.
- The authentication method (`password`) is explicitly identified.
- Both the user name (`my-username`) and project name (`project-x`) are namespaced by an owning domain (where `id = default`). The default domain exists by default in Keystone, and automatically owns the namespace exposed by Identity API v2.0. Alternatively, you may reference users and projects that exist outside the namespace of the default domain, which are thus inaccessible to the v2.0 API.
- In v3, your token is returned to you in an `X-Subject-Token` header, instead of as part of the request body. You should still authenticate yourself to other services using the `X-Auth-Token` header.

5.4.4 Why do I see deployments with Keystone running on two ports?

During development of the v2.0 API, operational functionality was isolated into different applications within the project. One application was dedicated to end-user functionality and its sole purpose was to authenticate and validate user identities. The second application consisted of more features and allowed operators the ability to manage their deployment by adding or deleting users, creating projects, etc. These applications were referred to as the `public` and `admin` APIs, respectively. This deployment model was required by the architecture of the v2.0 API. In a way, authorization was limited to the application you had access to.

Once development began on the v3 API, the code paths for both applications were merged into one. Instead of isolating functionality into separate applications, all functionality was consolidated into a single application. Each v3 endpoint or API is protected by policy instead. This makes deployment and management of Keystones infrastructure easier for operators to deploy and for users to consume. As a result, Keystone deployments are not required to deploy separate `admin` and `public` endpoints, especially now that the v2.0 API implementation has been removed.

5.4.5 HTTP/1.1 Chunked Encoding

Warning

Running Keystone under HTTPD in the recommended (and tested) configuration does not support the use of `Transfer-Encoding: chunked`. This is due to a limitation with the WSGI spec and the implementation used by `mod_wsgi`. It is recommended that all clients assume Keystone will not support `Transfer-Encoding: chunked`.

5.5 Proposing Features

Requests for enhancements or new features must follow a process that requires using bug reports and specifications. We publish the contents of the keystone-specs repository at specs.openstack.org.

5.5.1 RFE Bug Reports

All code, documentation, and tests implementing a feature should be tracked. To do this, we use Launchpad bug reports. We use bug reports because the OpenStack review infrastructure has existing tooling that groups patches based on commit message syntax. When you propose a patch that is related to a bug or a feature, the OpenStack Infrastructure bot automatically links the patch as a comment in the bug report. Comments are also immutable, allowing us to track long-running initiatives without losing context.

To create an RFE bug report, file a bug against the appropriate project. For example, if we were to create an RFE bug report for supporting a new Foobar API within keystone, wed open that RFE against the keystone project. The title should start with RFE: , followed by a snippet of the feature or enhancement. For example, RFE: Implement a Foobar API. The description should be short. Since we use specifications for details, we dont need to duplicate information in the body of the bug report. After you create the bug, you can tag it with the rfe tag, which helps people filter feature work from other bug reports. Finally, if your specification has already merged, be sure to include a link to it as a comment. If it hasnt, you can propose, or re-propose, your specification with `Partial-Bug:` followed by the bug number, at the bottom of your commit message. The OpenStack Infrastructure bot automatically updates the RFE bug report you just created with a link to the proposed specification. The specification template explains how

to link to RFE bug reports, which should prompt you to open your RFE bug prior to proposing your specification.

If your feature is broken up into multiple commits, make sure to include `Partial-Bug` in your commit messages. Additionally, use `Closes-Bug` in the last commit implementing the feature. This process ensures all patches written for a feature are tracked in the bug report, making it easier to audit. If you miss the opportunity to use the `Closes-Bug` tag and your feature work is complete, set the bug status to Fix Committed.

5.5.2 Specifications

We use specifications as a way to describe, in detail, the change that were making and why.

To write a specification, you can follow the template provided in the repository. To start writing a new specification, copy the template to the directory that fits the project and release you plan to target. For example, if you want to propose a feature to keystone for the Stein release, you should do the following:

```
$ cp specs/template.rst specs/keystone/stein/feature-foobar.rst
```

Once you have a template in place, work through each section. Specifications should be descriptive and include use cases that justify the work. There are sections dedicated to the problem statement, the proposed solution, alternative solutions, security concerns, among other things. These sections are meant to prompt you to think about how your feature impacts users, operators, developers, related projects, and the existing code base. The template acts as a guide, so if you need to inject an ad-hoc section to describe additional details of your feature, dont hesitate to add one. Do not remove sections from the template that do not apply to your specification. Instead, simply explain why your proposed change doesnt have an impact on that aspect of the template. Propose your specification for review when youre ready for feedback:

```
$ git review
```

The process for reviewing specifications is handled using Gerrit. We dont restrict the specification selection process to a particular group of individuals, which allows for open and collaborative feedback. We encourage everyone to be a part of the review process. Applying a code-review methodology to specifications allows different people to think through the problem youre trying to solve. Everyone wants to ensure the best design possible, given various resource constraints. This process takes time. Dont be discouraged if it takes longer than you anticipated for your specification to get feedback. A specification must have support (+2) from at least two keystone-spec core reviewers and it is typically approved (+Workflow) by the PTL, in order to be formally accepted.

5.6 Working with Release Notes

The Keystone team uses reno to generate release notes. These are important user-facing documents that must be included when a user or operator facing change is performed, like a bug-fix or a new feature. A release note should be included in the same patch the work is being performed. Release notes should be short, easy to read, and easy to maintain. They also *must* link back to any appropriate documentation if it exists. The following conventions help ensure all release notes achieve those goals.

Most release notes either describe bug fixes or announce support for new features, both of which are tracked using Launchpad. The conventions below rely on links in Launchpad to provide readers with more context.

Warning

We highly recommend taking careful thought when writing and reviewing release notes. Once a release note has been published with a formal release, updating it across releases will cause it to be published in a subsequent release. Reviews that update, or modify, a release note from a previous release outside of the branch it was added in should be rejected unless its required for a very specific case.

Please refer to renos documentation for more information.

5.6.1 Release Notes for Bugs

When creating a release note that communicates a bug fix, use the bug number in the name of the release note:

```
$ reno new bug-1652012
Created new notes file in releasenotes/notes/bug-1652012-7c53b9702b10084d.yaml
```

The body of the release note should clearly explain how the impact will affect users and operators. It should also include why the change was necessary but not be overspecific about implementation details, as that can be found in the commit and the bug report. It should contain a properly formatted link in reStructuredText that points back to the original bug report used to track the fix. This ensures the release note is kept short and to-the-point while providing readers with additional resources:

```
---
fixes:
  - |
    [`bug 1652012 <https://bugs.launchpad.net/keystone/+bug/1652012>`_]
    Changes the token_model to return is_admin_project False if the
    attribute is not defined. Returning True for this has the potential to
    be dangerous and the given reason for keeping it True was strictly for
    backwards compatibility.
```

5.6.2 Release Notes for Features

Release notes detailing feature work follow the same basic format, since features are also tracked as bugs.

```
$ reno new bug-1652012
Created new notes file in releasenotes/notes/bug-1652012-7c53b9702b10084d.yaml
```

Just like release notes communicating bug fixes, release notes detailing feature work must contain a link back to the RFE bug report. Readers should be able to easily discover all patches that implement the feature, as well as find links to the full specification and documentation. The release notes can be added to the last patch of the feature. All of this is typically found in the RFE bug report registered in Launchpad:

```
---
features:
  - >
    [`bug 1652012 <https://bugs.launchpad.net/keystone/+bug/1652012>`_]
    Keystone now fully supports the usage of fizzbangs.
```

In the rare case there is a release note that does not pertain to a bug or feature work, use a sensible slug and include any documentation relating to the note. We can iterate on the content and application of the release note during the review process.

For more information on how and when to create release notes, see the project-team-guide.

5.7 Testing Keystone

5.7.1 Running Tests

Before running tests, you should have `tox` installed and available in your environment (in addition to the other external dependencies in *Setting up Keystone*):

```
$ pip install tox
```

Note

You may need to perform both the above operation and the next inside a python virtualenv, or prefix the above command with `sudo`, depending on your preference.

To execute the full suite of tests maintained within keystone, simply run:

```
$ tox
```

This iterates over multiple configuration variations, and uses external projects to do light integration testing to verify the Identity API against other projects.

Note

The first time you run `tox`, it will take additional time to build virtualenvs. You can later use the `-r` option with `tox` to rebuild your virtualenv in a similar manner.

To run tests for one or more specific test environments (for example, the most common configuration of Python 3.6 and PEP-8), list the environments with the `-e` option, separated by spaces:

```
$ tox -e py36,pep8
```

Note

Keystone dropped the support of python 2.7 in the Ussuri release of Openstack.

Use `tox --listenvs` to list all testing environments specified in keystones `tox.ini` file.

Interactive debugging

Using pdb breakpoints with `tox` and `testr` normally doesnt work since the tests just fail with a `BdbQuit` exception rather than stopping at the breakpoint.

To capture breakpoints while running tests, use the `debug` environment. The following example uses the environment while invoking a specific test run.

```
$ tox -e debug keystone.tests.unit.test_module.TestClass.test_case
```

For reference, the `debug` environment implements the instructions here: https://wiki.openstack.org/wiki/Testr#Debugging_.28pdb.29_Tests

5.7.2 Building the Documentation

The `docs` and `api-ref` environments will automatically generate documentation and the API reference respectively. The results are written to `doc/` and `api-ref/`.

For example, use the following command to render all documentation and manual pages:

```
$ tox -e docs
```

5.7.3 Tests Structure

Not all of the tests in the `keystone/tests/unit` directory are strictly unit tests. Keystone intentionally includes tests that run the service locally and drives the entire configuration to achieve basic functional testing.

For the functional tests, an in-memory key-value store or in-memory SQLite database is used to keep the tests fast.

Within the tests directory, the general structure of the backend tests is a basic set of tests represented under a test class, and then subclasses of those tests under other classes with different configurations to drive different backends through the APIs. To add tests covering all drivers, update the base test class in `test_backend.py`.

Note

The structure of backend testing is in transition, migrating from having all classes in a single file (`test_backend.py`) to one where there is a directory structure to reduce the size of the test files. See:

- `keystone.tests.unit.backend.role`
- `keystone.tests.unit.backend.domain_config`

To add new drivers, subclass the base class at `test_backend.py` (look at `test_backend_sql.py` for examples) and update the configuration of the test class in `setUp()`.

For example, `test_backend.py` has a sequence of tests under the class `keystone.tests.unit.test_backend.IdentityTests` that will work with the default drivers. The `test_backend_sql.py` module subclasses those tests, changing the configuration by overriding with configuration files stored in the `tests/unit/config_files` directory aimed at enabling the SQL backend for the Identity module.

5.7.4 Testing Schema Migrations

Tests for database migrations can be found in `keystone/tests/unit/test_sql_upgrade.py` and `keystone/tests/unit/test_sql_banned_operations.py`.

5.7.5 LDAP Tests

LDAP has a fake backend that performs rudimentary operations. If you are building more significant LDAP functionality, you should test against a live LDAP server. Devstack has an option to set up a directory server for Keystone to use. Add ldap to the `ENABLED_SERVICES` environment variable, and set environment variables `KEYSTONE_IDENTITY_BACKEND=ldap` and `KEYSTONE_CLEAR_LDAP=yes` in your `localrc` file.

The unit tests can be run against a live server with `keystone/tests/unit/test_ldap_livetest.py` and `keystone/tests/unit/test_ldap_pool_livetest.py`. The default password is `test` but if you have installed devstack with a different LDAP password, modify the file `keystone/tests/unit/config_files/backend_liveldap.conf` and `keystone/tests/unit/config_files/backend_pool_liveldap.conf` to reflect your password.

Note

To run the live tests you need to set the environment variable `ENABLE_LDAP_LIVE_TEST` to a non-negative value.

5.7.6 Work in progress Tests

Work in progress (WIP) tests are very useful in a variety of situations including:

- While doing test-driven-development they can be used to add tests to a review while they are not yet working and will not cause test failures. They can be removed when the functionality is fixed in a later patch set.
- A common practice is to recreate bugs by exposing the broken behavior in a functional or unit test. To encapsulate the correct behavior in the test, the test will usually assert the correct outcome, which will break without a fix. Marking the test as WIP gives us the ability to capture the broken behavior in code if a fix isnt ready yet.

The `keystone.tests.unit.utils.wip()` decorator can be used to mark a test as WIP. A WIP test will always be run. If the test fails then a SkipTest exception is raised because we expect the test to fail. We do not pass the test in this case so that it doesnt count toward the number of successfully run tests. If the test passes an AssertionError exception is raised so that the developer knows they made the test pass. This is a reminder to remove the decorator.

The `keystone.tests.unit.utils.wip()` decorator requires that the author provides a message. This message is important because it will tell other developers why this test is marked as a work in progress. Reviewers will require that these messages are descriptive and accurate.

Note

The `keystone.tests.unit.utils.wip()` decorator is not a replacement for skipping tests.

```
@wip('waiting on bug #000000')
def test():
    pass
```

Note

Another strategy is to not use the wip decorator and instead show how the code currently incorrectly works. Which strategy is chosen is up to the developer.

5.7.7 API & Scenario Tests

Keystone provides API and scenario tests via a tempest plugin which is located in a separate repository. This tempest plugin is mainly intended for specific scenarios that require a special deployment, such as the tests for the `Federated Identity` feature or live testing against LDAP. For the deployment of these scenarios, keystone also provides a devstack plugin.

For example, to setup a working federated environment, add the following lines in your *devstack local.conf*' file:

```
[[local|localrc]]
enable_plugin keystone https://opendev.org/openstack/keystone
enable_service keystone-saml2-federation
```

Clone and install keystone-tempest-plugin.

```
git clone https://opendev.org/openstack/keystone-tempest-plugin
sudo pip install ./keystone-tempest-plugin
```

Finally, to run keystones API and scenario tests, deploy tempest with devstack (using the configuration above) and then run the following command from the tempest directory:

```
tox -e all -- keystone_tempest_plugin
```

Note

Most of keystones API tests are implemented in tempest and it is usually the correct place to add new tests.

Writing new API & Scenario Tests

When writing tests for the keystone tempest plugin, we should follow the official tempest guidelines, details about the guidelines can be found at the tempest coding guide. There are also specific guides for the API and scenario tests: Tempest Field Guide to API tests and Tempest Field Guide to Scenario tests.

The keystone tempest plugin also provides a base class. For most cases, the tests should inherit from it: `keystone_tempest_plugin.tests.base.BaseIdentityTest`. This class already setups the identity API version and is the container of all API services clients. New API services clients `keystone_tempest_plugin.services` (which are used to communicate with the REST API from

the services) should also be added to this class. For example, below we have a snippet from the tests at `keystone_tempest_plugin.tests.api.identity.v3.test_identity_providers.py`.

```
class IdentityProvidersTest(base.BaseIdentityTest):

...

def _create_idp(self, idp_id, idp_ref):
    idp = self.idps_client.create_identity_provider(
        idp_id, **idp_ref)['identity_provider']
    self.addCleanup(
        self.idps_client.delete_identity_provider, idp_id)
    return idp

@decorators.idempotent_id('09450910-b816-4150-8513-a2fd4628a0c3')
def test_identity_provider_create(self):
    idp_id = data_utils.rand_uuid_hex()
    idp_ref = fixtures.idp_ref()
    idp = self._create_idp(idp_id, idp_ref)

    # The identity provider is disabled by default
    idp_ref['enabled'] = False

    # The remote_ids attribute should be set to an empty list by default
    idp_ref['remote_ids'] = []

    self._assert_identity_provider_attributes(idp, idp_id, idp_ref)
```

The test class extends `keystone_tempest_plugin.tests.base.BaseIdentityTest`. Also, the `_create_idp` method calls keystones API using the `idps_client`, which is an instance from. `keystone_tempest_plugin.tests.services.identity.v3.identity_providers_client.IdentityProvidersClient`.

Additionally, to illustrate the construction of a new test class, below we have a snippet from the scenario test that checks the complete federated authentication workflow (`keystone_tempest_plugin.tests.scenario.test_federated_authentication.py`). In the test setup, all of the needed resources are created using the API service clients. Since it is a scenario test, it is common to need some customized settings that will come from the environment (in this case, from the devstack plugin) - these settings are collected in the `_setup_settings` method.

```
class TestSaml2EcpFederatedAuthentication(base.BaseIdentityTest):

...

def _setup_settings(self):
    self.idp_id = CONF.fed_scenario.idp_id
    self.idp_url = CONF.fed_scenario.idp_ecp_url
    self.keystone_v3_endpoint = CONF.identity.uri_v3
    self.password = CONF.fed_scenario.idp_password
    self.protocol_id = CONF.fed_scenario.protocol_id
    self.username = CONF.fed_scenario.idp_username
```

(continues on next page)

(continued from previous page)

```
...

def setUp(self):
    super(TestSaml2EcpFederatedAuthentication, self).setUp()
    self._setup_settings()

    # Reset client's session to avoid getting garbage from another runs
    self.saml2_client.reset_session()

    # Setup identity provider, mapping and protocol
    self._setup_idp()
    self._setup_mapping()
    self._setup_protocol()
```

Finally, the tests perform the complete workflow of the feature, asserting correctness in each step:

```
def _request_unscoped_token(self):
    resp = self.saml2_client.send_service_provider_request(
        self.keystone_v3_endpoint, self.idp_id, self.protocol_id)
    self.assertEqual(http_client.OK, resp.status_code)
    saml2_authn_request = etree.XML(resp.content)

    relay_state = self._str_from_xml(
        saml2_authn_request, self.ECP_RELAY_STATE)
    sp_consumer_url = self._str_from_xml(
        saml2_authn_request, self.ECP_SERVICE_PROVIDER_CONSUMER_URL)

    # Perform the authn request to the identity provider
    resp = self.saml2_client.send_identity_provider_authn_request(
        saml2_authn_request, self.idp_url, self.username, self.password)
    self.assertEqual(http_client.OK, resp.status_code)
    saml2_idp_authn_response = etree.XML(resp.content)

    idp_consumer_url = self._str_from_xml(
        saml2_idp_authn_response, self.ECP_IDP_CONSUMER_URL)

    # Assert that both saml2_authn_request and saml2_idp_authn_response
    # have the same consumer URL.
    self.assertEqual(sp_consumer_url, idp_consumer_url)

    ...


@testtools.skipUnless(CONF.identity_feature_enabled.federation,
                      "Federated Identity feature not enabled")
def test_request_unscoped_token(self):
    self._request_unscoped_token()
```

Notice that the `test_request_unscoped_token` test only executes if the `federation` feature flag is

enabled.

Note

For each patch submitted upstream, all of the tests from the keystone tempest plugin are executed in the `gate-keystone-dsvm-functional-v3-only-*` job.

5.8 Developing `doctor` checks

As noted in the section above, keystones management CLI provides various tools for administrating OpenStack Identity. One of those tools is called `keystone-manage doctor` and it is responsible for performing health checks about the deployment. If `keystone-manage doctor` detects a symptom, it will provide the operator with suggestions to improve the overall health of the deployment. This section is dedicated to documenting how to write symptoms for `doctor`.

The `doctor` tool consists of a list of symptoms. Each symptom is something that we can check against, and provide a warning for if we detect a misconfiguration. The `doctor` module is located in *keystone.cmd.doctor*. The current checks are based heavily on inspecting configuration values. As a result, many of the submodules within the `doctor` module are named after the configuration section for the symptoms they check. For example, if we want to ensure the `keystone.conf [DEFAULT] max_token_size` option is properly configured for whatever `keystone.conf [token] provider` is set to, we can place that symptom in a module called *keystone.cmd.doctor.tokens*. The symptom will be loaded by importing the `doctor` module, which is done when `keystone-manage doctor` is invoked from the command line. When adding new symptoms, its important to remember to add new modules to the `SYMPTOM_MODULES` list in `keystone.cmd.doctor.__init__`. Doing that will ensure `doctor` discovers properly named symptoms when executed.

Now that we know symptoms are organized according to configuration sections, and how to add them, how exactly do we write a new symptom? `doctor` will automatically discover new symptoms by inspecting the methods of each symptom module (i.e. `SYMPTOM_MODULES`). If a method declaration starts with `def symptom_` it is considered a symptom that `doctor` should check for, and it should be run. The naming of the symptom, or method name, is extremely important since `doctor` will use it to describe what its doing to whoever runs `doctor`. In addition to a well named method, we also need to provide a complete documentation string for the method. If `doctor` detects a symptom, it will use the methods documentation string as feedback to the operator. It should describe why the check is being done, why it was triggered, and possible solutions to cure the symptom. For examples of this, see the existing symptoms in any of `doctor`s symptom modules.

The last step is evaluating the logic within the symptom. As previously stated, `doctor` will check for a symptom if methods within specific symptom modules make a specific naming convention. In order for `doctor` to suggest feedback, it needs to know whether or not the symptom is actually present. We accomplish this by making all symptoms return `True` when a symptom is present. When a symptom evaluates to `False`, `doctor` will move along to the next symptom in the list since. If the deployment isnt suffering for a specific symptom, `doctor` should not suggest any actions related to that symptom (i.e. if you have your cholesterol under control, why would a physician recommend cholesterol medication if you dont need it).

To summarize:

- Symptoms should live in modules named according to the most relevant configuration section they apply to. This ensure we keep our symptoms organized, grouped, and easy to find.

- When writing symptoms for a new section, remember to add the module name to the `SYMPTOM_MODULES` list in `keystone.cmd.doctor.__init__`.
- Remember to use a good name for the symptom method signature and to prepend it with `symptom_` in order for it to be discovered automatically by `doctor`.
- Symptoms have to evaluate to `True` in order to provide feedback to operators.
- Symptoms should have very thorough documentation strings that describe the symptom, side-effects of the symptom, and ways to remedy it.

For examples, feel free to run `doctor` locally using `keystone-manage` and inspect the existing symptoms.

5.9 Making an API Change

This document will guide you through the process of proposing and submitting an API change to keystone.

5.9.1 Prerequisites

In order to follow this tutorial, it is assumed that you have read our *Contributor Documentation* and *Keystone Architecture* documents.

5.9.2 Proposing a change

You need to create a RFE bug report, submit a specification against the keystone-specs repository and bring it up to discussion with the keystone core team for agreement. Please refer to the *guide for proposing features* to learn more about the process.

Create

1. Create a RFE bug report in launchpad;
2. git clone https://opendev.org/openstack/keystone-specs;
3. cp *specs/template.rst specs/backlog/<feature>.rst*;
4. Write the spec based on the template. Ensure the bug link points to the one created in step 1;
5. Also update the documentation at *api/v3/identity-api-v3.rst* to reflect the proposed API changes;
6. Push to gerrit for review;
7. Propose agenda items to the keystone meeting, and make sure someone who understands the subject can attend the meeting to answer questions.

Agreement

The keystone core team will evaluate the specification and vote on accepting it or not. If accepted, the proposal will be targeted to a release; otherwise, the specification will be abandoned.

As soon as there is an agreement on the specification, the change may start rolling out.

5.9.3 Implementing a change

In this section, lets assume that a specification proposing the addition of a *description* field to the roles API was accepted. In the next subsections, you will find a detailed explanation on the needed code changes to the keystone code to implement such change.

Architectural Recapitulation

As you saw in the *Keystone Architecture* document, there are three logical levels of code at which a request passes: the API routing and request handling layer, the resource manager, and the driver.

For the role backend, the API resource can be found under the *keystone/api* directory in the *roles.py* file, and the manager and driver can be found in the *keystone/assignment* directory in the *core.py* and *role_backends/sql.py* files, respectively (currently only the SQL driver is supported).

Changing the SQL Model and Driver

Note

The below guidance is out-of-date and refers to the legacy `migrate_repo` migration repository, which was removed in 21.0.0 (Yoga). Nowadays, for a change like this, you would create an additive or expand migration in the `expand_repo` repository along with null migrations in the `contract_repo` and `data_migration_repo` repositories. For more information, refer to *Database Migrations*.

Todo

Update this section to reflect the new migration model.

First, you need to change the role model to include the description attribute. Go to *keystone/assignment/role_backends/sql.py* and update it like:

```
class RoleTable(sql.ModelBase, sql.ModelDictMixin):

    attributes = ['id', 'name', 'domain_id', 'description']
    description = sql.Column(sql.String(255), nullable=True)
    ...
```

Now, when keystone runs, the table will be created with the new attribute. However, what about existing deployments which already have the role table created? You need to migrate their database schema!

The directory *keystone/common/sql/migrate_repo/versions* owns all the migrations since keystone day 1. Create a new file there with the next migration number. For example, if the latest migration number there is *101*, create yours as *102_add_role_description.py*, which will look like:

```
def upgrade(migrate_engine):
    meta = sql.MetaData()
    meta.bind = migrate_engine

    role_table = sql.Table('role', meta, autoload=True)
    description = sql.Column('description', sql.String(255),
                             nullable=True)
    role_table.create_column(description)
```

Do not forget to add tests for your migration at *keystone/tests/unit/test_sql_upgrade.py*, you may take other tests as example and learn how to develop yours. In this case, you would need to upgrade to *102* check the migration has added the *description* column to the role table.

Changing the role driver itself in *keystone/assignment/role_backends/sql.py* will not be necessary, because the driver handles the role entities as Python dictionaries, thus the new attribute will be handled automatically.

Changing the Manager

Managers handle the business logic. Keystone provides the basic CRUD for role entities, that means that the role manager simply calls the driver with the arguments received from the API resource, and then returns the drivers result back to API resource. Additionally, it handles the cache management.

Thus, there is no manager change needed to make it able to operate role entities with the new *description* attribute.

However, you should add tests for the role CRUD operations with the new attribute to *keystone/tests/unit/assignment/test_core.py*.

When trying to determine whether a change goes in the driver or in the manager, the test is whether the code is business logic and/or needs to be executed for each driver. Both common and business logics go in the manager, while backend specific logic goes in the drivers.

Changing the API Interface

Business logic should not go in the API resource. The API resource should be viewed as a binding between the business logic and the HTTP protocol. Thus, it is in charge of calling the appropriate manager call and wrapping responses into HTTP format.

API resource use JSON schemas do determine whether a provided role is a valid representation or not. Role create and role update schemas are available at *keystone/assignment/schema.py*. You will need to update their properties to include a *description* attribute:

```
_role_properties = {
    'name': parameter_types.name,
    'description': parameter_types.description
}
```

Besides doing the entity validation using such schemas, API resource pass and accept all the attributes to and from the manager. Thus, there is no further change needed at the API resource level.

You should add tests for API unit test to *keystone/tests/unit/test_v3_role.py* and document about the new parameter in the api-ref.

Furthermore, as role entities are passed in the request body to keystone calls, the role routes do not need to be changed; i.e the routes still are:

```
POST /v3/roles
GET /v3/roles/{id}
HEAD /v3/roles/{id}
PATCH /v3/roles/{id}
DELETE /v3/roles/{id}
```

5.9.4 Conclusion

At this point, keystone role entities contain a *description* attribute. In order to make it happen, you have learned how the keystone architecture is, what is the responsibility of each layer, how database migrations occur and the way entities are represented into tables.

The pattern of the change made in this tutorial applies to other keystone subsystems as well, such as *resource* and *identity*.

5.10 Authentication Plugins

Note

This feature is only supported by keystone for the Identity API v3 clients.

Keystone supports authentication plugins and they are specified in the `[auth]` section of the configuration file. However, an authentication plugin may also have its own section in the configuration file. It is up to the plugin to register its own configuration options.

- `methods` - comma-delimited list of authentication plugin names
- `<plugin name>` - specify the class which handles to authentication method, in the same manner as one would specify a backend driver.

Keystone provides three authentication methods by default. `password` handles password authentication and `token` handles token authentication. `external` is used in conjunction with authentication performed by a container web server that sets the `REMOTE_USER` environment variable. For more details, refer to *External Authentication*.

5.10.1 How to Implement an Authentication Plugin

All authentication plugins must extend the *keystone.auth.plugins.base.AuthMethodHandler* class and implement the `authenticate()` method. The `authenticate()` method expects the following parameters.

- `context` - keystones request context
- `auth_payload` - the content of the authentication for a given method
- `auth_context` - user authentication context, a dictionary shared by all plugins. It contains `method_names` and `bind` by default. `method_names` is a list and `bind` is a dictionary.

If successful, the `authenticate()` method must provide a valid `user_id` in `auth_context` and return `None`. `method_name` is used to convey any additional authentication methods in case authentication is for re-scoping. For example, if the authentication is for re-scoping, a plugin must append the previous method names into `method_names`.

If authentication requires multiple steps, the `authenticate()` method must return the payload in the form of a dictionary for the next authentication step.

If authentication is unsuccessful, the `authenticate()` method must raise a *keystone.exception.Unauthorized* exception.

Simply add the new plugin name to the `methods` list along with your plugin class configuration in the `[auth]` sections of the configuration file to deploy it.

If the plugin requires additional configurations, it may register its own section in the configuration file.

Plugins are invoked in the order in which they are specified in the `methods` attribute of the `authentication` request body. If multiple plugins are invoked, all plugins must succeed in order to for the entire authentication to be successful. Furthermore, all the plugins invoked must agree on the `user_id` in the `auth_context`.

The `REMOTE_USER` environment variable is only set from a containing webserver. However, to ensure that a user must go through other authentication mechanisms, even if this variable is set, remove `external` from the list of plugins specified in `methods`. This effectively disables external authentication. For more details, refer to *External Authentication*.

5.11 Database Migrations

Changed in version 21.0.0: (Yoga)

The database migration framework was changed from SQLAlchemy-Migrate to Alembic in the Yoga release. Previously there were three SQLAlchemy-Migrate repos, corresponding to different type of migration operation: the *expand* repo, the *data migration* repo, and the *contract* repo. There are now only two Alembic branches, the *expand* branch and the *contract* branch, and data migration operations have been folded into the former

Changed in version 24.0.0: (Bobcat)

Added support for auto-generation of migrations using the `keystone.common.sql.migrations.manage` script.

Starting with Newton, keystone supports upgrading both with and without downtime. In order to support this, there are two separate branches (all under `keystone/common/sql/migrations`): the *expand* and the *contract* branch.

expand
: For additive schema modifications and triggers to ensure data is kept in sync between the old and new schema until the point when there are no keystone instances running old code.

  May also contain data migrations to ensure new tables/columns are fully populated with data from the old schema.

contract
: Run after all old code versions have been upgraded to running the new code, so remove any old schema columns/tables that are not used by the new version of the code. Drop any triggers added in the expand phase.

A migration script must belong to one branch. If a migration has both additive and destruction operations, it must be split into two migrations scripts, one in each branch.

In order to support rolling upgrades, where two releases of keystone briefly operate side-by-side using the same database without downtime, each phase of the migration must adhere to following constraints:

Expand phase:
: Only additive schema changes, such as new columns, tables, indices, and triggers, and data insertion are allowed.

  Data modification or removal is not allowed.

  Triggers must be created to keep data in sync between the previous release and the next release. Data written by the previous release must be readable by both the previous release and the next release. Data written by the next release must be readable by both the next release and the previous release.

  In cases it is not possible for triggers to maintain data integrity across multiple schemas, writing data should be forbidden using triggers.

Contract phase:
: Only destructive schema changes, such as dropping or altering columns, tables, indices, and triggers, or data modification or removal are allowed.

  Triggers created during the expand phase must be dropped.

5.11.1 Writing your own migrations

Because Keystone uses the expand-contract pattern for database migrations, it is not possible to use the standard `alembic` CLI tool. Instead, Keystone provides its own tool which provides a similar UX to the `alembic` tool but which auto-configures alembic (the library) for this pattern.

To create a new *expand* branch migration:

```
$ tox -e venv -- python -m keystone.common.sql.migrations.manage \
    revision --expand -m "My expand migration"
```

To create a new *contract* branch migration:

```
$ tox -e venv -- python -m keystone.common.sql.migrations.manage \
    revision --contract -m "My contract migration"
```

To auto-generate an *expand* and/or *contract* branch migration:

```
$ tox -e venv -- python -m keystone.common.sql.migrations.manage \
    revision --autogenerate -m "My auto-generated migration"
```

Important

Because of discrepancies between the migrations and models which are yet to be ironed out, a number of columns are intentionally ignored. You can view these by inspecting the `env.py` file in `keystone/common/sql/migrations`.

To view the help page:

```
python -m keystone.common.sql.migrations.manage --help
```

For information on how this tool works, refer to this blog post. For more information on writing migration scripts in general refer to the Alembic documentation.

5.12 Identity entity ID management for domain-specific backends

Keystone supports the option of having domain-specific backends for the identity driver (i.e. for user and group storage), allowing, for example, a different LDAP server for each domain. To ensure that Keystone can determine to which backend it should route an API call, starting with Juno, the identity manager will, provided that *domain-specific backends* are enabled, build on-the-fly a persistent mapping table between Keystone Public IDs that are presented to the API and the domain that holds the entity, along with whatever local ID is understood by the driver. This hides, for instance, the LDAP specifics of whatever ID is being used.

To ensure backward compatibility, the default configuration of either a single SQL or LDAP backend for Identity will not use the mapping table, meaning that public facing IDs will be the unchanged. If keeping these IDs the same for the default LDAP backend is not required, then setting the configuration variable `backward_compatible_ids` to `False` will enable the mapping for the default LDAP driver, hence hiding the LDAP specifics of the IDs being used.

5.13 Learning Architecture Internals

5.13.1 Caching Layer

The caching layer is designed to be applied to any `manager` object within Keystone via the use of `keystone.common.cache` module. This leverages oslo.cache caching system to provide a flexible caching backend.

The caching can be setup for all or some subsystems. It is recommended that each of the managers have an independent toggle within the config file to enable caching. The easiest method to utilize the toggle within the configuration file is to define a `caching` boolean option within that managers configuration section (e.g. `identity`). Enable the global cache `enabled` option as well as the specific managers caching enable toggle in order to cache that subsystem.

The oslo.cache is simple and easy to adopt by any system. See the usage guide of it. There are various cache *backends* supported by it. Example use of oslo.cache in keystone (in this example, `token` is the

manager):

```
from keystone.common import cache

TOKENS_REGION = cache.create_region(name='tokens')
MEMOIZE_TOKENS = cache.get_memoization_decorator(
    group='token',
    region=TOKENS_REGION)

@MEMOIZE_TOKENS
def _validate_token(self, token_id):
    ...
    return token
```

With the above example, each call to the `cacheable_function` would check to see if the arguments passed to it matched a currently valid cached item. If the return value was cached, the caching layer would return the cached value; if the return value was not cached, the caching layer would call the function, pass the value to the `MEMOIZE_TOKEN` decorator, which would then determine if caching was globally enabled and enabled for the `token` manager. If either caching toggle is disabled, the value is returned but not cached.

It is recommended that each of the managers have an independent configurable time-to-live (TTL). The option `cache_time` is to be set for every manager under its section in keystone.conf file. If the `cache_time` is set to `None`, the expiration time will be set to the global default `expiration_time` option in the `[cache]` configuration section. These options are passed to and handled by oslo.cache.

Cache invalidation can be done if specific cache entries are changed. Example of invalidating a cache (in this example, `token` is the manager):

```
def invalidate_individual_token_cache(self, token_id):
    ...
    self._validate_token.invalidate(self, token_id)
```

For cache invalidation, there is an `invalidate` method (attribute) on the decorated function. To invalidate the cache, pass the same arguments to the `invalidate` method as you would the normal function. This means you need to pass `self` as the first argument.

5.13.2 Filtering responsibilities between API resources and drivers

Keystone supports the specification of filtering on list queries as part of the v3 identity API. By default these queries are satisfied in the API resource when it calls the `wrap_collection` method at the end of a `get` method. However, to enable optimum performance, any driver can implement some or all of the specified filters (for example, by adding filtering to the generated SQL statements to generate the list).

The communication of the filter details between the API resource and its drivers is handled by the passing of a reference to a Hints object, which is a list of dicts describing the filters. A driver that satisfies a filter must delete the filter from the Hints object so that when it is returned to the API, it knows to only execute any unsatisfied filters.

The contract for a driver for `list_{entity}` methods is therefore:

- It MUST return a list of entities of the specified type

- It MAY either just return all such entities, or alternatively reduce the list by filtering for one or more of the specified filters in the passed Hints reference, and removing any such satisfied filters. An exception to this is that for identity drivers that support domains, then they should at least support filtering by domain_id.

5.13.3 Entity list truncation by drivers

Keystone supports the ability for a deployment to restrict the number of entries returned from list operations, typically to prevent poorly formed searches (e.g. without sufficient filters) from becoming a performance issue.

These limits are set in the configuration file, either for a specific driver or across all drivers. A global `list_limit` set in `[DEFAULT]` section of keystone is considered in case no limit is set for specific driver. These limits are read at the Manager level and passed into individual drivers as part of the Hints list object. A driver should try and honor any such limit if possible, but if it is unable to do so then it may ignore it (and the truncation of the returned list of entities will happen at the API level by `wrap_collection` method).

5.14 Keystone for Other Services

This document provides a summary of some things that other services need to know about how keystone works, and specifically about how they can take advantage of the v3 API. The v3 API was introduced as a stable API in the Grizzly release.

5.14.1 Glossary

Authentication
: The process of determining if a user is who they claim to be (authN).

Authorization
: The process of determining if a user can do what they are requesting (authZ).

Scope
: A specific operating context. This is commonly used when describing the authorization a user may have. For example, a user with a role assignment on a project can get a token scoped to that project, ultimately operating within that projects scope.

System
: An assignment target that refers to a collection of API services as a whole. Users and groups can be granted authorization on the *deployment system*.

Service
: OpenStack services like identity, compute, image, etc.

Domain
: A container for users, projects, and groups. A domain is also an assignment target for users and groups. Its possible for users and groups to have authorization on domains outside of the domain associated to their reference.

Project
: A container and a namespace for resources isolated within OpenStack. A user, or group of users, must have a role assignment on a project in order to interact with it.

Token
A self-service resource that proves a users identity and authentication. It can optionally carry a users authorization, allowing them to interact with OpenStack services.

Role
A string that represents one or more permissions or capabilities.

Role Assignment
An association between an actor and a target that results in authorization. Actors can be users or groups of users. Targets can be projects, domains, or the deployment system itself.

User
A entity modeling an end-user of the system.

Group
A container for users. Users indirectly inherit any authorization the group has on projects, domains, or the system.

5.14.2 Domains

A major new feature in v3 is domains. Every project, user, and user group is owned by a domain (reflected by their `domain_id` value) which provides them their own namespace. For example, unlike in v2.0, usernames are no longer unique across the deployment. You can have two users with the same name, but they must be in different domains. However, user IDs are assigned to users by keystone and are expected to be unique across the deployment. All of this logic applies to projects, user groups and roles.

One of the great things about domains is that you can have one domain backed by SQL (for service users) and another backed by LDAP (the cloud is deployed into existing infrastructure).

5.14.3 The default domain

Note

The v2.0 API has been removed as of the Queens release. While this section references the v2.0 API, it is purely for historical reasons that clarify the existence of the *default* domain.

Domains were introduced as a v3-only feature. As a result, the v2.0 API didnt understand the concept of domains. To allow for both versions of the Identity API to run side-by-side, the idea of a *default* domain was established.

The *default* domain was a domain that was guaranteed to exist and was created during the `keystone-manage db_sync` process. By default, the domain ID is `default` and the name is `Default`, but it is possible to change these values through keystones configuration file. The v2.0 API would consider users and projects existing within that domain as valid, but it would never expose domain information through the API. This allowed the v2.0 API to operate under the assumption that everything within the *default* domain was accessible. This was crucial in avoiding namespace conflicts between v2.0 and v3 where multiple domains existed. Using v3 allowed deployers the ability to experiment with domains, while isolating them from the v2.0 API.

As far as the v3 API is concerned, the *default* domain is simply a domain and doesnt carry any special connotation like it did with v2.0.

5.14.4 Authorization Scopes

End users use the Identity API as a way to express their authoritative power to other OpenStack services. This is done using tokens, which can be scoped to one of several targets depending on the users role assignments. This is typically referred to as a tokens *scope*. This happens when a user presents credentials, in some form or fashion, to keystone in addition to a desired scope. If keystone can prove the user is who they say they are (authN), it will then validate that the user has access to the scope they are requesting (authZ). If successful, the token response will contain a token ID and data about the transaction, such as the scope target and role assignments. Users can use this token ID in requests to other OpenStack services, which consume the authorization information associated to that token to make decisions about what that user can or cannot do within that service.

This section describes the various scopes available, and what they mean for services consuming tokens.

System Scope

A *system-scoped* token implies the user has authorization to act on the *deployment system*. These tokens are useful for interacting with resources that affect the deployment as a whole, or exposes resources that may otherwise violate project or domain isolation.

Good examples of system-scoped resources include:

- Services: Service entities within keystone that describe the services deployed in a cloud.
- Endpoints: Endpoints that tell users where to find services deployed in a cloud.
- Hypervisors: Physical compute infrastructure that hosts instances where the instances may, or may not, be owned by the same project.

Domain Scope

A *domain-scoped* token carries a users authorization on a specific domain. Ideally, these tokens would be useful for listing resources aggregated across all projects with that domain. They can also be useful for creating entities that must belong to a domain. Users and groups are good examples of this. The following is an example of how a domain-scoped token could be used against a service.

Assume a domain exists called *Foo*, and it contains projects called *bar* and *baz*. Lets also assume both projects contain instances running a workload. If Alice is a domain administrator for *Foo*, she should be able to pass her domain-scoped token to nova and ask for a list of instances. If nova supports domain-scoped tokens, the response would contain all instances in projects *bar* and *baz*.

Another example of using a domain-scoped token would be if Alice wanted to create a new project in domain *Foo*. When Alice sends a request to create a new project (*POST /v3/projects*), keystone should ensure the new project is created within the *Foo* domain, since thats the authorization associated to Alices token.

Warning

This behavior isnt completely implemented, and is still in progress. This example describes the ideal behavior, specifically for developers looking to implement scope into their APIs.

Project Scope

A *project-scoped* token carries the role assignments a user has on a project. This type of scope is great for managing resources that fit nicely within project boundaries. Good examples of project-level resources that can be managed with project-scoped tokens are:

- Instances: Virtual compute servers that require a project association in order to be created.
- Volumes: Storage devices that can be attached to instances.

Unscoped

An *unscoped* token is a token that proves authentication, but doesnt carry any authorization. Users can obtain unscoped tokens by simply proving their identity with credentials. Unscoped tokens can be exchanged for any of the various scoped tokens if a user has authorization on the requested scope.

An example of where unscoped tokens are specifically useful is when users perform federated authentication. First, a user will receive an unscoped token pending successful federated authentication, which they can use to query keystone for a list of projects theyre allowed to access. Then they can exchange their unscoped token for a project-scoped token allowing them to perform actions within a particular project.

5.14.5 Why are authorization scopes important?

Flexibility for exposing your work

OpenStack provides a rich set of APIs and functionality. We wrote some APIs with the intent of managing the deployment hardware, otherwise referred to as the deployment system. We wrote others to orchestrate resources in a project or a domain. Some APIs even operate on multiple levels. Since we use tokens to authorize a users actions against a given service, they needed to handle different scope targets. For example, when a user asks for a new instance, we expect that instance to belong to a project; thus we expect a project relayed through the tokens scope. This idea is fundamental in providing isolation, or tenancy, between projects in OpenStack.

Initially, keystone only supported the ability to generate project-scoped tokens as a product of a user having a role assignment on a project. Consequently, services had no other choice but to require project-scoped tokens to protect almost all of their APIs, even if that wasnt an ideal option. Using project-scoped tokens to protect APIs they werent designed to protect required operators to write custom policy checks to secure those APIs. An example showcases this more clearly.

Lets assume an operator wanted to create a read-only role. Users with the *reader* role would be able to list things owned by the project, like instances, volumes, or snapshots. The operator also wants to have a read-only role for fellow operators or auditors, allowing them to view hypervisor information or endpoints and services. Reusing the existing *reader* role is difficult because users with that role on a project shouldnt see data about hypervisors, which would violate tenancy. Operators could create a new role called *operator* or *system-reader*, but then those users would still need to have that role assigned on a project to access deployment-level APIs. The concept of getting project-scoped tokens to access deployment-level resources makes no sense for abstractions like hypervisors that cannot belong to a single project. Furthermore, this requires deployers to maintain all of this in policy files. You can quickly see how only using project-scope limits our ability to protect APIs without convoluted or expensive-to-maintain solutions.

Each scope offered by keystone helps operators and users avoid these problems by giving you, the developer, multiple options for protecting APIs you write, instead of the one-size-fits-all approach we outgrew.

You no longer have to hope an operator configures policy correctly so their users can consume the feature you wrote. The more options you have for protecting an API, the easier it is to provide default policies that expose more of your work to users safely.

Less custom code

Another crucial benefit of authorization scopes offered by keystone is less custom code. For example, if you were writing an API to manage a deployment-level resource but only allowed to consume project-scoped tokens, how would you determine an operator from an end user? Would you attempt to standardize a role name? Would you look for a unique project in the tokens scope? Would these checks be configurable in policy or hardcoded in your service?

Chances are, different services will come up with different, inconsistent solution for the same problem. These inconsistencies make it harder for developers to context switch between services that process things differently. Users also suffer from inconsistencies by having to maintain a mental mapping of different behavior between services. Having different scopes at your disposal, through keystone tokens, lets you build on a standard solution that other projects also consume, reducing the likelihood of accidentally developing inconsistencies between services. This commonality also gives us a similar set of terms we can use when we communicate with each other and users, allowing us to know what someone means by a *system-admin* and how that is different from a *project-admin*.

Reusable default roles

When OpenStack services originally started developing a policy enforcement engine to protect APIs, the only real concrete role we assumed to be present in the deployment was a role called *admin*. Because we assumed this, we were able to write policies with *admin* as the default. Keystone also took steps to ensure it had a role with that name during installation. While making this assumption is beneficial for some APIs, having only one option is underwhelming and leaves many common policy use cases for operators to implement through policy overrides. For example, a typical ask from operators is to have a read-only role, that only allows users with that role on a target to view its contents, restricting them from making writable changes. Another example is a membership role that isnt the administrator. To put it clearly, a user with a *member* role assignment on a project may create new storage volumes, but theyre unable to perform backups. Users with the *admin* role on a project can access the backups functionality.

Keep in mind, the examples above are only meant to describe the need for other roles besides *admin* in a deployment. Service developers should be able to reuse these definitions for similar APIs and assume those roles exist. As a result, keystone implemented support for ensuring the *admin*, *member*, and *reader* roles are present during the installation process, specifically when running `keystone-manage bootstrap`. Additionally, keystone creates a relationship among these roles that make them easier for service developers to use. During creation, keystone implies that the *admin* role is a superset of the *member* role, and the *member* role is a superset of the *reader* role. The benefit may not be obvious, but what this means is that users with the *admin* role on a target also have the *member* and *reader* roles generated in their token. Similarly, users with the *member* role also have the *reader* role relayed in their token, even though they dont have a direct role assignment using the *reader* role. This subtle relationship allows developers to use a short-hand notation for writing policies. The following assumes `foobar` is a project-level resource available over a service API and is protected by policies using generic roles:

```
"service:foobar:get": "role:admin OR role:member OR role:reader"
"service:foobar:list": "role:admin OR role:member OR role:reader"
"service:foobar:create": "role:admin OR role:member"
```

(continues on next page)

(continued from previous page)

```
"service:foobar:update": "role:admin OR role:member"
"service:foobar:delete": "role:admin"
```

The following policies are functionally equivalent to the policies above, but rely on the implied relationship between the three roles, resulting in a simplified check string expression:

```
"service:foobar:get": "role:reader"
"service:foobar:list": "role:reader"
"service:foobar:create": "role:member"
"service:foobar:update": "role:member"
"service:foobar:delete": "role:admin"
```

In addition to above roles, from 2023.2 (Bobcat) release `keystone-manage bootstrap` will provide *service* role as well. If a `service` role is already present in the deployment, then a new one is not created. This way any local scripts relying on the role ID will not be broken.

Note

If you already have a `service` role in your deployment, you should review its usage to make sure it is used only for service-to-service communication.

Once `service` role is created, OpenStack service developers can start integrating it into their default policies as expressed:

```
policy.DocumentedRuleDefault(
    name='os_compute_api:os-server-external-events:create',
    check_str='role:service',
    scope_types=['project']
)
```

It is important to note that we need to keep all the service-to-service APIs default to `service` role only. For example, a policy that requires `service` can be expressed as:

```
"service:foobar:create": "role:service"
```

There might be exception service-to-service APIs which project think are useful to be used by admin or non-admin user then they can take the exceptional decision to default them to user role and `service` role. For example, a policy that requires `service` and `admin` can be expressed as:

```
"service:foobar:create": "role:service" OR "role:admin"
```

Additionally, any deployment tools that create service accounts for OpenStack services, should start preparing for these policy changes by updating their role assignments and performing the deployment language equivalent of the following:

```
$ openstack role add --user nova --project service service
$ openstack role add --user cinder --project service service
$ openstack role add --user neutron --project service service
$ openstack role add --user glance  --project service service
$ openstack role add --user manila  --project service service
```

5.14.6 How do I incorporate authorization scopes into a service?

Now that you understand the advantages of a shared approach to policy enforcement, the following section details the order of operations you can use to implement it in your service.

Ruthless Testing

Policy enforcement implementations vary greatly across OpenStack services. Some enforce authorization near the top of the API while others push the logic deeper into the service. Differences and intricacies between services make testing imperative to adopt a uniform, consistent approach. Positive and negative protection testing helps us assert users with specific roles can, or cannot, access APIs. A protection test is similar to an API, or functional test, but purely focused on the authoritative outcome. In other words, protection testing is sufficient when we can assert that a user is or isnt allowed to do or see something. For example, a user with a role assignment on project *foo* shouldnt be able to list volumes in project *bar*. A user with a role on a project shouldnt be able to modify entries in the service catalog. Users with a *reader* role on the system, a domain, or a project shouldnt be able to make writable changes. You commonly see protection tests conclude with an assertion checking for a successful response code or an HTTP 403 Forbidden.

If your service has minimal or non-existent protection coverage, you should start by introducing tests that exercise the current default policies, whatever those are. This step serves three significant benefits.

First, it puts us in the shoes of our users from an authorization perspective, allowing us to see the surface of the API a user has access to with a given assignment. This information helps audit the API to make sure the user has all the authorization to do what they need, but nothing more. We should note inconsistencies here as feedback that we should fix, especially since operators are probably attempting to fix these inconsistencies through customized policy today.

Second, a collection of protection tests make sure we dont have unwanted security-related regressions. Imagine making a policy change that introduced a regression and allowed a user to access an API and data they arent supposed to see. Conversely, imagine a patch that accidentally tightened restriction on an API that resulted in a broken workflow for users. Testing makes sure we catch cases like this early and handle them accordingly.

Finally, protection tests help us use test-driven development to evolve policy enforcement. We can make a change and assert the behavior using tests locally, allowing us to be proactive and not reactive in our authoritative business logic.

To get started, refer to the oslo.policy documentation that describes techniques for writing useful protection tests. This document also describes some historical context you might recognize in your service and how you should deal with it. You can also look at protection tests examples in other services, like keystone or cinder. Note that these examples test the three default roles provided from keystone (reader, member, and admin) against the three scopes keystone offers, allowing for nine different personas without operators creating roles specific to their deployment. We recommend testing these personas where applicable in your service:

- project reader
- project member
- project admin
- system reader
- system member

- system admin
- domain reader
- domain member
- domain admin

Auditing the API

After going through the API and adding protection tests, you should have a good idea of how each API is or isnt exposed to end users with different role assignments. You might also have a list of areas where policies could be improved. For example, maybe you noticed an API in your service that consumes project-scoped tokens to protect a system-level resource. If your service has a bug tracker, you can use it to document these gaps. The keystone team went through this exercise and used bugs. Feel free to use these bug reports as a template for describing gaps in policy enforcement. For example, if your service has APIs for listing or getting resources, you could implement the reader role on that API.

Setting scope types

With testing in place and gaps documented, you can start refactoring. The first step is to start using oslo.policy for scope checking, which reduces complexity in your service by having a library do some lifting for you. For example, if you have an API that requires a project-scoped token, you can set the scope of the policy protecting that API accordingly. If an instance of `RuleDefault` has scope associated to it, oslo.policy checks that it matches the scope of the token used to make the request. This behavior is configurable, allowing operators to turn it on once all policies have a scope type and once operators have audited their assignments and educated their users on how to get the scope necessary to access an API. Once that happens, an operator can configure oslo.policy to reject requests made with the wrong scope. Otherwise, oslo.policy logs a warning for operators that describes the mismatched scope.

The oslo.policy library provides documentation for setting scope. You can also see keystone examples or placement examples of setting scope types on policies.

If you have difficulty deciding which scope an API or resource requires, try thinking about the intended user. Are they an operator managing the deployment? Then you might choose *system*. Are they an end user meant to operate only within a given project? Then *project* scope is likely what you need. Scopes arent mutually exclusive.

You may have APIs that require more than one scope. Keystones user and project APIs are good examples of resources that need different scopes. For example, a system administrator should be able to list all users in the system, but domain administrators should only be able to list users within their domain. If you have an API that falls into this category, you may be required to implicitly filter responses based on the scope type. If your service uses oslo.context and keystonemiddleware, you can query a *RequestContext* object about the tokens scope. There are keystone patches that show how to filter responses according to scope using oslo.context, in case you need inspiration.

If you still cant seem to find a solution, dont hesitate to send a note to the OpenStack Discuss mailing list tagged with *[keystone]* or ask in #openstack-keystone on IRC.

Rewriting check string

With oslo.policy able to check scope, you can start refactoring check strings where-ever necessary. For example, adding support for default roles or removing hard-coded `is_admin: True` checks. Remember that oslo.policy provides deprecation tooling that makes upgrades easier for operators. Specifically, upgrades are made easier by combining old defaults or overrides with the new defaults using a logical *OR*. We encourage you to use the available deprecation tooling when you change policy names or check strings. You can refer to examples that show you how to build descriptive rule objects using all the default roles from keystone and consuming scopes.

Communication

Communicating early and often is never a bad thing, especially when a change is going to impact operators. At this point, its crucial to emphasize the changes youve made to policy enforcement in your service. Release notes are an excellent way to signal changes to operators. You can find examples when keystone implemented support for default roles. Additionally, you might have operators or users ask questions about the various scopes or what they mean. Dont hesitate to refer them to keystones *scope documentation*.

5.14.7 Auth Token middleware

The `auth_token` middleware handles token validation for the different services. Conceptually, what happens is that `auth_token` pulls the token out of the `X-Auth-Token` request header, validates the token using keystone, produces information about the identity (the API user) and authorization context (the project, roles, etc) of the token, and sets environment variables with that data. The services typically take the environment variables, put them in the services context, and use the context for policy enforcement via `oslo.policy`.

Service tokens

Service tokens are a feature where the `auth_token` middleware will also accept a service token in the `X-Service-Token` header. It does the same thing with the service token as the user token, but the results of the token are passed separately in environment variables for the service token (the service user, project, and roles). If the service knows about these then it can put this info in its context and use it for policy checks. For example, assuming theres a special policy rule called `service_role` that works like the `role` rule except checks the service roles, you could have an `oslo.policy` rule like `service_role:service and user_id:%(user_id)s` such that a service token is required along with the user owning the object.

5.14.8 Picking the version

Use version discovery to figure out what version the identity server supports rather than configuring the version. This will make it easier to adopt new API versions as they are implemented.

For information about how to accomplish service discovery with the keystoneauth library, please see the documentation.

5.14.9 Hierarchical Multitenancy

This feature is specific to v3 and allows projects to have parents, siblings, and children relationships with other projects.

Tokens scoped to projects in a hierarchical structure wont contain information about the hierarchy in the token response. If the service needs to know the hierarchy it should use the v3 API to fetch the hierarchy.

5.15 Developing Keystone Drivers

A driver, also known as a backend, is an important architectural component of Keystone. It is an abstraction around the data access needed by a particular subsystem. This pluggable implementation is not only how Keystone implements its own data access, but how you can implement your own!

Each major subsystem (that has data access needs) implements the data access by using drivers. Some examples of Keystones drivers:

- `keystone.identity.backends.ldap.Identity`
- *`keystone.token.providers.fernet.core.Provider`*
- `keystone.contrib.federation.backends.sql.Federation`

5.15.1 In/Out of Tree

Its best to start developing your custom driver outside of the Keystone development process. This means developing it in your own public or private git repository and not worrying about getting it upstream (for now).

This is better for you because it gives you more freedom and you are not bound to the strict OpenStack development rules or schedule. You can iterate faster and take whatever shortcuts you need to get your product out of the door.

This is also good for Keystone because it will limit the amount of drivers that must be maintained by the team. If the team had to maintain a driver for each NoSQL DB that deployers want to use in production there would be less time to make Keystone itself better. Not to mention that the team would have to start gaining expertise in potentially dozens of new technologies.

As youll see below there is no penalty for open sourcing your driver, on GitHub for example, or even keeping your implementation private. We use Setuptools entry points to load your driver from anywhere in the Python path.

5.15.2 How To Make a Driver

The TLDR; steps (and too long didnt write yet):

1. Determine which subsystem you would like write a driver for
2. Subclass the most current version of the driver interface
3. Implement each of the abstract methods for that driver
    a. We are currently not documenting the exact input/outputs of the driver methods. The best approach right now is to use an existing driver as an example of what data your driver will receive and what data your driver will be required to return.

b. There is a plan in place to document these APIs in more detail.

4. Register your new driver as an entry point
5. Configure your new driver in `keystone.conf`
6. Sit back and enjoy!

5.15.3 Identity Driver Configuration

As described in the *Domain-specific configuration* there are 2 ways of configuring domain specific drivers: using files and using database. Configuration with files is straight forward but is having a major disadvantage of requiring restart of Keystone for the refresh of configuration or even for Keystone to start using chosen driver after adding a new domain.

Configuring drivers using database is a flexible alternative that allows dynamic reconfiguration and even changes using the API (requires admin privileges by default). There are 2 independent parts for this to work properly:

Defining configuration options

Driver class (as pointed by EntryPoints) may have a static method *register_opts* accepting *conf* argument. This method, if present, is being invoked during loading the driver and registered options are then available when the driver is being instantiated.

```
class CustomDriver(base.IdentityDriverBase):

    @classmethod
    def register_opts(cls, conf):
        grp = cfg.OptGroup("foo")
        opts = [cfg.StrOpt("opt1")]
        conf.register_group(grp)
        conf.register_opts(opts, group=grp)

    def __init__(self, conf=None):
        # conf contains options registered above and domain specific values
        # being set.
        pass

    ...
```

Allowing domain configuration per API

A safety measure of the Keystone domain configuration API is that options allowed for the change need to be explicitly whitelisted. This is done in the *domain_config* section of the main Keystone configuration file.

```
[domain_config]
additional_whitelisted_options=<GROUP_NAME>:[opt1,opt2,opt3]
additional_sensitive_options=<GROUP_NAME>:[password]
```

The <*GROUP_NAME*> is the name of the configuration group as defined by the driver. Sensitive options are not included in the GET api call and are stored in a separate database table.

5.15.4 Driver Interface Changes

We no longer support driver versioning. Thus, if a driver interface changes, you will need to upgrade your custom driver to meet the new driver contract.

Removing Methods

Newer driver interfaces may remove methods that are currently required. Methods are removed when they are no longer required or invoked by Keystone. There is no reason why methods removed from the Keystone interface need to be removed from custom drivers.

Adding Methods

The most common API changes will be adding methods to support new features. The new method must be implemented by custom driver implementations.

Updating Methods

We will do our best not to update existing methods in ways that will break custom driver implementations. However, if that is not possible, again you will need to upgrade your custom driver implementation to meet the new driver contract.

5.16 Service Catalog Overview

The OpenStack keystone service catalog allows API clients to dynamically discover and navigate to cloud services. The service catalog may differ from deployment-to-deployment, user-to-user, and project-to-project.

The service catalog is the first hurdle that API consumers will need to understand after successfully authenticating with Keystone, making it a critical focal point for the overall user experience of OpenStack.

If youre integrating your OpenStack service with Keystone, then please follow the guidelines provided below.

If youre writing an OpenStack client, hopefully this helps you navigate the service catalog that youre being presented so that you can quickly move on to the business of consuming cloud services.

5.16.1 An example service catalog

The following is an example service catalog. It actually excludes several common attributes such as `id`, which are of no concern to end users, `region_id`, which are a bit out of scope for this topic, and `enabled`, which is always `true` for end users.

This service catalog contains just one service, Keystone, which is accessible via a single endpoint URL:

```
{
    "catalog": [
        {
            "name": "Keystone",
            "type": "identity",
            "endpoints": [
                {
                    "interface": "public",
                    "url": "https://identity.example.com:5000/"
                }
            ]
        }
    ]
}
```

The service catalog itself may appear in a token creation response (`POST /v3/auth/tokens`), a token validation response (`GET /v3/auth/tokens`), or as a standalone resource (`GET /v3/auth/catalog`).

5.16.2 Services

The service catalog itself is composed of a list of services.

Service entities represent web services in the OpenStack deployment. A service may have zero or more endpoints associated with it, although a service with zero endpoints is essentially useless in an OpenStack configuration.

In addition to the related endpoints, there are two attributes of services that important to end users:

- `name` (string): user-facing name of the service

This attribute is not intended to be machine-parseable or otherwise meaningful beyond branding or name-recognition for end users. Logical values might include Keystone or maybe Brand X Public Cloud Identity Service. Deployers should be free to rename, and therefore rebrand, a service at will.

- `type` (string): describes the API implemented by the service. To support future projects, the value should not be validated against a list.

An OpenStack-wide effort to standardize service types has been done outside of Keystone and is known as the service-types authority.

This should not convey the version of the API implemented by the service (as in Cinders `volumev2` service type) because both the `volume` service and `volumev2` service provide block storage as a service which is what the service type is meant to convey. The underlying implementation is completely irrelevant here.

In the general case, there should only be one service in a deployment per service type, although Keystone does not enforce this today.

5.16.3 Endpoints

Each service should have one or more related endpoints. An endpoint is essentially a base URL for an API, along with some metadata about the endpoint itself and represents a set of URL endpoints for OpenStack web services.

- `interface` (string): describes the visibility of the endpoint according to one of three values (`public`, `internal`, and `admin`)

`public` endpoints are intended for consumption by end users or other service users, generally on a publicly available network interface.

`internal` endpoints are intended for consumption by end users, generally on an unmetered internal network interface.

`admin` endpoints are intended only for consumption by those needing administrative access to the service, generally on a secure network interface.

You might also think of each interface value as the result of a matrix of use cases:

- **Public API** on a **public network**: use a `public` interface.
- **Public API** on an **internal network**: use an `internal` interface.
- **Privileged API** on a **public network**: unsupported! Use access controls on your `public` endpoint instead.
- **Privileged API** on an **internal network**: `admin` interface, but use access controls on your `public` endpoint instead. The notion of a privileged API endpoint makes security-conscious developers instantly lazy (security becomes someone elses problem), and is an obvious attack vector if someone were to infiltrate your internal network. It also adds more complexity to your API architecture which makes documentation, testing, and API evolution that much more difficult.
- `url` (string): fully qualified URL of the service endpoint

This should be unversioned base URL for an API. Good examples include `https://identity.example.com:5000/` and `https://keystone.example.com/`.

Conversely, `https://identity.example.com:5000/v3/` is an unfortunate example because it directs all clients to connect to a versioned endpoint, regardless of which API versions they understand. This makes it hard for services to do any sort of API versioning, and for clients to dynamically discover additional available versions.

For a period of time, keystone was stuck in a position where it implements a `/v3/` API, but for backwards compatibility with existing v2 clients, was forced to continue advertising the `/v2.0/` endpoint in the service catalog until it was reasonable to assume that all clients in the ecosystem are capable of handling an unversioned URL. As a side effect, this has had a tremendous impact on the awareness of, and thus adoption of, Keystones Identity API v3 (which has been enabled by default and stable since the 2013.1 Grizzly release). Dont put your project in that position!

Similarly, `https://object-store.example.com/v1/KEY_\$(project_id)s` (which would ultimately be rendered to clients as a project-specific URL, such as `https://object-store.example.com/v1/KEY_d12af07f4e2c4390a21acc31517ebec9`) is an unfortunate example because not only does it hardcode an API version as in the above example, but it also exposes the clients project ID directly to the client. Instead, the operational scope or a request can be determined by inspecting the users token or consuming values populated by `keystonemiddleware.auth_token`. Its also far less cacheable than a URL that is neither project nor user specific, which is important given that every client needs access to consume the service catalog prior to nearly every API request.

5.17 Technical Vision for Keystone

This document is a self-evaluation of keystone with regard to the Technical Committees technical vision and serves as a basis for guiding the mission of the keystone project. The objectives captured here are what the keystone team strives to build. New features and design changes should be compared with this document before being embarked upon. When such proposals are not in alignment, propose a change to this document or to the overall technical vision to initiate a discussion on the renewed vision for the project.

5.17.1 Mission Statement

Keystones mission is to provide secure, resilient, and user-friendly discovery, authentication, and authorization for multitenant services.

5.17.2 Vision for OpenStack

Self-service

Keystone needs to strive to provide a flexible and simple mechanism to expose OpenStack functionality safely and securely in a multi-tenant environment, to enable a true self-service experience for end users in a shared-resource system.

Application Control

Keystone provides the ability for applications to have their own identity through *application credentials*, in service of developers building applications that need to access cloud APIs and cloud-native applications.

Interoperability

Keystone strives for a completely seamless experience for end users and applications running on multiple clouds. Initiatives in service of providing such a consistent user experience include providing a discovery mechanism for available functionality, eliminating optional API extensions, and providing useful default roles which eliminate the need for inconsistently-named, operator-defined roles for similar access levels between clouds. Keystone is also capable of itself acting as a bridge between separate clouds through its Keystone-to-Keystone federated authentication functionality.

Bidirectional Compatibility

To support clients operating across multiple clouds of potentially different versions, changes in keystones major API are additive-only, and updates to the API are signaled by the minor version number, which allows clients to discover, to a reasonable degree, what capabilities are available in the keystone version they are connecting to. Keystone also provides a JSON-home document to aid clients in discovering the availability and status of features. Enhancements to the discoverability of keystones APIs are a priority.

Partitioning

Keystones service catalog mechanism makes it possible for users to have authorization for resources in geographically distributed regions, and keystones various mechanisms for distributed authentication, such as using a distributed database or LDAP identity backend, using an external authentication source, or federating keystone itself to provide distributed identity providers, support geographically distributed computing. Keystone hopes to create a consistent user story and reference architecture for large-scale distributed deployments, including edge-computing use cases.

Basic Physical Data Center Management

In support of OpenStack being primarily a data center management tool, keystone should always work out of the box and not rely on the pre-existence of another identity management system in the data center. In practice this means always continuing to support a SQL storage backend for user data.

Plays Well With Others

Keystone encourages its use outside of an OpenStack environment. In support of this, keystone supports a standard authentication token format (JWT) that can be understood by many applications, and seeks to support full Single-Sign-On functionality that can be used in front of any web application.

Customizable Integration

In service of supporting customizable integration both between OpenStack services and from client applications, keystone has an ongoing mission to fulfill the Principle of Least Privilege and permit the cloud consumer to delegate only the minimum permissions needed to an application. Keystone works to provide this both through reforming OpenStack policy to make it easier to manage across services, and by providing new mechanisms such as application credential access rules to allow users to restrict capabilities of applications to a subset of service APIs.

Graphical User Interface

Keystone does not provide a graphical user interface, but must always be mindful of how its APIs will be presented in dashboards. For some features, such as Single-Sign-On authentication, keystone may provide its own graphical user interface in order to provide a smooth web-login experience without requiring a dependency on another dashboard.

Secure by Design

Keystone strives to be secure by design, by making opinionated choices about the default security configuration. Making it easier to administer fine-grained access control in support of the Principle of Least Privilege is an ongoing effort.

5.18 Programming Exercises for Interns and New Contributors

The keystone team participates in open source internship programs such as Outreachy and Google Summer of Code and welcomes contributions from students and developers of all skill levels. To help with formal applications for work programs or to give casual contributors a taste of what working on keystone is like, weve created a few exercises to showcase what we think are valuable development skills.

These exercises are samples, and code produced to solve them should most likely not be merged into keystone. However, you should still propose them to Gerrit to get practice with the code review system and to get feedback from the team. This is a good way to get used to the development workflow and get acquainted with the benefits of working in a collaborative development environment. Also feel free to *talk to the keystone team* to get help with these exercises, and refer to the *contributor documentation* for more context on the architecture and contributing guidelines for keystone.

The exercises provide some ideas of what you can do in keystone, but feel free to get creative.

5.18.1 Add a Parameter to an API

Add a string parameter named `nickname` to the Project API. The end result will be that you can use the new parameter when you create a new project using the POST /v3/projects API, update the parameter using the PATCH /v3/projects/{project_id} API, and the value displayed using the GET /v3/projects/{project_id}.

Refer to the *API Change tutorial*. In short, you will need to follow these steps:

1. Create a SQL migration to add the parameter to the database table (`keystone.common.sql.migrations.versions`)
2. Add a SQL migration unit test (*keystone/tests/unit/test_sql_upgrade.py*)
3. Add the parameter to the SQL model for projects (`keystone.resource.backends.sql`)
4. Add unit tests (*keystone/tests/unit/resource/test_backend.py*) for the manager (`keystone.resource.core`) to show that the project can be created and updated with the new parameter using the provider mechanism
5. Add the parameter to the API schema (`keystone.resource.schema`)
6. Add an API unit test (*keystone/tests/unit/test_v3_resource.py*)
7. Document the new parameter in the api-ref

5.18.2 Write an External Driver

Write an external driver named `file` that implements the Project API. The end result will be that you can set `[resource]/driver = file` in *keystone.conf* to have keystone load a list of project names from a text file, and querying keystone for projects will return projects with those names in the default domain.

Refer to the *Developing Keystone Drivers* tutorial. Your driver can start as an in-tree driver: create a class named `Resource` in *keystone/resource/backends/file.py* that implements `keystone.resource.backends.base.Resource`. Once you have that working, break it out into a separate repository and create a Setuptools entrypoint to allow you to register it with keystone.

5.18.3 Write an Auth Plugin

Write an auth plugin named `hacker` that allows any existing user to authenticate if they provide a valid username and the password `"hax0r"`. The end result will be that you can add `hacker` as an auth method in `[auth]/methods` in *keystone.conf*, and users will be able to get an *unscoped token* using POST /v3/auth/tokens and providing `"hacker"` as the auth method, a valid username as the username, and `"hax0r"` as the password.

Refer to the *Authentication Plugins* documentation. You should create a class `Hacker` in *keystone/auth/plugins/hacker.py* that implements *keystone.auth.plugins.base.AuthMethodHandler*. For bonus points, also add the plugin to keystoneauth so that Python clients can also use this auth method.

CHAPTER

SIX

USER DOCUMENTATION

An end user can find the specific API documentation here, OpenStacks Identity API.

6.1 Supported clients

There are two supported clients, python-keystoneclient project provides python bindings and python-openstackclient provides a command line interface.

6.1.1 Authenticating with a Password via CLI

To authenticate with keystone using a password and `python-openstackclient`, set the following flags, note that the following user referenced below should be granted the `admin` role.

- `--os-username OS_USERNAME`: Name of your user
- `--os-user-domain-name OS_USER_DOMAIN_NAME`: Name of the users domain
- `--os-password OS_PASSWORD`: Password for your user
- `--os-project-name OS_PROJECT_NAME`: Name of your project
- `--os-project-domain-name OS_PROJECT_DOMAIN_NAME`: Name of the projects domain
- `--os-auth-url OS_AUTH_URL`: URL of the keystone authentication server
- `--os-identity-api-version OS_IDENTITY_API_VERSION`: This should always be set to 3

You can also set these variables in your environment so that they do not need to be passed as arguments each time:

```
$ export OS_USERNAME=my_username
$ export OS_USER_DOMAIN_NAME=my_user_domain
$ export OS_PASSWORD=my_password
$ export OS_PROJECT_NAME=my_project
$ export OS_PROJECT_DOMAIN_NAME=my_project_domain
$ export OS_AUTH_URL=http://localhost:5000/v3
$ export OS_IDENTITY_API_VERSION=3
```

For example, the commands `user list`, `token issue` and `project create` can be invoked as follows:

```
# Using password authentication, with environment variables
$ export OS_USERNAME=admin
$ export OS_USER_DOMAIN_NAME=Default
$ export OS_PASSWORD=secret
$ export OS_PROJECT_NAME=admin
$ export OS_PROJECT_DOMAIN_NAME=Default
$ export OS_AUTH_URL=http://localhost:5000/v3
$ export OS_IDENTITY_API_VERSION=3
$ openstack user list
$ openstack project create demo
$ openstack token issue

# Using password authentication, with flags
$ openstack --os-username=admin --os-user-domain-name=Default \
            --os-password=secret \
            --os-project-name=admin --os-project-domain-name=Default \
            --os-auth-url=http://localhost:5000/v3 --os-identity-api-
↪version=3 \
            user list
$ openstack --os-username=admin --os-user-domain-name=Default \
            --os-password=secret \
            --os-project-name=admin --os-project-domain-name=Default \
            --os-auth-url=http://localhost:5000/v3 --os-identity-api-
↪version=3 \
            project create demo
```

6.2 Application Credentials

Users can create application credentials to allow their applications to authenticate to keystone. Users can delegate a subset of their role assignments on a project to an application credential, granting the application the same or restricted authorization to a project. With application credentials, applications authenticate with the application credential ID and a secret string which is not the users password. This way, the users password is not embedded in the applications configuration, which is especially important for users whose identities are managed by an external system such as LDAP or a single-signon system.

See the Identity API reference for more information on authenticating with and managing application credentials.

6.2.1 Managing Application Credentials

Create an application credential using python-openstackclient:

```
$ openstack application credential create monitoring
+--------------+----------------------------------------------------------------
↪--------------------------+
| Field        | Value
↪                          |
+--------------+----------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪-------------------------+
| description  | None                                                          ␣
↪                          |
| expires_at   | None                                                          ␣
↪                          |
| id           | 26bb287fd56a41f8a577c47f79221187                              ␣
↪                          |
| name         | monitoring                                                    ␣
↪                          |
| project_id   | e99b6f4b9bf84a9da27e20c9cbfe887a                              ␣
↪                          |
| roles        | Member anotherrole                                            ␣
↪                          |
| secret       | PJXxBFGPOLwdl3PA6tSivJT9S4RpWhLcNZH2gXzCoxX1C2cnZsj2_Xmfw-
↪LE7Wc-NwuJEYoHcG0gQ5bjWwe-bg |
| unrestricted | False                                                         ␣
↪                          |
+--------------+-----------------------------------------------------------------
↪-------------------------+
```

The only required parameter is a name. The application credential is created for the project to which the user is currently scoped with the same role assignments the user has on that project. Keystone will automatically generate a secret string that will be revealed once at creation time. You can also provide your own secret, if desired:

```
$ openstack application credential create monitoring --secret securesecret
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| description  | None                             |
| expires_at   | None                             |
| id           | bc257241e21747768c83fb9806af392d |
| name         | monitoring                       |
| project_id   | e99b6f4b9bf84a9da27e20c9cbfe887a |
| roles        | Member anotherrole               |
| secret       | securesecret                     |
| unrestricted | False                            |
+--------------+----------------------------------+
```

The secret is hashed before it is stored, so the original secret is not retrievable after creation. If the secret is lost, a new application credential must be created.

If none are provided, the application credential is created with the same role assignments on the project that the user has. You can find out what role assignments you have on a project by examining your token or your keystoneauth session:

```
>>> mysession.auth.auth_ref.role_names
[u'anotherrole', u'Member']
```

If you have more than one role assignment on a project, you can grant your application credential only a subset of your role assignments if desired. This is useful if you have administrator privileges on a project

but only want the application to have basic membership privileges, or if you have basic membership privileges but want the application to only have read-only privileges. You cannot grant the application a role assignment that your user does not already have; for instance, if you are an admin on a project, and you want your application to have read-only access to the project, you must acquire a read-only role assignment on that project yourself before you can delegate it to the application credential. Removing a users role assignment on a project will invalidate the users application credentials for that project.

```
$ openstack application credential create monitoring --role Member
+--------------+----------------------------------------------------------------
↪--------------------------+
| Field        | Value                                                          ␣
↪                          |
+--------------+----------------------------------------------------------------
↪--------------------------+
| description  | None                                                           ␣
↪                          |
| expires_at   | None                                                           ␣
↪                          |
| id           | 5d04e42491a54e83b313aa2625709411                               ␣
↪                          |
| name         | monitoring                                                     ␣
↪                          |
| project_id   | e99b6f4b9bf84a9da27e20c9cbfe887a                               ␣
↪                          |
| roles        | Member                                                         ␣
↪                          |
| secret       | vALEOMENxB_QaKFZOA2XOd7stwrhTlqPKrOdrXXM5BORss9u3O6GT-w_
↪HYCPaZbtg96sDPCdtzVARZLpgUOY_g |
| unrestricted | False                                                          ␣
↪                          |
+--------------+----------------------------------------------------------------
↪--------------------------+
```

An alternative way to limit the application credentials privileges is to use *Access Rules*.

You can provide an expiration date for application credentials:

```
$ openstack application credential create monitoring --expiration '2019-02-
↪12T20:52:43'
+--------------+----------------------------------------------------------------
↪--------------------------+
| Field        | Value                                                          ␣
↪                          |
+--------------+----------------------------------------------------------------
↪--------------------------+
| description  | None                                                           ␣
↪                          |
| expires_at   | 2019-02-12T20:52:43.000000                                     ␣
↪                          |
| id           | 4ea8c4a84f7b4c65a3d84460be9cd1f7                               ␣
↪                          |
```

(continues on next page)

(continued from previous page)

```
| name         | monitoring                                                    ␣
↪                           |
| project_id   | e99b6f4b9bf84a9da27e20c9cbfe887a                              ␣
↪                           |
| roles        | Member anotherrole                                            ␣
↪                           |
| secret       | _My16dlySn6jr7pGvBxjcMrmPA0MCpYlkKWs3gpY3-
↪Ybk05yt2Hh83uMdTLPWlFeh8lOXajIAVHrQaBQ06iz5Q |
| unrestricted | False                                                         ␣
↪                           |
+--------------+---------------------------------------------------------------
↪--------------------------+
```

By default, application credentials are restricted from creating or deleting other application credentials and from creating or deleting trusts. If your application needs to be able to perform these actions and you accept the risks involved, you can disable this protection:

Warning

Restrictions on these Identity operations are deliberately imposed as a safeguard to prevent a compromised application credential from regenerating itself. Disabling this restriction poses an inherent added risk.

```
$ openstack application credential create monitoring --unrestricted
+--------------+---------------------------------------------------------------
↪--------------------------+
| Field        | Value                                                         ␣
↪                           |
+--------------+---------------------------------------------------------------
↪--------------------------+
| description  | None                                                          ␣
↪                           |
| expires_at   | None                                                          ␣
↪                           |
| id           | 0a0372dbedfb4e82ab66449c3316ef1e                              ␣
↪                           |
| name         | monitoring                                                    ␣
↪                           |
| project_id   | e99b6f4b9bf84a9da27e20c9cbfe887a                              ␣
↪                           |
| roles        | Member anotherrole                                            ␣
↪                           |
| secret       |␣
↪ArOy6DYcLeLTRlTmfvF1TH1QmRzYbmD91cbVPOHL3ckyRaLXlaq5pTGJqvCvqg6leEvTI1SQeX3QK-
↪3iwmdPxg |
| unrestricted | True                                                          ␣
↪                           |
+--------------+---------------------------------------------------------------
↪--------------------------+
```

6.2.2 Access Rules

In addition to delegating a subset of roles to an application credential, you may also delegate more fine-grained access control by using access rules.

Note

Application credentials with access rules require additional configuration of each service that will use it. See below for details.

If application credentials with access rules are required, an OpenStack service using keystonemiddleware to authenticate with keystone, needs to define `service_type` in its configuration file. Following is an example for the cinder V3 service:

```
[keystone_authtoken]
service_type = volumev3
```

For other OpenStack sevices, their types can be obtained using the OpenStack client. For example:

```
$ openstack service list -c Name -c Type
+-----------+-----------+
| Name      | Type      |
+-----------+-----------+
| glance    | image     |
| cinderv3  | volumev3  |
| cinderv2  | volumev2  |
| keystone  | identity  |
| nova      | compute   |
| neutron   | network   |
| placement | placement |
+-----------+-----------+
```

Note

Updates to the configuration files of a service require restart of the appropriate services for the changes to take effect.

In order to create an example application credential that is constricted to creating servers in nova, the user can add the following access rules:

```
openstack application credential create scaler-upper --access-rules '[
    {
        "path": "/v2.1/servers",
        "method": "POST",
        "service": "compute"
    }
]'
```

The `"path"` attribute of application credential access rules uses a wildcard syntax to make it more flexible. For example, to create an application credential that is constricted to listing server IP addresses, you

could use either of the following access rules:

```
[
    {
        "path": "/v2.1/servers/*/ips",
        "method": "GET",
        "service": "compute"
    }
]
```

or equivalently:

```
[
    {
        "path": "/v2.1/servers/{server_id}/ips",
        "method": "GET",
        "service": "compute"
    }
]
```

In both cases, a request path containing any server ID will match the access rule. For even more flexibility, the recursive wildcard ** indicates that request paths containing any number of / will be matched. For example:

```
[
    {
        "path": "/v2.1/**",
        "method": "GET",
        "service": "compute"
    }
]
```

will match any nova API for version 2.1.

An access rule created for one application credential can be re-used by providing its ID to another application credential. You can list existing access rules:

```
$ openstack access rule list
+--------+---------+--------+---------------+
| ID     | Service | Method | Path          |
+--------+---------+--------+---------------+
| abcdef | compute | POST   | /v2.1/servers |
+--------+---------+--------+---------------+
```

and create an application credential using that rule:

```
$ openstack application credential create scaler-upper-02 \
 --access-rules '[{"id": "abcdef"}]'
```

6.2.3 Using Application Credentials

Applications can authenticate using the application_credential auth method. For a service using keystonemiddleware to authenticate with keystone, the auth section would look like this:

```
[keystone_authtoken]
auth_url = https://keystone.server/identity/v3
auth_type = v3applicationcredential
application_credential_id = 6cb5fa6a13184e6fab65ba2108adf50c
application_credential_secret= glance_secret
```

You can also identify your application credential with its name and the name or ID of its owner. For example:

```
[keystone_authtoken]
auth_url = https://keystone.server/identity/v3
auth_type = v3applicationcredential
username = glance
user_domain_name = Default
application_credential_name = glance_cred
application_credential_secret = glance_secret
```

6.2.4 Rotating Application Credentials

A user can create multiple application credentials with the same role assignments on the same project. This allows the application credential to be gracefully rotated with minimal or no downtime for your application. In contrast, changing a service users password results in immediate downtime for any application using that password until the application can be updated with the new password.

Note

Rotating application credentials is essential if a team member who has knowledge of the application credential identifier and secret leaves the team for any reason. Rotating application credentials is also recommended as part of regular application maintenance.

Rotating an application credential is a simple process:

1. Create a new application credential. Application credential names must be unique within the users set of application credentials, so this new application credential must not have the same name as the old one.
2. Update your applications configuration to use the new ID (or name and user identifier) and the new secret. For a distributed application, this can be done one node at a time.
3. When your application is fully set up with the new application credential, delete the old one.

6.2.5 Frequently Asked Questions

Why is the application credential owned by the user rather than the project?

Having application credentials be owned by a project rather than by an individual user would be convenient for cases where teams want applications to continue running after the creating user has left the team. However, this would open up a security hole by which the creating user could still gain access to the resources accessible by the application credential even after the user is disabled. Rather than relying on the application credential persisting after users are disabled, it is recommended to proactively rotate the application credential to another user prior to the original creating user being disabled.

6.3 Trusts

OpenStack Identity manages authentication and authorization. A trust is an OpenStack Identity extension that enables delegation and, optionally, impersonation through `keystone`. A trust extension defines a relationship between:

Trustor
The user delegating a limited set of their own rights to another user.

Trustee
The user trust is being delegated to, for a limited time.

The trust can eventually allow the trustee to impersonate the trustor. For security reasons, some safeties are added. For example, if a trustor loses a given role, any trusts the user issued with that role, and the related tokens, are automatically revoked.

The delegation parameters are:

User ID
The user IDs for the trustor and trustee.

Privileges
The delegated privileges are a combination of a project ID and a number of roles that must be a subset of the roles assigned to the trustor.

If you omit all privileges, nothing is delegated. You cannot delegate everything.

Delegation depth
Defines whether or not the delegation is recursive. If it is recursive, defines the delegation chain length.

Specify one of the following values:

- `0`. The delegate cannot delegate these permissions further.
- `1`. The delegate can delegate the permissions to any set of delegates but the latter cannot delegate further.
- `inf`. The delegation is infinitely recursive.

Endpoints
A list of endpoints associated with the delegation.

This parameter further restricts the delegation to the specified endpoints only. If you omit the endpoints, the delegation is useless. A special value of `all_endpoints` allows the trust to be used by all endpoints associated with the delegated project.

Duration
(Optional) Comprised of the start time and end time for the trust.

Note

See the administrator guide on *removing expired trusts* for recommended maintenance procedures.

6.4 API Discovery with JSON Home

6.4.1 What is JSON Home?

JSON Home describes a method of API discovery for non-browser HTTP clients. The draft is still in review, but keystone supplies an implementation accessible to end-users. The result of calling keystones JSON Home API is a JSON document that informs the user about API endpoints, where to find them, and even information about the APIs status (e.g. experimental, supported, deprecated). More information keystones implementation of JSON Home can be found in the specification.

6.4.2 Requesting JSON Home Documents

Requesting keystones JSON Home document is easy. The API does not require a token, but future implementations might expand in its protection with token validation and enforcement. To get a JSON Home document, just query a keystone endpoint with `application/json-home` specified in the `Accept` header:

```
curl -X GET -H "Accept: application/json-home" http://example.com/identity/
```

The result will be a JSON document containing a list of `resources`:

```
{
    "resources": [
        "https://docs.openstack.org/api/openstack-identity/3/ext/OS-TRUST/1.0/
↪rel/trusts": {
            "href": "/v3/OS-TRUST/trusts"
        },
        "https://docs.openstack.org/api/openstack-identity/3/ext/s3tokens/1.0/
↪rel/s3tokens": {
            "href": "/v3/s3tokens"
        },
        "https://docs.openstack.org/api/openstack-identity/3/rel/application_
↪credential": {
            "href-template": "/v3/users/{user_id}/application_credentials/
↪{application_credential_id}",
            "href-vars": {
                "application_credential_id": "https://docs.openstack.org/api/
↪openstack-identity/3/param/application_credential_id",
                "user_id": "https://docs.openstack.org/api/openstack-identity/
↪3/param/user_id"
            }
```

(continues on next page)

(continued from previous page)

```
        },
        "https://docs.openstack.org/api/openstack-identity/3/rel/auth_catalog
↪": {
            "href": "/v3/auth/catalog"
        },
        "https://docs.openstack.org/api/openstack-identity/3/rel/auth_domains
↪": {
            "href": "/v3/auth/domains"
        },
        "https://docs.openstack.org/api/openstack-identity/3/rel/auth_projects
↪": {
            "href": "/v3/auth/projects"
        },
        "https://docs.openstack.org/api/openstack-identity/3/rel/auth_system
↪": {
            "href": "/v3/auth/system"
        },
        ...
    ]
}
```

The list of resources can then be parsed based on the relationship key for a dictionary of data about that endpoint. This includes a path where users can find interact with the endpoint for that specific resources. API status information will also be present.

6.5 API Examples using Curl

6.5.1 v3 API Examples Using Curl

Note

Following are some API examples using curl. Note that these examples are not automatically generated. They can be outdated as things change and are subject to regular updates and changes.

GET /

Discover API version information, links to documentation (PDF, HTML, WADL), and supported media types:

Warning

The v2.0 portion of this response will be removed in the T release. It is only advertised here because the v2.0 API supports the ec2tokens API until the T release. All other functionality of the v2.0 has been removed as of the Queens release. Use v3 for all functionality as it is more complete and secure.

```
$ curl "http://localhost:5000"
```


```
{
    "versions": {
        "values": [
            {
                "id": "v3.10",
                "links": [
                    {
                        "href": "http://127.0.0.1:5000/v3/",
                        "rel": "self"
                    }
                ],
                "media-types": [
                    {
                        "base": "application/json",
                        "type": "application/vnd.openstack.identity-v3+json"
                    }
                ],
                "status": "stable",
                "updated": "2018-02-28T00:00:00Z"
            },
            {
                "id": "v2.0",
                "links": [
                    {
                        "href": "http://127.0.0.1:5000/v2.0/",
                        "rel": "self"
                    },
                    {
                        "href": "https://docs.openstack.org/",
                        "rel": "describedby",
                        "type": "text/html"
                    }
                ],
                "media-types": [
                    {
                        "base": "application/json",
                        "type": "application/vnd.openstack.identity-v2.0+json"
                    }
                ],
                "status": "deprecated",
                "updated": "2016-08-04T00:00:00Z"
            }
        ]
    }
}
```

Tokens

Unscoped

Get an unscoped token:

```
curl -i \
  -H "Content-Type: application/json" \
  -d '
{ "auth": {
    "identity": {
      "methods": ["password"],
      "password": {
        "user": {
          "name": "admin",
          "domain": { "id": "default" },
          "password": "adminpwd"
        }
      }
    }
  }
}' \
  "http://localhost:5000/v3/auth/tokens" ; echo
```

Example response:

```
HTTP/1.1 201 Created
X-Subject-Token: MIIFvgY...
Vary: X-Auth-Token
Content-Type: application/json
Content-Length: 312
Date: Fri, 11 May 2018 03:15:01 GMT

{
  "token": {
      "issued_at": "2018-05-11T03:15:01.000000Z",
      "audit_ids": [
          "0PKh_BDKTWqqaFONE-Sxbg"
      ],
      "methods": [
          "password"
      ],
      "expires_at": "2018-05-11T04:15:01.000000Z",
      "user": {
          "password_expires_at": null,
          "domain": {
              "id": "default",
              "name": "Default"
          },
          "id": "9a7e43333cc44ef4b988f05fc3d3a49d",
          "name": "admin"
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

Project-scoped

Get a project-scoped token:

```
curl -i \
  -H "Content-Type: application/json" \
  -d '
{ "auth": {
    "identity": {
      "methods": ["password"],
      "password": {
        "user": {
          "name": "admin",
          "domain": { "id": "default" },
          "password": "adminpwd"
        }
      }
    },
    "scope": {
      "project": {
        "name": "admin",
        "domain": { "id": "default" }
      }
    }
  }
}' \
  "http://localhost:5000/v3/auth/tokens" ; echo
```

Example response:

```
HTTP/1.1 201 Created
X-Subject-Token: MIIFfQ...
Vary: X-Auth-Token
Content-Type: application/json
Content-Length: 3518
Date: Fri, 11 May 2018 03:38:39 GMT

{
  "token": {
      "is_domain": false,
      "methods": [
          "password"
      ],
      "roles": [
```

(continues on next page)

(continued from previous page)


```
            {
                "id": "b57680c826b44b5ca6122d0f792c3184",
                "name": "Member"
            },
            {
                "id": "3a7bd258345f47479a26aea11a6cc2bb",
                "name": "admin"
            }
        ],
        "expires_at": "2018-05-11T04:38:39.000000Z",
        "project": {
            "domain": {
                "id": "default",
                "name": "Default"
            },
            "id": "3a705b9f56bb439381b43c4fe59dccce",
            "name": "admin"
        },
        "catalog": [
            {
                "endpoints": [
                    {
                        "url": "http://localhost/identity",
                        "interface": "public",
                        "region": "RegionOne",
                        "region_id": "RegionOne",
                        "id": "30a91932e4e94a8ca4dc145bb1bb6b4b"
                    },
                    {
                        "url": "http://localhost/identity",
                        "interface": "admin",
                        "region": "RegionOne",
                        "region_id": "RegionOne",
                        "id": "94d4768735104c9091f0468e7d31c189"
                    }
                ],
                "type": "identity",
                "id": "09af9253500b41ef976a07322b2fa388",
                "name": "keystone"
            },
            {
                "endpoints": [
                    {
                        "url": "http://localhost/volume/v2/
↪3a705b9f56bb439381b43c4fe59dccce",
                        "interface": "public",
                        "region": "RegionOne",
                        "region_id": "RegionOne",
                        "id": "1c4ffe935e7643d99b55938cb12bc38d"
```


(continues on next page)

(continued from previous page)


```
                }
            ],
            "type": "volumev2",
            "id": "413a44234e1a4c3781d4a3c7a7e4c895",
            "name": "cinderv2"
        },
        {
            "endpoints": [
                {
                    "url": "http://localhost/image",
                    "interface": "public",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "33237fdd1a744d0fb40f9127f21ddad4"
                }
            ],
            "type": "image",
            "id": "4d473252145546d2aa589605f1e177c7",
            "name": "glance"
        },
        {
            "endpoints": [
                {
                    "url": "http://localhost/placement",
                    "interface": "public",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "1a421e2f97684d3f86ab4d2cc9c86362"
                }
            ],
            "type": "placement",
            "id": "5dcecbdd4a1d44d0855c560301b27bb5",
            "name": "placement"
        },
        {
            "endpoints": [
                {
                    "url": "http://localhost/compute/v2.1",
                    "interface": "public",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "8e7ea663cc41477c9629cc710bbb1c7d"
                }
            ],
            "type": "compute",
            "id": "87d49efa8fb64006bdb123d223ddcae2",
            "name": "nova"
        },
        {
```


(continues on next page)

(continued from previous page)


```
            "endpoints": [
                {
                    "url": "http://localhost/volume/v1/
↪3a705b9f56bb439381b43c4fe59dccce",
                    "interface": "public",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "97a2c0ac7e304316a1eb58a3757e6ef8"
                }
            ],
            "type": "volume",
            "id": "9408080f1970482aa0e38bc2d4ea34b7",
            "name": "cinder"
        },
        {
            "endpoints": [
                {
                    "url": "http://localhost:8080/v1/AUTH_
↪3a705b9f56bb439381b43c4fe59dccce",
                    "interface": "public",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "d0d823615b0747a9aeca8b83fba105f0"
                },
                {
                    "url": "http://localhost:8080",
                    "interface": "admin",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "e4cb86d9232349f091e0a02390deeb79"
                }
            ],
            "type": "object-store",
            "id": "957ba1fe8b0443f0afe64bfd0858ba5e",
            "name": "swift"
        },
        {
            "endpoints": [
                {
                    "url": "http://localhost:9696/",
                    "interface": "public",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "aa4a0e61cdc54372967ee9e2298f1d53"
                }
            ],
            "type": "network",
            "id": "960fbc66bfcb4fa7900023f647fdc3a5",
            "name": "neutron"
```


(continues on next page)

(continued from previous page)


```
        },
        {
            "endpoints": [
                {
                    "url": "http://localhost/volume/v3/
↪3a705b9f56bb439381b43c4fe59dccce",
                    "interface": "public",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "0c38045a91c34d798e0d2008fee7521d"
                }
            ],
            "type": "volumev3",
            "id": "98adb083914f423d9cb74ad5527e37cb",
            "name": "cinderv3"
        },
        {
            "endpoints": [
                {
                    "url": "http://localhost/compute/v2/
↪3a705b9f56bb439381b43c4fe59dccce",
                    "interface": "public",
                    "region": "RegionOne",
                    "region_id": "RegionOne",
                    "id": "562e12b9ee9549e8b857218ccf2ae321"
                }
            ],
            "type": "compute_legacy",
            "id": "a31e688016614430b28cddddf12d7b88",
            "name": "nova_legacy"
        }
    ],
    "user": {
        "password_expires_at": null,
        "domain": {
            "id": "default",
            "name": "Default"
        },
        "id": "9a7e43333cc44ef4b988f05fc3d3a49d",
        "name": "admin"
    },
    "audit_ids": [
        "TbdrnW4MQDq_GPAVN9-JOQ"
    ],
    "issued_at": "2018-05-11T03:38:39.000000Z"
  }
}
```

Domain-Scoped

Get a domain-scoped token (Note that youre going to need a role-assignment on the domain first!):

```
curl -i \
  -H "Content-Type: application/json" \
  -d '
{ "auth": {
    "identity": {
      "methods": ["password"],
      "password": {
        "user": {
          "name": "admin",
          "domain": { "id": "default" },
          "password": "adminpwd"
        }
      }
    },
    "scope": {
      "domain": {
        "id": "default"
      }
    }
  }
}' \
  "http://localhost:5000/v3/auth/tokens" ; echo
```

Example response:

```
HTTP/1.1 201 Created
X-Subject-Token: MIIFNg...
Vary: X-Auth-Token
Content-Type: application/json
Content-Length: 2590
Date: Fri, 11 May 2018 03:37:09 GMT

{
  "token": {
      "domain": {
          "id": "default",
          "name": "Default"
      },
      "methods": [
          "password"
      ],
      "roles": [
          {
              "id": "b57680c826b44b5ca6122d0f792c3184",
              "name": "Member"
          },
          {
```

(continues on next page)

(continued from previous page)


```
            "id": "3a7bd258345f47479a26aea11a6cc2bb",
            "name": "admin"
        }
    ],
    "expires_at": "2018-05-11T04:37:09.000000Z",
    "catalog": [
        {
            "endpoints": [
                {
                    "region_id": "RegionOne",
                    "url": "http://localhost/identity",
                    "region": "RegionOne",
                    "interface": "public",
                    "id": "30a91932e4e94a8ca4dc145bb1bb6b4b"
                },
                {
                    "region_id": "RegionOne",
                    "url": "http://localhost/identity",
                    "region": "RegionOne",
                    "interface": "admin",
                    "id": "94d4768735104c9091f0468e7d31c189"
                }
            ],
            "type": "identity",
            "id": "09af9253500b41ef976a07322b2fa388",
            "name": "keystone"
        },
        {
            "endpoints": [],
            "type": "volumev2",
            "id": "413a44234e1a4c3781d4a3c7a7e4c895",
            "name": "cinderv2"
        },
        {
            "endpoints": [
                {
                    "region_id": "RegionOne",
                    "url": "http://localhost/image",
                    "region": "RegionOne",
                    "interface": "public",
                    "id": "33237fdd1a744d0fb40f9127f21ddad4"
                }
            ],
            "type": "image",
            "id": "4d473252145546d2aa589605f1e177c7",
            "name": "glance"
        },
        {
            "endpoints": [
```


(continues on next page)

(continued from previous page)


```
                {
                    "region_id": "RegionOne",
                    "url": "http://localhost/placement",
                    "region": "RegionOne",
                    "interface": "public",
                    "id": "1a421e2f97684d3f86ab4d2cc9c86362"
                }
            ],
            "type": "placement",
            "id": "5dcecbdd4a1d44d0855c560301b27bb5",
            "name": "placement"
        },
        {
            "endpoints": [
                {
                    "region_id": "RegionOne",
                    "url": "http://localhost/compute/v2.1",
                    "region": "RegionOne",
                    "interface": "public",
                    "id": "8e7ea663cc41477c9629cc710bbb1c7d"
                }
            ],
            "type": "compute",
            "id": "87d49efa8fb64006bdb123d223ddcae2",
            "name": "nova"
        },
        {
            "endpoints": [],
            "type": "volume",
            "id": "9408080f1970482aa0e38bc2d4ea34b7",
            "name": "cinder"
        },
        {
            "endpoints": [
                {
                    "region_id": "RegionOne",
                    "url": "http://localhost:8080",
                    "region": "RegionOne",
                    "interface": "admin",
                    "id": "e4cb86d9232349f091e0a02390deeb79"
                }
            ],
            "type": "object-store",
            "id": "957ba1fe8b0443f0afe64bfd0858ba5e",
            "name": "swift"
        },
        {
            "endpoints": [
                {
```


(continues on next page)

(continued from previous page)


```
                        "region_id": "RegionOne",
                        "url": "http://localhost:9696/",
                        "region": "RegionOne",
                        "interface": "public",
                        "id": "aa4a0e61cdc54372967ee9e2298f1d53"
                    }
                ],
                "type": "network",
                "id": "960fbc66bfcb4fa7900023f647fdc3a5",
                "name": "neutron"
            },
            {
                "endpoints": [],
                "type": "volumev3",
                "id": "98adb083914f423d9cb74ad5527e37cb",
                "name": "cinderv3"
            },
            {
                "endpoints": [],
                "type": "compute_legacy",
                "id": "a31e688016614430b28cddddf12d7b88",
                "name": "nova_legacy"
            }
        ],
        "user": {
            "password_expires_at": null,
            "domain": {
                "id": "default",
                "name": "Default"
            },
            "id": "9a7e43333cc44ef4b988f05fc3d3a49d",
            "name": "admin"
        },
        "audit_ids": [
            "Sfc8_kywQx-tWNkEVqA1Iw"
        ],
        "issued_at": "2018-05-11T03:37:09.000000Z"
    }
}
```

Getting a token from a token

Get a token from a token:

```
curl -i \
  -H "Content-Type: application/json" \
  -d '
{ "auth": {
    "identity": {
      "methods": ["token"],
      "token": {
        "id": "'$OS_TOKEN'"
      }
    }
  }
}' \
  "http://localhost:5000/v3/auth/tokens" ; echo
```

Example response:

```
HTTP/1.1 201 Created
X-Subject-Token: MIIFxw...
Vary: X-Auth-Token
Content-Type: application/json
Content-Length: 347
Date: Fri, 11 May 2018 03:41:29 GMT

{
  "token": {
      "issued_at": "2018-05-11T03:41:29.000000Z",
      "audit_ids": [
          "zS_C_KROTFeZm-VlG1LjbA",
          "RAjE82q8Rz-Cd50ogCpx3Q"
      ],
      "methods": [
          "token",
          "password"
      ],
      "expires_at": "2018-05-11T04:40:00.000000Z",
      "user": {
          "password_expires_at": null,
          "domain": {
              "id": "default",
              "name": "Default"
          },
          "id": "9a7e43333cc44ef4b988f05fc3d3a49d",
          "name": "admin"
      }
  }
}
```

Note

If a scope was included in the request body then this would get a token with the new scope.

DELETE /v3/auth/tokens

Revoke a token:

```
curl -i -X DELETE \
  -H "X-Auth-Token: $OS_TOKEN" \
  -H "X-Subject-Token: $OS_TOKEN" \
  "http://localhost:5000/v3/auth/tokens"
```

If theres no error then the response is empty.

Domains

GET /v3/domains

List domains:

```
curl -s \
  -H "X-Auth-Token: $OS_TOKEN" \
  "http://localhost:5000/v3/domains" | python -mjson.tool
```

Example response:

```
{
    "domains": [
        {
            "description": "Owns users and tenants (i.e. projects) available␣
↪on Identity API v2.",
            "enabled": true,
            "id": "default",
            "links": {
                "self": "http://identity-server:5000/v3/domains/default"
            },
            "name": "Default"
        }
    ],
    "links": {
        "next": null,
        "previous": null,
        "self": "http://identity-server:5000/v3/domains"
    }
}
```

POST /v3/domains

Create a domain:

```
curl -s \
  -H "X-Auth-Token: $OS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{ "domain": { "name": "newdomain"}}' \
  "http://localhost:5000/v3/domains" | python -mjson.tool
```

Example response:

```
{
    "domain": {
        "enabled": true,
        "id": "3a5140aecd974bf08041328b53a62458",
        "links": {
            "self": "http://identity-server:5000/v3/domains/
↪3a5140aecd974bf08041328b53a62458"
        },
        "name": "newdomain"
    }
}
```

Projects

GET /v3/projects

List projects:

```
curl -s \
 -H "X-Auth-Token: $OS_TOKEN" \
 "http://localhost:5000/v3/projects" | python -mjson.tool
```

Example response:

```
{
    "links": {
        "next": null,
        "previous": null,
        "self": "http://localhost:5000/v3/projects"
    },
    "projects": [
        {
            "description": null,
            "domain_id": "default",
            "enabled": true,
            "id": "3d4c2c82bd5948f0bcab0cf3a7c9b48c",
            "links": {
                "self": "http://localhost:5000/v3/projects/
```

(continues on next page)

(continued from previous page)

```
↪3d4c2c82bd5948f0bcab0cf3a7c9b48c"
            },
            "name": "demo"
        }
    ]
}
```

PATCH /v3/projects/{id}

Disable a project:

```
curl -s -X PATCH \
  -H "X-Auth-Token: $OS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '
{
  "project": {
      "enabled": false
    }
}'\
  "http://localhost:5000/v3/projects/$PROJECT_ID"  | python -mjson.tool
```

Example response:

```
{
    "project": {
        "description": null,
        "domain_id": "default",
        "enabled": false,
        "extra": {},
        "id": "3d4c2c82bd5948f0bcab0cf3a7c9b48c",
        "links": {
            "self": "http://localhost:5000/v3/projects/
↪3d4c2c82bd5948f0bcab0cf3a7c9b48c"
        },
        "name": "demo"
    }
}
```

GET /v3/services

List the services:

```
curl -s \
  -H "X-Auth-Token: $OS_TOKEN" \
  "http://localhost:5000/v3/services" | python -mjson.tool
```

Example response:

```
{
    "links": {
        "next": null,
        "previous": null,
        "self": "http://localhost:5000/v3/services"
    },
    "services": [
        {
            "description": "Keystone Identity Service",
            "enabled": true,
            "id": "bd7397d2c0e14fb69bae8ff76e112a90",
            "links": {
                "self": "http://localhost:5000/v3/services/
↪bd7397d2c0e14fb69bae8ff76e112a90"
            },
            "name": "keystone",
            "type": "identity"
        }
    ]
}
```

GET /v3/endpoints

List the endpoints:

```
curl -s \
 -H "X-Auth-Token: $OS_TOKEN" \
 "http://localhost:5000/v3/endpoints" | python -mjson.tool
```

Example response:

```
{
    "endpoints": [
        {
            "enabled": true,
            "id": "29beb2f1567642eb810b042b6719ea88",
            "interface": "admin",
            "links": {
                "self": "http://localhost:5000/v3/endpoints/
↪29beb2f1567642eb810b042b6719ea88"
```

(continues on next page)

(continued from previous page)

```
            },
            "region": "RegionOne",
            "service_id": "bd7397d2c0e14fb69bae8ff76e112a90",
            "url": "http://localhost:5000/v3"
        }
    ],
    "links": {
        "next": null,
        "previous": null,
        "self": "http://localhost:5000/v3/endpoints"
    }
}
```

Users

GET /v3/users

List users:

```
curl -s \
 -H "X-Auth-Token: $OS_TOKEN" \
 "http://localhost:5000/v3/users" | python -mjson.tool
```

POST /v3/users

Create a user:

```
curl -s \
 -H "X-Auth-Token: $OS_TOKEN" \
 -H "Content-Type: application/json" \
 -d '{"user": {"name": "newuser", "password": "changeme"}}' \
 "http://localhost:5000/v3/users" | python -mjson.tool
```

Example response:

```
{
    "user": {
        "domain_id": "default",
        "enabled": true,
        "id": "ec8fc20605354edd91873f2d66bf4fc4",
        "links": {
            "self": "http://identity-server:5000/v3/users/
↪ec8fc20605354edd91873f2d66bf4fc4"
        },
        "name": "newuser"
    }
}
```

GET /v3/users/{user_id}

Show details for a user:

```
USER_ID=ec8fc20605354edd91873f2d66bf4fc4

curl -s \
 -H "X-Auth-Token: $OS_TOKEN" \
 "http://localhost:5000/v3/users/$USER_ID" | python -mjson.tool
```

Example response:

```
{
    "user": {
        "domain_id": "default",
        "enabled": true,
        "id": "ec8fc20605354edd91873f2d66bf4fc4",
        "links": {
            "self": "http://localhost:5000/v3/users/
↪ec8fc20605354edd91873f2d66bf4fc4"
        },
        "name": "newuser"
    }
}
```

POST /v3/users/{user_id}/password

Change password (using the default policy, this can be done as the user):

```
USER_ID=b7793000f8d84c79af4e215e9da78654
ORIG_PASS=userpwd
NEW_PASS=newuserpwd

curl \
 -H "X-Auth-Token: $OS_TOKEN" \
 -H "Content-Type: application/json" \
 -d '{ "user": {"password": "'$NEW_PASS'", "original_password": "'$ORIG_PASS'
↪"} }' \
 "http://localhost:5000/v3/users/$USER_ID/password"
```

Note

This command doesnt print anything if the request was successful.

PATCH /v3/users/{user_id}

Reset password (using the default policy, this requires admin):

```
USER_ID=b7793000f8d84c79af4e215e9da78654
NEW_PASS=newuserpwd

curl -s -X PATCH \
 -H "X-Auth-Token: $OS_TOKEN" \
 -H "Content-Type: application/json" \
 -d '{ "user": {"password": "'$NEW_PASS'"} }' \
 "http://localhost:5000/v3/users/$USER_ID" | python -mjson.tool
```

Example response:

```
{
    "user": {
        "default_project_id": "3d4c2c82bd5948f0bcab0cf3a7c9b48c",
        "domain_id": "default",
        "email": "demo@example.com",
        "enabled": true,
        "extra": {
            "email": "demo@example.com"
        },
        "id": "269348fdd9374b8885da1418e0730af1",
        "links": {
            "self": "http://localhost:5000/v3/users/
↪269348fdd9374b8885da1418e0730af1"
        },
        "name": "demo"
    }
}
```

PUT /v3/projects/{project_id}/groups/{group_id}/roles/{role_id}

Create group role assignment on project:

```
curl -s -X PUT \
 -H "X-Auth-Token: $OS_TOKEN" \
 "http://localhost:5000/v3/projects/$PROJECT_ID/groups/$GROUP_ID/roles/$ROLE_
↪ID" |
   python -mjson.tool
```

Theres no data in the response if the operation is successful.

POST /v3/OS-TRUST/trusts

Create a trust:

```
curl -s \
 -H "X-Auth-Token: $OS_TOKEN" \
 -H "Content-Type: application/json" \
 -d '
{ "trust": {
    "expires_at": "2014-12-30T23:59:59.999999Z",
    "impersonation": false,
    "project_id": "'$PROJECT_ID'",
    "roles": [
        { "name": "admin" }
      ],
    "trustee_user_id": "'$DEMO_USER_ID'",
    "trustor_user_id": "'$ADMIN_USER_ID'"
}}'\
 "http://localhost:5000/v3/OS-TRUST/trusts" | python -mjson.tool
```

Example response:

```
{
    "trust": {
        "expires_at": "2014-12-30T23:59:59.999999Z",
        "id": "394998fa61f14736b1f0c1f322882949",
        "impersonation": false,
        "links": {
            "self": "http://localhost:5000/v3/OS-TRUST/trusts/
↪394998fa61f14736b1f0c1f322882949"
        },
        "project_id": "3d4c2c82bd5948f0bcab0cf3a7c9b48c",
        "remaining_uses": null,
        "roles": [
            {
                "id": "c703057be878458588961ce9a0ce686b",
                "links": {
                    "self": "http://localhost:5000/v3/roles/
↪c703057be878458588961ce9a0ce686b"
                },
                "name": "admin"
            }
        ],
        "roles_links": {
            "next": null,
            "previous": null,
            "self": "http://localhost:5000/v3/OS-TRUST/trusts/
↪394998fa61f14736b1f0c1f322882949/roles"
        },
        "trustee_user_id": "269348fdd9374b8885da1418e0730af1",
        "trustor_user_id": "3ec3164f750146be97f21559ee4d9c51"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

6.6 Multi-Factor Authentication

6.6.1 Configuring MFA

Configuring MFA right now has to be done entirely by an admin, for how to do that, see *Multi-Factor Authentication*.

6.6.2 Using MFA

Multi-Factor Authentication with Keystone can be used in two ways, either you treat it like current single method authentication and provide all the details upfront, or you doing it as a multi-step process with auth receipts.

Single step

In the single step approach you would supply all the required authentication methods in your request for a token.

Here is an example using 2 factors (`password` and `totp`):

```
{ "auth": {
        "identity": {
            "methods": [
                "password",
                "totp"
            ],
            "totp": {
                "user": {
                    "id": "2ed179c6af12496cafa1d279cb51a78f",
                    "passcode": "012345"
                }
            },
            "password": {
                "user": {
                    "id": "2ed179c6af12496cafa1d279cb51a78f",
                    "password": "super sekret pa55word"
                }
            }
        }
    }
}
```

If all the supplied auth methods are valid, Keystone will return a token.

Multi-Step

In the multi-step approach you can supply any one method from the auth rules:

Again we do a 2 factor example, starting with `password`:

```
{ "auth": {
        "identity": {
            "methods": [
                "password"
            ],
            "password": {
                "user": {
                    "id": "2ed179c6af12496cafa1d279cb51a78f",
                    "password": "super sekret pa55word"
                }
            }
        }
    }
}
```

Provided the method is valid, Keystone will still return a `401`, but will in the response header `Openstack-Auth-Receipt` return a receipt of valid auth method for reuse later.

The response body will also contain information about the auth receipt, and what auth methods may be missing:

```
{
    "receipt":{
        "expires_at":"2018-07-05T08:39:23.000000Z",
        "issued_at":"2018-07-05T08:34:23.000000Z",
        "methods": [
            "password"
        ],
        "user": {
            "domain": {
                "id": "default",
                "name": "Default"
            },
            "id": "ee4dfb6e5540447cb3741905149d9b6e",
            "name": "admin"
        }
    },
    "required_auth_methods": [
        ["totp", "password"]
    ]
}
```

Now you can continue authenticating by supplying the missing auth methods, and supplying the header `Openstack-Auth-Receipt` as gotten from the previous response:

```
{ "auth": {
        "identity": {
            "methods": [
                "totp"
            ],
            "totp": {
                "user": {
                    "id": "2ed179c6af12496cafa1d279cb51a78f",
                    "passcode": "012345"
                }
            }
        }
    }
}
```

Provided the auth methods are valid, Keystone will now supply a token. If not you can try again until the auth receipt expires (e.g in case of TOTP timeout).

CHAPTER

SEVEN

CLI DOCUMENTATION

7.1 keystone-manage

7.1.1 Keystone Management Utility

Author
openstack@lists.openstack.org

Date
2017-02-23

Copyright
OpenStack Foundation

Version
11.0.0

Manual section
1

Manual group
cloud computing

SYNOPSIS

keystone-manage [options]

DESCRIPTION

`keystone-manage` is the command line tool which interacts with the Keystone service to initialize and update data within Keystone. Generally, `keystone-manage` is only used for operations that cannot be accomplished with the HTTP API, such data import/export and database migrations.

USAGE

```
keystone-manage [options] action [additional args]
```

General keystone-manage options:

- `--help` : display verbose help output.

Invoking `keystone-manage` by itself will give you some usage information.

Available commands:

- `bootstrap`: Perform the basic bootstrap process.
- `create_jws_keypair`: Create an ECDSA key pair for JWS token signing.
- `credential_migrate`: Encrypt credentials using a new primary key.
- `credential_rotate`: Rotate Fernet keys for credential encryption.
- `credential_setup`: Setup a Fernet key repository for credential encryption.
- `db_sync`: Sync the database.
- `db_version`: Print the current migration version of the database.
- `doctor`: Diagnose common problems with keystone deployments.
- `domain_config_upload`: Upload domain configuration file.
- `fernet_rotate`: Rotate keys in the Fernet key repository.
- `fernet_setup`: Setup a Fernet key repository for token encryption.
- `mapping_populate`: Prepare domain-specific LDAP backend.
- `mapping_purge`: Purge the identity mapping table.
- `mapping_engine`: Test your federation mapping rules.
- `receipt_rotate`: Rotate auth receipts encryption keys.
- `receipt_setup`: Setup a key repository for auth receipts.
- `saml_idp_metadata`: Generate identity provider metadata.
- `token_rotate`: Rotate token keys in the key repository.
- `token_setup`: Setup a token key repository for token encryption.
- `trust_flush`: Purge expired trusts.

OPTIONS

-h, --help show this help message and exit

--config-dir DIR Path to a config directory to pull *.conf files from. This file set is sorted, so as to provide a predictable parse order if individual options are over-ridden. The set is parsed after the file(s) specified via previous config-file, arguments hence over-ridden options in the directory take precedence.

--config-file PATH Path to a config file to use. Multiple config files can be specified, with values in later files taking precedence. Defaults to None.

--debug, -d If set to true, the logging level will be set to DEBUG instead of the default INFO level.

--log-config-append PATH, --log_config PATH The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, logging_context_format_string).

--log-date-format DATE_FORMAT Defines the format string for %(asctime)s in log records. Default: None . This option is ignored if log_config_append is set.

--log-dir LOG_DIR, --logdir LOG_DIR (Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

--log-file PATH, --logfile PATH (Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

--nodebug The inverse of debug

--nostandard-threads The inverse of standard-threads

--nouse-syslog The inverse of use-syslog

--noverbose The inverse of verbose

--nowatch-log-file The inverse of watch-log-file

--pydev-debug-host PYDEV_DEBUG_HOST Host to connect to for remote debugger.

--pydev-debug-port PYDEV_DEBUG_PORT Port to connect to for remote debugger.

--standard-threads Do not monkey-patch threading system modules.

--syslog-log-facility SYSLOG_LOG_FACILITY Syslog facility to receive log lines. This option is ignored if log_config_append is set.

--use-syslog Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

--verbose, -v If set to false, the logging level will be set to WARNING instead of the default INFO level.

--version show programs version number and exit

--watch-log-file Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

FILES

None

SEE ALSO

- OpenStack Keystone

SOURCE

- Keystone is sourced in Gerrit git Keystone
- Keystone bugs are managed at Launchpad Keystone

7.2 keystone-status

7.2.1 Keystone Status Utility

Author
openstack@lists.openstack.org

Date
2018-10-15

Copyright
OpenStack Foundation

Version
15.0.0

Manual section
1

Manual group
cloud computing

SYNOPSIS

```
keystone-status [options]
```

DESCRIPTION

`keystone-status` is a command line tool that helps operators upgrade their deployment.

USAGE

```
keystone-status [options] action [additional args]
```

Categories are:

- `upgrade`

Detailed descriptions are below.

You can also run with a category argument such as `upgrade` to see a list of all commands in that category:

```
keystone-status upgrade
```

These sections describe the available categories and arguments for **`keystone-status`**.

Categories and commands

`keystone-status upgrade check`
Performs a release-specific readiness check before restarting services with new code, or upgrading. This command expects to have complete configuration and access to the database.

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History of Checks

15.0.0 (Stein)

- Placeholder to be filled in with checks as they are added in Stein.

OPTIONS

```
-h, --help            show this help message and exit
--config-dir DIR      Path to a config directory to pull \*.conf files from.
                      This file set is sorted, so as to provide a
                      predictable parse order if individual options are
                      over-ridden. The set is parsed after the file(s)
                      specified via previous --config-file, arguments hence
                      over-ridden options in the directory take precedence.
--config-file PATH    Path to a config file to use. Multiple config files
                      can be specified, with values in later files taking
                      precedence. Defaults to None.
```

FILES

None

SEE ALSO

- OpenStack Keystone

SOURCE

- Keystone is sourced on opendev.org
- Keystone bugs are managed at Launchpad Keystone

CHAPTER

EIGHT

ADMINISTRATOR GUIDES

OpenStack Identity, code-named keystone, is the default Identity management system for OpenStack. This section contains guides for keystone operators to help with administering a keystone deployment.

8.1 Getting Started

Everything you need to get started administering a keystone deployment.

8.1.1 Identity concepts

Authentication
: The process of confirming the identity of a user. To confirm an incoming request, OpenStack Identity validates a set of credentials users supply. 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. Users provide the token 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 Identity service provides.

Domain
: An Identity service API v3 entity. Domains are a collection of projects and users that define administrative boundaries for managing Identity entities. Domains can represent an individual, company, or operator-owned space. They expose 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
: An Identity service API v3 entity. Groups are a collection of users owned by a domain. A group role, granted to a domain or project, applies to all users in the group. Adding or removing users to or from a group 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. For example, a user can run the `openstack service create` and `openstack endpoint create` commands to register services in their OpenStack installation.

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
An Identity service API v3 entity. 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 Identity service issues a token to a user that includes a list of roles. When a user calls a service, that service interprets the user role set, and determines to which operations or resources each role grants access.

Service
An OpenStack service, such as Compute (nova), Object Storage (swift), or Image service (glance), that provides one or more endpoints through which users can access resources and perform operations.

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. While OpenStack Identity supports token-based authentication in this release, it intends to support additional protocols in the future. OpenStack Identity is an integration service that does not aspire to be a full-fledged identity store and management solution.

User
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.

User management

Identity user management examples:

- Create a user named `alice`:

```
$ openstack user create --password-prompt --email alice@example.com alice
```

- Create a project named `acme`:

```
$ openstack project create acme --domain default
```

- Create a domain named `emea`:

```
$ openstack --os-identity-api-version 3 domain create emea
```

- Create a role named `compute-user`:

```
$ openstack role create compute-user
```

> **Note**
>
> Individual services assign meaning to roles, typically through limiting or granting access to users with the role to the operations that the service supports. Role access is typically configured in the services `policy.yaml` file. For example, to limit Compute access to the `compute-user` role, edit the Compute services `policy.yaml` file to require this role for Compute operations.

The Identity service assigns a project and a role to a user. You might assign the `compute-user` role to the `alice` user in the `acme` project:

```
$ openstack role add --project acme --user alice compute-user
```

A user can have different roles in different projects. For example, Alice might also have the `admin` role in the `Cyberdyne` project. A user can also have multiple roles in the same project.

The `/etc/[SERVICE_CODENAME]/policy.yaml` file controls the tasks that users can perform for a given service. For example, the `/etc/nova/policy.yaml` file specifies the access policy for the Compute service, the `/etc/glance/policy.yaml` file specifies the access policy for the Image service, and the `/etc/keystone/policy.yaml` file specifies the access policy for the Identity service.

The default `policy.yaml` files in the Compute, Identity, and Image services recognize only the `admin` role. Any user with any role in a project can access all operations that do not require the `admin` role.

To restrict users from performing operations in, for example, the Compute service, you must create a role in the Identity service and then modify the `/etc/nova/policy.yaml` file so that this role is required for Compute operations.

For example, the following line in the `/etc/cinder/policy.yaml` file does not restrict which users can create volumes:

```
"volume:create": "",
```

If the user has any role in a project, he can create volumes in that project.

To restrict the creation of volumes to users who have the `compute-user` role in a particular project, you add `"role:compute-user"`:

```
"volume:create": "role:compute-user",
```

To restrict all Compute service requests to require this role, the resulting file looks like:

```
{
    "admin_or_owner": "role:admin or project_id:%(project_id)s",
    "default": "rule:admin_or_owner",
    "compute:create": "role:compute-user",
    "compute:create:attach_network": "role:compute-user",
    "compute:create:attach_volume": "role:compute-user",
    "compute:get_all": "role:compute-user",
    "compute:unlock_override": "rule:admin_api",
    "admin_api": "role:admin",
    "compute_extension:accounts": "rule:admin_api",
    "compute_extension:admin_actions": "rule:admin_api",
    "compute_extension:admin_actions:pause": "rule:admin_or_owner",
```

(continues on next page)

(continued from previous page)

```
    "compute_extension:admin_actions:unpause": "rule:admin_or_owner",
    "compute_extension:admin_actions:suspend": "rule:admin_or_owner",
    "compute_extension:admin_actions:resume": "rule:admin_or_owner",
    "compute_extension:admin_actions:lock": "rule:admin_or_owner",
    "compute_extension:admin_actions:unlock": "rule:admin_or_owner",
    "compute_extension:admin_actions:resetNetwork": "rule:admin_api",
    "compute_extension:admin_actions:injectNetworkInfo": "rule:admin_api",
    "compute_extension:admin_actions:createBackup": "rule:admin_or_owner",
    "compute_extension:admin_actions:migrateLive": "rule:admin_api",
    "compute_extension:admin_actions:migrate": "rule:admin_api",
    "compute_extension:aggregates": "rule:admin_api",
    "compute_extension:certificates": "role:compute-user",
    "compute_extension:cloudpipe": "rule:admin_api",
    "compute_extension:console_output": "role:compute-user",
    "compute_extension:consoles": "role:compute-user",
    "compute_extension:createserverext": "role:compute-user",
    "compute_extension:deferred_delete": "role:compute-user",
    "compute_extension:disk_config": "role:compute-user",
    "compute_extension:evacuate": "rule:admin_api",
    "compute_extension:extended_server_attributes": "rule:admin_api",
    "compute_extension:extended_status": "role:compute-user",
    "compute_extension:flavorextradata": "role:compute-user",
    "compute_extension:flavorextraspecs": "role:compute-user",
    "compute_extension:flavormanage": "rule:admin_api",
    "compute_extension:floating_ip_dns": "role:compute-user",
    "compute_extension:floating_ip_pools": "role:compute-user",
    "compute_extension:floating_ips": "role:compute-user",
    "compute_extension:hosts": "rule:admin_api",
    "compute_extension:keypairs": "role:compute-user",
    "compute_extension:multinic": "role:compute-user",
    "compute_extension:networks": "rule:admin_api",
    "compute_extension:quotas": "role:compute-user",
    "compute_extension:rescue": "role:compute-user",
    "compute_extension:security_groups": "role:compute-user",
    "compute_extension:server_action_list": "rule:admin_api",
    "compute_extension:server_diagnostics": "rule:admin_api",
    "compute_extension:simple_tenant_usage:show": "rule:admin_or_owner",
    "compute_extension:simple_tenant_usage:list": "rule:admin_api",
    "compute_extension:users": "rule:admin_api",
    "compute_extension:virtual_interfaces": "role:compute-user",
    "compute_extension:virtual_storage_arrays": "role:compute-user",
    "compute_extension:volumes": "role:compute-user",
    "compute_extension:volume_attachments:index": "role:compute-user",
    "compute_extension:volume_attachments:show": "role:compute-user",
    "compute_extension:volume_attachments:create": "role:compute-user",
    "compute_extension:volume_attachments:delete": "role:compute-user",
    "compute_extension:volumetypes": "role:compute-user",
    "volume:create": "role:compute-user",
    "volume:get_all": "role:compute-user",
```

(continues on next page)

(continued from previous page)

```
    "volume:get_volume_metadata": "role:compute-user",
    "volume:get_snapshot": "role:compute-user",
    "volume:get_all_snapshots": "role:compute-user",
    "network:get_all_networks": "role:compute-user",
    "network:get_network": "role:compute-user",
    "network:delete_network": "role:compute-user",
    "network:disassociate_network": "role:compute-user",
    "network:get_vifs_by_instance": "role:compute-user",
    "network:allocate_for_instance": "role:compute-user",
    "network:deallocate_for_instance": "role:compute-user",
    "network:validate_networks": "role:compute-user",
    "network:get_instance_uuids_by_ip_filter": "role:compute-user",
    "network:get_floating_ip": "role:compute-user",
    "network:get_floating_ip_pools": "role:compute-user",
    "network:get_floating_ip_by_address": "role:compute-user",
    "network:get_floating_ips_by_project": "role:compute-user",
    "network:get_floating_ips_by_fixed_address": "role:compute-user",
    "network:allocate_floating_ip": "role:compute-user",
    "network:deallocate_floating_ip": "role:compute-user",
    "network:associate_floating_ip": "role:compute-user",
    "network:disassociate_floating_ip": "role:compute-user",
    "network:get_fixed_ip": "role:compute-user",
    "network:add_fixed_ip_to_instance": "role:compute-user",
    "network:remove_fixed_ip_from_instance": "role:compute-user",
    "network:add_network_to_project": "role:compute-user",
    "network:get_instance_nw_info": "role:compute-user",
    "network:get_dns_domains": "role:compute-user",
    "network:add_dns_entry": "role:compute-user",
    "network:modify_dns_entry": "role:compute-user",
    "network:delete_dns_entry": "role:compute-user",
    "network:get_dns_entries_by_address": "role:compute-user",
    "network:get_dns_entries_by_name": "role:compute-user",
    "network:create_private_dns_domain": "role:compute-user",
    "network:create_public_dns_domain": "role:compute-user",
    "network:delete_dns_domain": "role:compute-user"
}
```

Service management

The Identity service provides identity, token, catalog, and policy services. It consists of:

- **keystone Web Server Gateway Interface (WSGI) service**
  Can be run in a WSGI-capable web server such as Apache httpd to provide the Identity service. The service and administrative APIs are run as separate instances of the WSGI service.
- **Identity service functions**
  Each has a pluggable back end that allow different ways to use the particular service. Most support standard back ends like LDAP or SQL.

The Identity service also maintains a user that corresponds to each service, such as, a user named `nova`

for the Compute service, and a special service project called `service`.

For information about how to create services and endpoints, see the *Administrator Guide*.

Groups

A group is a collection of users in a domain. Administrators can create groups and add users to them. A role can then be assigned to the group, rather than individual users. Groups were introduced with the Identity API v3.

Identity API V3 provides the following group-related operations:

- Create a group
- Delete a group
- Update a group (change its name or description)
- Add a user to a group
- Remove a user from a group
- List group members
- List groups for a user
- Assign a role on a project to a group
- Assign a role on a domain to a group
- Query role assignments to groups

Note

The Identity service server might not allow all operations. For example, if you use the Identity server with the LDAP Identity back end and group updates are disabled, a request to create, delete, or update a group fails.

Here are a couple of examples:

- Group A is granted Role A on Project A. If User A is a member of Group A, when User A gets a token scoped to Project A, the token also includes Role A.
- Group B is granted Role B on Domain B. If User B is a member of Group B, when User B gets a token scoped to Domain B, the token also includes Role B.

8.1.2 Configuring Keystone

Identity sources

One of the most impactful decisions youll have to make when configuring keystone is deciding how you want keystone to source your identity data. Keystone supports several different choices that will substantially impact how youll configure, deploy, and interact with keystone.

You can also mix-and-match various sources of identity (see *Domain-specific Configuration* for an example). For example, you can store OpenStack service users and their passwords in SQL, manage customers in LDAP, and authenticate employees via SAML federation. Summary

Feature	*Status*	LDAP	OAuth v1.0a	OpenID Connect	REMOTE_USER	SAML v2	SQL
Local authentication	optional	✓	✓	×	×	×	✓
External authentication	optional	×	×	✓	✓	✓	×
Identity management	optional	✓	✓	×	×	×	✓
PCI-DSS controls	optional	✓	×	×	✓	×	✓
Auditing	optional	✓	×	✓	×	✓	✓

Details

- **Local authentication Status: optional.**

  Notes: Authenticate with keystone by providing credentials directly to keystone.

  Driver Support:

  - **LDAP:** `complete`
  - **OAuth v1.0a:** `complete`
  - **OpenID Connect:** `missing`
  - **REMOTE_USER:** `missing`
  - **SAML v2:** `missing`
  - **SQL:** `complete`

- **External authentication Status: optional.**

  Notes: Authenticate with keystone by providing credentials to an external system that keystone trusts (as with federation).

  Driver Support:

  - **LDAP:** `missing`
  - **OAuth v1.0a:** `missing`
  - **OpenID Connect:** `complete`
  - **REMOTE_USER:** `complete`
  - **SAML v2:** `complete`
  - **SQL:** `missing`

- **Identity management Status: optional.**

  Notes: Create, update, enable/disable, and delete users via Keystones HTTP API.

  Driver Support:

  - **LDAP:** `partial`
  - **OAuth v1.0a:** `complete`

  - **OpenID Connect:** `missing`
  - **REMOTE_USER:** `missing`
  - **SAML v2:** `missing`
  - **SQL:** `complete`

- **PCI-DSS controls Status: optional.**

  Notes: Configure keystone to enforce PCI-DSS compliant security controls.

  Driver Support:

  - **LDAP:** `partial`
  - **OAuth v1.0a:** `missing`
  - **OpenID Connect:** `missing`
  - **REMOTE_USER:** `partial`
  - **SAML v2:** `missing`
  - **SQL:** `complete`

- **Auditing Status: optional.**

  Notes: Audit authentication flows using PyCADF.

  Driver Support:

  - **LDAP:** `complete`
  - **OAuth v1.0a:** `missing`
  - **OpenID Connect:** `complete`
  - **REMOTE_USER:** `missing`
  - **SAML v2:** `complete`
  - **SQL:** `complete`

Notes:

- **This document is a continuous work in progress**

8.1.3 Bootstrapping Identity

After keystone is deployed and configured, it must be pre-populated with some initial data before it can be used. This process is known as bootstrapping and it typically involves creating the systems first user, project, domain, service, and endpoint, among other things. The goal of bootstrapping is to put enough information into the system such that it can function solely through the API using normal authentication flows. After the first user is created, which must be an administrator, you can use that account to interact with keystone via the API.

Keystone provides two separate ways to bootstrap a deployment. The first is with the `keystone-manage bootstrap` command. This is the preferred and recommended way to bootstrap new installations. The second, and original way of bootstrapping involves configuring a secret and deploying special middleware in front of the identity service. The secret is known as the `ADMIN_TOKEN`. Any requests made to the identity API with the `ADMIN_TOKEN` will completely bypass authentication allowing access to the entire API.

Using the CLI

The process requires access to an environment with keystone binaries installed, typically on the service host.

The `keystone-manage bootstrap` command will create a user, project and role, and will assign the newly created role to the newly created user on the newly created project. By default, the names of these new resources will be called `admin`.

The defaults may be overridden by calling `--bootstrap-username`, `--bootstrap-project-name` and `--bootstrap-role-name`. Each of these have an environment variable equivalent: `OS_BOOTSTRAP_USERNAME`, `OS_BOOTSTRAP_PROJECT_NAME` and `OS_BOOTSTRAP_ROLE_NAME`.

A user password must also be supplied. This can be passed in as either `--bootstrap-password`, or set as an environment variable using `OS_BOOTSTRAP_PASSWORD`.

Optionally, if specified by `--bootstrap-public-url`, `--bootstrap-admin-url` and/or `--bootstrap-internal-url` or the equivalent environment variables, the command will create an identity service with the specified endpoint information. You may also configure the `--bootstrap-region-id` and `--bootstrap-service-name` for the endpoints to your deployments requirements.

Note

We strongly recommend that you configure the identity service and its endpoints while bootstrapping keystone.

Minimally, keystone can be bootstrapped with:

```
$ keystone-manage bootstrap --bootstrap-password s3cr3t
```

Verbosely, keystone can be bootstrapped with:

```
$ keystone-manage bootstrap \
    --bootstrap-password s3cr3t \
    --bootstrap-username admin \
    --bootstrap-project-name admin \
    --bootstrap-role-name admin \
    --bootstrap-service-name keystone \
    --bootstrap-region-id RegionOne \
    --bootstrap-admin-url http://localhost:5000 \
    --bootstrap-public-url http://localhost:5000 \
    --bootstrap-internal-url http://localhost:5000
```

This will create an `admin` user with the `admin` role on the `admin` project and the system. This allows the user to generate project-scoped and system-scoped tokens which ensures they have full RBAC authorization. The user will have the password specified in the command. Note that both the user and the project will be created in the `default` domain. By not creating an endpoint in the catalog users will need to provide endpoint overrides to perform additional identity operations.

This command will also create `member` and `reader` roles. The `admin` role implies the `member` role and `member` role implies the `reader` role. By default, these three roles are immutable, meaning they are created with the `immutable` resource option and cannot be modified or deleted unless the option is removed. To disable this behavior, add the `--no-immutable-roles` flag.

By creating an `admin` user and an identity endpoint you may authenticate to keystone and perform identity operations like creating additional services and endpoints using the `admin` user. This will preclude the need to ever use or configure the `admin_token` (described below). It is also, by design, more secure.

To test a proper configuration, a user can use OpenStackClient CLI:

```
$ openstack project list --os-username admin --os-project-name admin \
    --os-user-domain-id default --os-project-domain-id default \
    --os-identity-api-version 3 --os-auth-url http://localhost:5000 \
    --os-password s3cr3t
```

Using a shared secret

Note

We strongly recommended that you configure the identity service with the `keystone-manage bootstrap` command and not the `ADMIN_TOKEN`. The `ADMIN_TOKEN` can leave your deployment vulnerable by exposing administrator functionality through the API based solely on a single secret. You shouldnt have to use `ADMIN_TOKEN` at all, unless you have some special case bootstrapping requirements.

Before you can use the identity API, you need to configure keystone with a shared secret. Requests made with this secret will bypass authentication and grant administrative access to the identity API. The following configuration snippet shows the shared secret as being `ADMIN`:

```
[DEFAULT]
admin_token = ADMIN
```

You can use the shared secret, or `admin_token`, to make API request to keystone that bootstrap the rest of the deployment. You must create a project, user, and role in order to use normal user authentication through the API.

The `admin_token` does not represent a user or explicit authorization of any kind. After bootstrapping, failure to remove this functionality exposes an additional attack vector and security risk.

8.1.4 Manage projects, users, and roles

As an administrator, you manage projects, users, and roles. Projects are organizational units in the cloud to which you can assign users. Projects are also known as *tenants* or *accounts*. Users can be members of one or more projects. Roles define which actions users can perform. You assign roles to user-project pairs.

You can define actions for OpenStack service roles in the `/etc/PROJECT/policy.yaml` files. For example, define actions for Compute service roles in the `/etc/nova/policy.yaml` file.

You can manage projects, users, and roles independently from each other.

During cloud set up, the operator defines at least one project, user, and role.

You can add, update, and delete projects and users, assign users to one or more projects, and change or remove the assignment. To enable or temporarily disable a project or user, update that project or user. You can also change quotas at the project level.

Before you can delete a user account, you must remove the user account from its primary project.

Before you can run client commands, you need to have a cloud config file or you can download and source an OpenStack RC file. See the Configuration documentation from the python-openstackclient project for more details.

Projects

A project is a group of zero or more users. In Compute, a project owns virtual machines. In Object Storage, a project owns containers. Users can be associated with more than one project. Each project and user pairing can have a role associated with it.

List projects

List all projects with their ID, name, and whether they are enabled or disabled:

```
$ openstack project list
+----------------------------------+--------------------+
| ID                               | Name               |
+----------------------------------+--------------------+
| f7ac731cc11f40efbc03a9f9e1d1d21f | admin              |
| c150ab41f0d9443f8874e32e725a4cc8 | alt_demo           |
| a9debfe41a6d4d09a677da737b907d5e | demo               |
| 9208739195a34c628c58c95d157917d7 | invisible_to_admin |
| 3943a53dc92a49b2827fae94363851e1 | service            |
| 80cab5e1f02045abad92a2864cfd76cb | test_project       |
+----------------------------------+--------------------+
```

Create a project

Create a project named `new-project`:

```
$ openstack project create --description 'my new project' new-project \
  --domain default
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | my new project                   |
| domain_id   | e601210181f54843b51b3edff41d4980 |
| enabled     | True                             |
| id          | 1a4a0618b306462c9830f876b0bd6af2 |
| is_domain   | False                            |
| name        | new-project                      |
| parent_id   | e601210181f54843b51b3edff41d4980 |
| tags        | []                               |
+-------------+----------------------------------+
```

- Creating a project without using a domain scoped token, i.e. using a project scoped token or a

system scoped token, and also without specifying a domain or domain_id, the project will automatically be created on the default domain.

Update a project

Specify the project ID to update a project. You can update the name, description, and enabled status of a project.

- To temporarily disable a project:

```
$ openstack project set PROJECT_ID --disable
```

- To enable a disabled project:

```
$ openstack project set PROJECT_ID --enable
```

- To update the name of a project:

```
$ openstack project set PROJECT_ID --name project-new
```

- To verify your changes, show information for the updated project:

```
$ openstack project show PROJECT_ID
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | my new project                   |
| domain_id   | e601210181f54843b51b3edff41d4980 |
| enabled     | True                             |
| id          | 0b0b995694234521bf93c792ed44247f |
| is_domain   | False                            |
| name        | new-project                      |
| parent_id   | e601210181f54843b51b3edff41d4980 |
| tags        | []                               |
+-------------+----------------------------------+
```

Delete a project

Specify the project ID to delete a project:

```
$ openstack project delete PROJECT_ID
```

Users

List users

List all users:

```
$ openstack user list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 352b37f5c89144d4ad0534139266d51f | admin    |
| 86c0de739bcb4802b8dc786921355813 | demo     |
| 32ec34aae8ea432e8af560a1cec0e881 | glance   |
| 7047fcb7908e420cb36e13bbd72c972c | nova     |
+----------------------------------+----------+
```

Create a user

To create a user, you must specify a name. Optionally, you can specify a project ID, password, and email address. It is recommended that you include the project ID and password because the user cannot log in to the dashboard without this information.

Create the `new-user` user:

```
$ openstack user create --project new-project --password PASSWORD new-user
+------------+----------------------------------+
| Field      | Value                            |
+------------+----------------------------------+
| email      | None                             |
| enabled    | True                             |
| id         | 6322872d9c7e445dbbb49c1f9ca28adc |
| name       | new-user                         |
| project_id | 0b0b995694234521bf93c792ed44247f |
| username   | new-user                         |
+------------+----------------------------------+
```

Update a user

You can update the name, email address, and enabled status for a user.

- To temporarily disable a user account:

  ```
  $ openstack user set USER_NAME --disable
  ```

  If you disable a user account, the user cannot log in to the dashboard. However, data for the user account is maintained, so you can enable the user at any time.

- To enable a disabled user account:

```
$ openstack user set USER_NAME --enable
```

- To change the name and description for a user account:

```
$ openstack user set USER_NAME --name user-new --email new-user@example.
↪com
User has been updated.
```

Delete a user

Delete a specified user account:

```
$ openstack user delete USER_NAME
```

Roles and role assignments

List available roles

List the available roles:

```
$ openstack role list
+----------------------------------+---------------+
| ID                               | Name          |
+----------------------------------+---------------+
| 71ccc37d41c8491c975ae72676db687f | member        |
| 149f50a1fe684bfa88dae76a48d26ef7 | ResellerAdmin |
| 9fe2ff9ee4384b1894a90878d3e92bab | reader        |
| 6ecf391421604da985db2f141e46a7c8 | admin         |
| deb4fffd123c4d02a907c2c74559dccf | anotherrole   |
+----------------------------------+---------------+
```

Create a role

Users can be members of multiple projects. To assign users to multiple projects, define a role and assign that role to a user-project pair.

Create the `new-role` role:

```
$ openstack role create new-role
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | None                             |
| domain_id   | None                             |
| id          | a34425c884c74c8881496dc2c2e84ffc |
| name        | new-role                         |
+-------------+----------------------------------+
```

Note

If you are using identity v3, you may need to use the `--domain` option with a specific domain name.

Assign a role

To assign a user to a project, you must assign the role to a user-project pair.

1. Assign a role to a user-project pair:

```
$ openstack role add --user USER_NAME --project PROJECT_NAME ROLE_NAME
```

   For example, assign the `new-role` role to the `demo` user and `test-project` project pair:

```
$ openstack role add --user demo --project test-project new-role
```

2. Verify the role assignment:

```
$ openstack role assignment list --user USER_NAME \
  --project PROJECT_NAME --names
+-------------+--------------+-------+--------------+--------+--------+---
↪--------+
| Role        | User         | Group | Project      | Domain | System |␣
↪Inherited |
+-------------+--------------+-------+--------------+--------+--------+---
↪--------+
| new-role    | demo@Default |       | demo@Default |        |        |␣
↪False     |
| member      | demo@Default |       | demo@Default |        |        |␣
↪False     |
| anotherrole | demo@Default |       | demo@Default |        |        |␣
↪False     |
+-------------+--------------+-------+--------------+--------+--------+---
↪--------+
```

Note

Before the Newton release, users would run the **`openstack role list --user USER_NAME --project TENANT_ID`** command to verify the role assignment.

View role details

View details for a specified role:

```
$ openstack role show ROLE_NAME
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | None                             |
| domain_id   | None                             |
| id          | a34425c884c74c8881496dc2c2e84ffc |
| name        | new-role                         |
+-------------+----------------------------------+
```

Remove a role

Remove a role from a user-project pair:

1. Run the **openstack role remove** command:

   ```
   $ openstack role remove --user USER_NAME --project PROJECT_NAME ROLE_NAME
   ```

2. Verify the role removal:

   ```
   $ openstack role assignment list --user USER_NAME --project PROJECT_NAME -
   ↪-names
   ```

   If the role was removed, the command output omits the removed role.

Creating implied roles

It is possible to build role hierarchies by having roles imply other roles. These are called implied roles, or role inference rules.

To illustrate the capability, lets have the `admin` role imply the `member` role. In this example, if a user was assigned the prior role, which in this case is the `admin` role, they would also get the `member` role that it implies.

```
$ openstack implied role create admin --implied-role member
+------------+----------------------------------+
| Field      | Value                            |
+------------+----------------------------------+
| implies    | 71ccc37d41c8491c975ae72676db687f |
| prior_role | 29c09e68e6f741afa952a837e29c700b |
+------------+----------------------------------+
```

Note

Role implications only go one way, from a prior role to an implied role. Therefore assigning a user the `member` will not grant them the `admin` role.

This makes it easy to break up large roles into smaller pieces, allowing for fine grained permissions, while still having an easy way to assign all the pieces as if they were a single one. For example, you can have a `member` role imply `compute_member`, `network_member`, and `volume_member`, and then assign either the full-blown `member` role to users or any one of the subsets.

Listing implied roles

To list implied roles:

```
$ openstack implied role list
+----------------------------------+----------------+-------------------------
↪----------+--------------------+
| Prior Role ID                    | Prior Role Name | Implied Role ID       ␣
↪          | Implied Role Name |
+----------------------------------+----------------+-------------------------
↪----------+--------------------+
| 29c09e68e6f741afa952a837e29c700b | admin          |␣
↪71ccc37d41c8491c975ae72676db687f | member           |
+----------------------------------+----------------+-------------------------
↪----------+--------------------+
```

Deleting implied roles

To delete a role inference rule:

```
$ openstack implied role delete admin --implied-role member
```

Note

Deleting an implied role removes the role inference rule. It does not delete the prior or implied role. Therefore if a user was assigned the prior role, they will no longer have the roles that it implied.

8.1.5 Create and manage services and service users

Service Catalog

OpenStack services can be discovered when registered in keystones service catalog. The service catalog can be managed as either a static file template or as a dynamic database table.

File-based Service Catalog (templated.Catalog)

The templated catalog is an in-memory backend initialized from a read-only `template_file`. Choose this option only if you know that your service catalog will not change very much over time.

> **Note**
>
> Attempting to change your service catalog against this driver will result in `HTTP 501 Not Implemented` errors. This is the expected behavior. If you want to use these commands, you must instead use the SQL-based Service Catalog driver.

`keystone.conf` example:

```
[catalog]
driver = templated
template_file = /opt/stack/keystone/etc/default_catalog.templates
```

The value of `template_file` is expected to be an absolute path to your service catalog configuration. An example `template_file` is included in keystone, however you should create your own to reflect your deployment.

SQL-based Service Catalog (sql.Catalog)

A dynamic database-backed driver fully supporting persistent configuration.

`keystone.conf` example:

```
[catalog]
driver = sql
```

> **Note**
>
> A *template_file* does not need to be defined for the sql based catalog.

To build your service catalog using this driver, see the built-in help:

```
$ openstack --help
$ openstack service create --help
$ openstack endpoint create --help
```

Create a service

1. List the available services:

```
$ openstack service list
+----------------------------------+----------+------------+
| ID                               | Name     | Type       |
+----------------------------------+----------+------------+
| 9816f1faaa7c4842b90fb4821cd09223 | cinder   | volume     |
| 1250f64f31e34dcd9a93d35a075ddbe1 | cinderv2 | volumev2   |
| da8cf9f8546b4a428c43d5e032fe4afc | ec2      | ec2        |
| 5f105eeb55924b7290c8675ad7e294ae | glance   | image      |
| dcaa566e912e4c0e900dc86804e3dde0 | keystone | identity   |
| 4a715cfbc3664e9ebf388534ff2be76a | nova     | compute    |
| 1aed4a6cf7274297ba4026cf5d5e96c5 | novav21  | computev21 |
| bed063c790634c979778551f66c8ede9 | neutron  | network    |
| 6feb2e0b98874d88bee221974770e372 | s3       | s3         |
+----------------------------------+----------+------------+
```

2. To create a service, run this command:

```
$ openstack service create --name SERVICE_NAME --description SERVICE_
↪DESCRIPTION SERVICE_TYPE
```

The arguments are:

- `service_name`: the unique name of the new service.
- `service_type`: the service type, such as `identity`, `compute`, `network`, `image`, `object-store` or any other service identifier string.
- `service_description`: the description of the service.

For example, to create a `swift` service of type `object-store`, run this command:

```
$ openstack service create --name swift --description "object store␣
↪service" object-store
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | object store service             |
| enabled     | True                             |
| id          | 84c23f4b942c44c38b9c42c5e517cd9a |
| name        | swift                            |
| type        | object-store                     |
+-------------+----------------------------------+
```

3. To get details for a service, run this command:

```
$ openstack service show SERVICE_TYPE|SERVICE_NAME|SERVICE_ID
```

For example:

```
$ openstack service show object-store
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | object store service             |
| enabled     | True                             |
| id          | 84c23f4b942c44c38b9c42c5e517cd9a |
| name        | swift                            |
| type        | object-store                     |
+-------------+----------------------------------+
```

Create an endpoint

1. Once a service is created, register it at an endpoint:

```
$ openstack endpoint create nova public http://example.com/compute/v2.1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | c219aa779e90403eb4a78cf0aa7d38b1 |
| interface    | public                           |
| region       | None                             |
| region_id    | None                             |
| service_id   | 0f5da035b8e94629bf35e7ec1703a8eb |
| service_name | nova                             |
| service_type | compute                          |
| url          | http://example.com/compute/v2.1  |
+--------------+----------------------------------+
```

Delete a service

To delete a specified service, specify its ID.

```
$ openstack service delete SERVICE_TYPE|SERVICE_NAME|SERVICE_ID
```

For example:

```
$ openstack service delete object-store
```

Service users

To authenticate users against the Identity service, you must create a service user for each OpenStack service. For example, create a service user for the Compute, Block Storage, and Networking services.

To configure the OpenStack services with service users, create a project for all services and create users for each service. Assign the admin role to each service user and project pair. This role enables users to validate tokens and authenticate and authorize other user requests.

Create service users

1. Create a project for the service users. Typically, this project is named `service`, but choose any name you like:

```
$ openstack project create service --domain default
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | None                             |
| domain_id   | e601210181f54843b51b3edff41d4980 |
| enabled     | True                             |
| id          | 3e9f3f5399624b2db548d7f871bd5322 |
| is_domain   | False                            |
| name        | service                          |
| parent_id   | e601210181f54843b51b3edff41d4980 |
+-------------+----------------------------------+
```

2. Create service users for the relevant services for your deployment. For example:

```
$ openstack user create nova --password Sekr3tPass
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 95ec3e1d5dd747f5a512d261731d29c7 |
| name                | nova                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

3. Assign the admin role to the user-project pair.

```
$ openstack role add --project service --user nova admin
+-------+----------------------------------+
| Field | Value                            |
+-------+----------------------------------+
| id    | 233109e756c1465292f31e7662b429b1 |
| name  | admin                            |
+-------+----------------------------------+
```

Configuring service tokens

A lot of operations in OpenStack require communication between multiple services on behalf of the user. For example, the Image service storing the users images in the Object Storage service. If the image is significantly large, the operation might fail due to the users token having expired during upload.

In the above scenarios, the Image service will attach both the users token and its own token (called the service token), as per the diagram below.

```
+----------------+
|      User      |
+-------+--------+
        | Access Image Data Request
        | X-AUTH-TOKEN: <end user token>
        |
+-------v---------+
|     Glance      |
+-------+---------+
        | Access Image Data Request
        | X-AUTH-TOKEN: <original end user token>
        | X-SERVICE-TOKEN: <glance service user token>
        |
+-------v---------+
|      Swift      |
+-----------------+
```

When a service receives a call from another service, it validates that the token has the appropriate roles for a service user. This is configured in each individual service configuration, under the section `[keystone_authtoken]`.

If the service token is valid, the operation will be allowed even if the users token has expired.

The `service_token_roles` option is the list of roles that the service token must contain to be a valid service token. In the previous steps, we have assigned the *admin* role to service users, so set the option to that and set `service_token_roles_required` to `true`.

```
[keystone_authtoken]
service_token_roles = admin
service_token_roles_required = true
```

For more information regarding service tokens, please see the `keystonemiddleware` release notes.

8.2 Keystone Configuration

Information and recommendations for general configuration of keystone for keystone administrators. See the main *Configuration* section for complete keystone configuration documentation and sample config files.

8.2.1 Troubleshoot the Identity service

To troubleshoot the Identity service, review the logs in the `/var/log/keystone/keystone.log` file.

Use the `/etc/keystone/logging.conf` file to configure the location of log files.

Note

The `insecure_debug` flag is unique to the Identity service. If you enable `insecure_debug`, error messages from the API change to return security-sensitive information. For example, the error message on failed authentication includes information on why your authentication failed.

The logs show the components that have come in to the WSGI request, and ideally show an error that explains why an authorization request failed. If you do not see the request in the logs, run keystone with the `--debug` parameter. Pass the `--debug` parameter before the command parameters.

8.2.2 Logging

You configure logging externally to the rest of Identity. The name of the file specifying the logging configuration is set using the `log_config_append` option in the `[DEFAULT]` section of the `/etc/keystone/keystone.conf` file. To route logging through syslog, set `use_syslog=true` in the `[DEFAULT]` section.

A sample logging configuration file is available with the project in `etc/logging.conf.sample`. Like other OpenStack projects, Identity uses the Python logging module, which provides extensive configuration options that let you define the output levels and formats.

8.2.3 Domain-specific configuration

The Identity service supports domain-specific Identity drivers. The drivers allow a domain to have its own LDAP or SQL back end. By default, domain-specific drivers are disabled.

Domain-specific Identity configuration options can be stored in domain-specific configuration files, or in the Identity SQL database using API REST calls.

Note

Storing and managing configuration options in an SQL database is experimental in Kilo, and added to the Identity service in the Liberty release.

Enable drivers for domain-specific configuration files

To enable domain-specific drivers, set these options in the `/etc/keystone/keystone.conf` file:

```
[identity]
domain_specific_drivers_enabled = True
domain_config_dir = /etc/keystone/domains
```

When you enable domain-specific drivers, Identity looks in the `domain_config_dir` directory for configuration files that are named as `keystone.DOMAIN_NAME.conf`. A domain without a domain-specific configuration file uses options in the primary configuration file.

Enable drivers for storing configuration options in SQL database

To enable domain-specific drivers, set these options in the `/etc/keystone/keystone.conf` file:

```
[identity]
domain_specific_drivers_enabled = True
domain_configurations_from_database = True
```

Any domain-specific configuration options specified through the Identity v3 API will override domain-specific configuration files in the `/etc/keystone/domains` directory.

Unlike the file-based method of specifying domain-specific configurations, options specified via the Identity API will become active without needing to restart the keystone server. For performance reasons, the current state of configuration options for a domain are cached in the keystone server, and in multi-process and multi-threaded keystone configurations, the new configuration options may not become active until the cache has timed out. The cache settings for domain config options can be adjusted in the general keystone configuration file (option `cache_time` in the `domain_config` group).

Note

It is important to notice that when using either of these methods of specifying domain-specific configuration options, the main keystone configuration file is still maintained. Only those options that relate to the Identity driver for users and groups (i.e. specifying whether the driver for this domain is SQL or LDAP, and, if LDAP, the options that define that connection) are supported in a domain-specific manner. Further, when using the configuration options via the Identity API, the driver option must be set to an LDAP driver (attempting to set it to an SQL driver will generate an error when it is subsequently used).

For existing installations that already use file-based domain-specific configurations who wish to migrate to the SQL-based approach, the `keystone-manage` command can be used to upload all configuration files to the SQL database:

```
$ keystone-manage domain_config_upload --all
```

Once uploaded, these domain-configuration options will be visible via the Identity API as well as applied to the domain-specific drivers. It is also possible to upload individual domain-specific configuration files by specifying the domain name:

```
$ keystone-manage domain_config_upload --domain-name DOMAINA
```

Note

It is important to notice that by enabling either of the domain-specific configuration methods, the operations of listing all users and listing all groups are not supported, those calls will need either a domain filter to be specified or usage of a domain scoped token.

Note

Keystone does not support moving the contents of a domain (i.e. its users and groups) from one backend to another, nor group membership across backend boundaries.

Note

When using the file-based domain-specific configuration method, to delete a domain that uses a domain specific backend, its necessary to first disable it, remove its specific configuration file (i.e. its corresponding keystone.<domain_name>.conf) and then restart the Identity server. When managing configuration options via the Identity API, the domain can simply be disabled and deleted via the Identity API; since any domain-specific configuration options will automatically be removed.

Note

Although keystone supports multiple LDAP backends via the above domain-specific configuration methods, it currently only supports one SQL backend. This could be either the default driver or a single domain-specific backend, perhaps for storing service users in a predominantly LDAP installation.

Note

Keystone has deprecated the `keystone-manage domain_config_upload` option. The keystone team recommends setting domain config options via the API instead.

Due to the need for user and group IDs to be unique across an OpenStack installation and for keystone to be able to deduce which domain and backend to use from just a user or group ID, it dynamically builds a persistent identity mapping table from a public ID to the actual domain, local ID (within that backend) and entity type. The public ID is automatically generated by keystone when it first encounters the entity. If the local ID of the entity is from a backend that does not guarantee to generate UUIDs, a hash algorithm will generate a public ID for that entity, which is what will be exposed by keystone.

The use of a hash will ensure that if the public ID needs to be regenerated then the same public ID will be created. This is useful if you are running multiple keystones and want to ensure the same ID would be generated whichever server you hit.

Note

In case of the LDAP backend, the names of users and groups are not hashed. As a result, these are length limited to 255 characters. Longer names will result in an error.

While keystone will dynamically maintain the identity mapping, including removing entries when entities are deleted via the keystone, for those entities in backends that are managed outside of keystone (e.g. a read-only LDAP), keystone will not know if entities have been deleted and hence will continue to carry stale identity mappings in its table. While benign, keystone provides an ability for operators to purge the mapping table of such stale entries using the keystone-manage command, for example:

```
$ keystone-manage mapping_purge --domain-name DOMAINA --local-id abc@de.com
```

A typical usage would be for an operator to obtain a list of those entries in an external backend that had been deleted out-of-band to keystone, and then call keystone-manage to purge those entries by specifying the domain and local-id. The type of the entity (i.e. user or group) may also be specified if this is needed to uniquely identify the mapping.

Since public IDs can be regenerated **with the correct generator implementation**, if the details of those entries that have been deleted are not available, then it is safe to simply bulk purge identity mappings periodically, for example:

```
$ keystone-manage mapping_purge --domain-name DOMAINA
```

will purge all the mappings for DOMAINA. The entire mapping table can be purged with the following command:

```
$ keystone-manage mapping_purge --all
```

Generating public IDs in the first run may take a while, and most probably first API requests to fetch user list will fail by timeout. To prevent this, `mapping_populate` command should be executed. It should be executed right after LDAP has been configured or after `mapping_purge`.

```
$ keystone-manage mapping_populate --domain DOMAINA
```

Public ID Generators

Keystone supports a customizable public ID generator and it is specified in the `[identity_mapping]` section of the configuration file. Keystone provides a sha256 generator as default, which produces regenerable public IDs. The generator algorithm for public IDs is a balance between key size (i.e. the length of the public ID), the probability of collision and, in some circumstances, the security of the public ID. The maximum length of public ID supported by keystone is 64 characters, and the default generator (sha256) uses this full capability. Since the public ID is what is exposed externally by keystone and potentially stored in external systems, some installations may wish to make use of other generator algorithms that have a different trade-off of attributes. A different generator can be installed by configuring the following property:

- `generator` - identity mapping generator. Defaults to `sha256` (implemented by *keystone.identity.id_generators.sha256.Generator*)

Warning

Changing the generator may cause all existing public IDs to be become invalid, so typically the generator selection should be considered immutable for a given installation.

Migrate domain-specific configuration files to the SQL database

You can use the `keystone-manage` command to migrate configuration options in domain-specific configuration files to the SQL database:

```
# keystone-manage domain_config_upload --all
```

To upload options from a specific domain-configuration file, specify the domain name:

```
# keystone-manage domain_config_upload --domain-name DOMAIN_NAME
```

8.2.4 Integrate Identity with LDAP

The OpenStack Identity service supports integration with existing LDAP directories for authentication and authorization services. LDAP back ends require initialization before configuring the OpenStack Identity service to work with it. For more information, see Setting up LDAP for use with Keystone.

When the OpenStack Identity service is configured to use LDAP back ends, you can split authentication (using the *identity* feature) and authorization (using the *assignment* feature). OpenStack Identity only supports read-only LDAP integration.

The *identity* feature enables administrators to manage users and groups by each domain or the OpenStack Identity service entirely. This is supported by the LDAP identity back end.

The *assignment* feature enables administrators to manage project role authorization using the OpenStack Identity service SQL database. There is no assignment back end for LDAP.

Identity LDAP server set up

Important

If you are using SELinux (enabled by default on RHEL derivatives), then in order for the OpenStack Identity service to access LDAP servers, you must enable the `authlogin_nsswitch_use_ldap` boolean value for SELinux on the server running the OpenStack Identity service. To enable and make the option persistent across reboots, set the following boolean value as the root user:

```
# setsebool -P authlogin_nsswitch_use_ldap on
```

The Identity configuration is split into two separate back ends; identity (back end for users and groups), and assignments (back end for domains, projects, roles, role assignments). To configure Identity, set options in the `/etc/keystone/keystone.conf` file. See *Integrate Identity back end with LDAP* for Identity back end configuration examples. Modify these examples as needed.

To define the destination LDAP server

Define the destination LDAP server in the `/etc/keystone/keystone.conf` file:

```
[ldap]
url = ldap://localhost
user = dc=Manager,dc=example,dc=org
```

(continues on next page)

(continued from previous page)

```
password = samplepassword
suffix = dc=example,dc=org
```

Although its not recommended (see note below), multiple LDAP servers can be supplied to `url` to provide high-availability support for a single LDAP backend. By default, these will be tried in order of apperance, but an additional option, `randomize_urls` can be set to true, to randomize the list in each process (when it starts). To specify multiple LDAP servers, simply change the `url` option in the `[ldap]` section to be a list, separated by commas:

```
url = "ldap://localhost,ldap://backup.localhost"
randomize_urls = true
```

Note

Failover mechanisms in the LDAP backend can cause delays when switching over to the next working LDAP server. Randomizing the order in which the servers are tried only makes the failure behavior not dependent on which of the ordered servers fail. Individual processes can still be delayed or time out, so this doesnt fix the issue at hand, but only makes the failure mode more gradual. This behavior cannot be easily fixed inside the service, because keystone would have to monitor the status of each LDAP server, which is in fact a task for a load balancer. Because of this, it is recommended to use a load balancer in front of the LDAP servers, which can monitor the state of the cluster and instantly redirect connections to the working LDAP server.

Additional LDAP integration settings

Set these options in the `/etc/keystone/keystone.conf` file for a single LDAP server, or `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` files for multiple back ends. Example configurations appear below each setting summary:

Query option

- Use `query_scope` to control the scope level of data presented (search only the first level or search an entire sub-tree) through LDAP.
- Use `page_size` to control the maximum results per page. A value of zero disables paging.
- Use `alias_dereferencing` to control the LDAP dereferencing option for queries.

```
[ldap]
query_scope = sub
page_size = 0
alias_dereferencing = default
chase_referrals =
```

Debug

Use `debug_level` to set the LDAP debugging level for LDAP calls. A value of zero means that debugging is not enabled.

```
[ldap]
debug_level = 4095
```

This setting sets `OPT_DEBUG_LEVEL` in the underlying python library. This field is a bit mask (integer), and the possible flags are documented in the OpenLDAP manpages. Commonly used values include 255 and 4095, with 4095 being more verbose and 0 being disabled. We recommend consulting the documentation for your LDAP back end when using this option.

Warning

Enabling `debug_level` will negatively impact performance.

Connection pooling

Various LDAP back ends use a common LDAP module to interact with LDAP data. By default, a new connection is established for each LDAP operation. This is expensive when TLS support is enabled, which is a likely configuration in an enterprise setup. Reusing connections from a connection pool drastically reduces overhead of initiating a new connection for every LDAP operation.

Use `use_pool` to enable LDAP connection pooling. Configure the connection pool size, maximum retry, reconnect trials, timeout (-1 indicates indefinite wait) and lifetime in seconds.

```
[ldap]
use_pool = true
pool_size = 10
pool_retry_max = 3
pool_retry_delay = 0.1
pool_connection_timeout = -1
pool_connection_lifetime = 600
```

Connection pooling for end user authentication

LDAP user authentication is performed via an LDAP bind operation. In large deployments, user authentication can use up all available connections in a connection pool. OpenStack Identity provides a separate connection pool specifically for user authentication.

Use `use_auth_pool` to enable LDAP connection pooling for end user authentication. Configure the connection pool size and lifetime in seconds. Both `use_pool` and `use_auth_pool` must be enabled to pool connections for user authentication.

```
[ldap]
use_auth_pool = false
auth_pool_size = 100
auth_pool_connection_lifetime = 60
```

When you have finished the configuration, restart the OpenStack Identity service.

Warning

During the service restart, authentication and authorization are unavailable.

Integrate Identity back end with LDAP

The Identity back end contains information for users, groups, and group member lists. Integrating the Identity back end with LDAP allows administrators to use users and groups in LDAP.

Important

For OpenStack Identity service to access LDAP servers, you must define the destination LDAP server in the `/etc/keystone/keystone.conf` file. For more information, see *Identity LDAP server set up*.

To integrate one Identity back end with LDAP

1. Enable the LDAP Identity driver in the `/etc/keystone/keystone.conf` file. This allows LDAP as an identity back end:

```
[identity]
#driver = sql
driver = ldap
```

2. Create the organizational units (OU) in the LDAP directory, and define the corresponding location in the `/etc/keystone/keystone.conf` file:

```
[ldap]
user_tree_dn = ou=Users,dc=example,dc=org
user_objectclass = inetOrgPerson

group_tree_dn = ou=Groups,dc=example,dc=org
group_objectclass = groupOfNames
```

Note

These schema attributes are extensible for compatibility with various schemas. For example, this entry maps to the person attribute in Active Directory:

```
user_objectclass = person
```

Restart the OpenStack Identity service.

Warning

During service restart, authentication and authorization are unavailable.

To integrate multiple Identity back ends with LDAP

1. Set the following options in the `/etc/keystone/keystone.conf` file:
    1. Enable the LDAP driver:

```
[identity]
#driver = sql
driver = ldap
```

2. Enable domain-specific drivers:

```
[identity]
domain_specific_drivers_enabled = True
domain_config_dir = /etc/keystone/domains
```

2. Restart the OpenStack Identity service.

Warning

During service restart, authentication and authorization are unavailable.

3. List the domains using the dashboard, or the OpenStackClient CLI. Refer to the Command List for a list of OpenStackClient commands.
4. Create domains using OpenStack dashboard, or the OpenStackClient CLI.
5. For each domain, create a domain-specific configuration file in the `/etc/keystone/domains` directory. Use the file naming convention `keystone.DOMAIN_NAME.conf`, where DOMAIN_NAME is the domain name assigned in the previous step.

Note

The options set in the `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` file will override options in the `/etc/keystone/keystone.conf` file.

6. Define the destination LDAP server in the `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` file. For example:

```
[ldap]
url = ldap://localhost
user = dc=Manager,dc=example,dc=org
password = samplepassword
suffix = dc=example,dc=org
```

7. Create the organizational units (OU) in the LDAP directories, and define their corresponding locations in the `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` file. For example:

```
[ldap]
user_tree_dn = ou=Users,dc=example,dc=org
user_objectclass = inetOrgPerson

group_tree_dn = ou=Groups,dc=example,dc=org
group_objectclass = groupOfNames
```

Note

These schema attributes are extensible for compatibility with various schemas. For example, this entry maps to the person attribute in Active Directory:

```
user_objectclass = person
```

8. Restart the OpenStack Identity service.

Warning

During service restart, authentication and authorization are unavailable.

Additional LDAP integration settings

Set these options in the `/etc/keystone/keystone.conf` file for a single LDAP server, or `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` files for multiple back ends. Example configurations appear below each setting summary:

Filters

Use filters to control the scope of data presented through LDAP.

```
[ldap]
user_filter = (memberof=cn=openstack-users,ou=workgroups,dc=example,
↪dc=org)
group_filter =
```

Identity attribute mapping

Mask account status values (include any additional attribute mappings) for compatibility with various directory services. Superfluous accounts are filtered with `user_filter`.

Setting attribute ignore to list of attributes stripped off on update.

For example, you can mask Active Directory account status attributes in the `/etc/keystone/keystone.conf` file:

```
[ldap]
user_id_attribute       = cn
user_name_attribute     = sn
user_mail_attribute     = mail
user_pass_attribute     = userPassword
user_enabled_attribute = userAccountControl
user_enabled_mask       = 2
user_enabled_invert     = false
user_enabled_default    = 512
user_default_project_id_attribute =
user_additional_attribute_mapping =

group_id_attribute      = cn
group_name_attribute    = ou
group_member_attribute = member
```

(continues on next page)

(continued from previous page)

```
group_desc_attribute    = description
group_additional_attribute_mapping =
```

It is possible to model more complex LDAP schemas. For example, in the user object, the objectClass posixAccount from RFC2307 is very common. If this is the underlying objectClass, then the `uid` field should probably be `uidNumber` and the `username` field should be either `uid` or `cn`. The following illustrates the configuration:

```
[ldap]
user_id_attribute = uidNumber
user_name_attribute = cn
```

Enabled emulation

OpenStack Identity supports emulation for integrating with LDAP servers that do not provide an `enabled` attribute for users. This allows OpenStack Identity to advertise `enabled` attributes when the user entity in LDAP does not. The `user_enabled_emulation` option must be enabled and the `user_enabled_emulation_dn` option must be a valid LDAP group. Users in the group specified by `user_enabled_emulation_dn` will be marked as `enabled`. For example, the following will mark any user who is a member of the `enabled_users` group as enabled:

```
[ldap]
user_enabled_emulation = True
user_enabled_emulation_dn = cn=enabled_users,cn=groups,dc=openstack,dc=org
```

If the directory server has an enabled attribute, but it is not a boolean type, a mask can be used to convert it. This is useful when the enabled attribute is an integer value. The following configuration highlights the usage:

```
[ldap]
user_enabled_attribute = userAccountControl
user_enabled_mask = 2
user_enabled_default = 512
```

In this case, the attribute is an integer and the enabled attribute is listed in bit 1. If the mask configured `user_enabled_mask` is different from 0, it retrieves the attribute from `user_enabled_attribute` and performs an add operation with the `user_enabled_mask`. If the sum of the operation matches the mask, then the account is disabled.

The value of `user_enabled_attribute` is also saved before applying the add operation in `enabled_nomask`. This is done in case the user needs to be enabled or disabled. Lastly, setting `user_enabled_default` is needed in order to create a default value on the integer attribute (512 = NORMAL ACCOUNT in Active Directory).

When you have finished configuration, restart the OpenStack Identity service.

Warning

During service restart, authentication and authorization are unavailable.

Secure the OpenStack Identity service connection to an LDAP back end

We recommend securing all connections between OpenStack Identity and LDAP. The Identity service supports the use of TLS to encrypt LDAP traffic. Before configuring this, you must first verify where your certificate authority file is located. For more information, see the OpenStack Security Guide SSL introduction.

Once you verify the location of your certificate authority file:

To configure TLS encryption on LDAP traffic

1. Open the `/etc/keystone/keystone.conf` configuration file.
2. Find the `[ldap]` section.
3. In the `[ldap]` section, set the `use_tls` configuration key to `True`. Doing so will enable TLS.
4. Configure the Identity service to use your certificate authorities file. To do so, set the `tls_cacertfile` configuration key in the `ldap` section to the certificate authorities files path.

   Note

   You can also set the `tls_cacertdir` (also in the `ldap` section) to the directory where all certificate authorities files are kept. If both `tls_cacertfile` and `tls_cacertdir` are set, then the latter will be ignored.

5. Specify what client certificate checks to perform on incoming TLS sessions from the LDAP server. To do so, set the `tls_req_cert` configuration key in the `[ldap]` section to `demand`, `allow`, or `never`:

   - `demand` - The LDAP server always receives certificate requests. The session terminates if no certificate is provided, or if the certificate provided cannot be verified against the existing certificate authorities file.
   - `allow` - The LDAP server always receives certificate requests. The session will proceed as normal even if a certificate is not provided. If a certificate is provided but it cannot be verified against the existing certificate authorities file, the certificate will be ignored and the session will proceed as normal.
   - `never` - A certificate will never be requested.

When you have finished configuration, restart the OpenStack Identity service.

Note

If you are unable to connect to LDAP via OpenStack Identity, or observe a *SERVER DOWN* error, set the `TLS_CACERT` in `/etc/ldap/ldap.conf` to the same value specified in the `[ldap] tls_certificate` section of `keystone.conf`.

On distributions that include openstack-config, you can configure TLS encryption on LDAP traffic by running the following commands instead.

```
# openstack-config --set /etc/keystone/keystone.conf \
  ldap use_tls True
```

(continues on next page)

(continued from previous page)

```
# openstack-config --set /etc/keystone/keystone.conf \
  ldap tls_cacertfile ``CA_FILE``
# openstack-config --set /etc/keystone/keystone.conf \
  ldap tls_req_cert ``CERT_BEHAVIOR``
```

Where:

- `CA_FILE` is the absolute path to the certificate authorities file that should be used to encrypt LDAP traffic.
- `CERT_BEHAVIOR` specifies what client certificate checks to perform on an incoming TLS session from the LDAP server (`demand`, `allow`, or `never`).

8.2.5 Caching layer

OpenStack Identity supports a caching layer that is above the configurable subsystems (for example, token). This gives you the flexibility to setup caching for all or some subsystems. OpenStack Identity uses the oslo.cache library which allows flexible cache back ends. The majority of the caching configuration options are set in the `[cache]` section of the `/etc/keystone/keystone.conf` file. The `enabled` option of the `[cache]` section must be set to `True` in order for any subsystem to cache responses. Each section that has the capability to be cached will have a `caching` boolean value that toggles caching behavior of that particular subsystem.

So to enable only the token back end caching, set the values as follows:

```
[cache]
enabled=true

[catalog]
caching=false

[domain_config]
caching=false

[federation]
caching=false

[resource]
caching=false

[revoke]
caching=false

[role]
caching=false

[token]
caching=true
```

Note

Each subsystem is configured to cache by default. However, the global toggle for caching defaults to `False`. A subsystem is only able to cache responses if the global toggle is enabled.

Current functional back ends are:

dogpile.cache.null
A null backend that effectively disables all cache operations.(Default)

dogpile.cache.memcached
Memcached back end using the standard `python-memcached` library.

dogpile.cache.pylibmc
Memcached back end using the `pylibmc` library.

dogpile.cache.bmemcached
Memcached using the `python-binary-memcached` library.

dogpile.cache.redis
Redis back end.

dogpile.cache.dbm
Local DBM file back end.

dogpile.cache.memory
In-memory cache, not suitable for use outside of testing as it does not cleanup its internal cache on cache expiration and does not share cache between processes. This means that caching and cache invalidation will not be consistent or reliable.

dogpile.cache.memory_pickle
In-memory cache, but serializes objects with pickle lib. Its not suitable for use outside of testing. The reason is the same with `dogpile.cache.memory`

oslo_cache.mongo
MongoDB as caching back end.

oslo_cache.memcache_pool
Memcached backend that does connection pooling.

oslo_cache.etcd3gw
Uses etcd 3.x for storage.

oslo_cache.dict
A DictCacheBackend based on dictionary, not suitable for use outside of testing as it does not share cache between processes.This means that caching and cache invalidation will not be consistent or reliable.

Caching for tokens and tokens validation

The token subsystem is OpenStack Identitys most heavily used API. As a result, all types of tokens benefit from caching, including Fernet tokens. Although Fernet tokens do not need to be persisted, they should still be cached for optimal token validation performance.

The token system has a separate `cache_time` configuration option, that can be set to a value above or below the global `expiration_time` default, allowing for different caching behavior from the other systems in OpenStack Identity. This option is set in the `[token]` section of the configuration file.

The token revocation list cache time is handled by the configuration option `revocation_cache_time` in the `[token]` section. The revocation list is refreshed whenever a token is revoked. It typically sees significantly more requests than specific token retrievals or token validation calls.

Here is a list of actions that are affected by the cached time:

- getting a new token
- revoking tokens
- validating tokens
- checking v3 tokens

The delete token API calls invalidate the cache for the tokens being acted upon, as well as invalidating the cache for the revoked token list and the validate/check token calls.

Token caching is configurable independently of the `revocation_list` caching. Lifted expiration checks from the token drivers to the token manager. This ensures that cached tokens will still raise a `TokenNotFound` flag when expired.

For cache consistency, all token IDs are transformed into the short token hash at the provider and token driver level. Some methods have access to the full ID (PKI Tokens), and some methods do not. Cache invalidation is inconsistent without token ID normalization.

Caching for non-token resources

Various other keystone components have a separate `cache_time` configuration option, that can be set to a value above or below the global `expiration_time` default, allowing for different caching behavior from the other systems in Identity service. This option can be set in various sections (for example, `[role]` and `[resource]`) of the configuration file. The create, update, and delete actions for domains, projects and roles will perform proper invalidations of the cached methods listed above.

For more information about the different back ends (and configuration options), see:

- dogpile.cache.memory
- dogpile.cache.memcached

> **Note**
>
> The memory back end is not suitable for use in a production environment.

- dogpile.cache.redis
- dogpile.cache.dbm

Cache invalidation

A common concern with caching is relaying inaccurate information after updating or deleting a resource. Most subsystems within OpenStack Identity invalidate specific cache entries once they have changed. In cases where a specific cache entry cannot be invalidated from the cache, the cache region will be invalidated instead. This invalidates all entries within the cache to prevent returning stale or misleading data. A subsequent request for the resource will be fully processed and cached.

Warning

Be aware that if a read-only back end is in use for a particular subsystem, the cache will not immediately reflect changes performed through the back end. Any given change may take up to the `cache_time` (if set in the subsystem section of the configuration) or the global `expiration_time` (set in the `[cache]` section of the configuration) before it is reflected. If this type of delay is an issue, we recommend disabling caching for that particular subsystem.

Configure the Memcached back end example

The following example shows how to configure the memcached back end:

```
[cache]

enabled = true
backend = dogpile.cache.memcached
backend_argument = url:127.0.0.1:11211
```

You need to specify the URL to reach the `memcached` instance with the `backend_argument` parameter.

Verbose cache logging

We do not recommend using verbose cache logging by default in production systems since its extremely noisy. However, you may need to debug cache issues. One way to see how keystone is interacting with a cache backend is to enhance logging. The following configuration will aggregate oslo and dogpile logs into keystones log file with increased verbosity:

```
[DEFAULT]
default_log_levels = oslo.cache=DEBUG,dogpile.core.dogpile=DEBUG

[cache]
debug_cache_backend = True
```

These logs will include cache hits and misses, making it easier to diagnose cache configuration and connectivity issues.

8.2.6 Security compliance and PCI-DSS

As of the Newton release, the Identity service contains additional security compliance features, specifically to satisfy Payment Card Industry - Data Security Standard (PCI-DSS) v3.1 requirements. See Security Hardening PCI-DSS for more information on PCI-DSS.

Security compliance features are disabled by default and most of the features only apply to the SQL backend for the identity driver. Other identity backends, such as LDAP, should implement their own security controls.

Enable these features by changing the configuration settings under the `[security_compliance]` section in `keystone.conf`.

Setting an account lockout threshold

The account lockout feature limits the number of incorrect password attempts. If a user fails to authenticate after the maximum number of attempts, the service disables the user. Users can be re-enabled by explicitly setting the enable user attribute with the update user v3 API call.

You set the maximum number of failed authentication attempts by setting the `lockout_failure_attempts`:

```
[security_compliance]
lockout_failure_attempts = 6
```

You set the number of minutes a user would be locked out by setting the `lockout_duration` in seconds:

```
[security_compliance]
lockout_duration = 1800
```

If you do not set the `lockout_duration`, users will be locked out indefinitely until the user is explicitly enabled via the API.

You can ensure specific users are never locked out. This can be useful for service accounts or administrative users. You can do this by setting the user option for *ignore_lockout_failure_attempts*.

Disabling inactive users

PCI-DSS 8.1.4 requires that inactive user accounts be removed or disabled within 90 days. You can achieve this by setting the `disable_user_account_days_inactive`:

```
[security_compliance]
disable_user_account_days_inactive = 90
```

This above example means that users that have not authenticated (inactive) for the past 90 days are automatically disabled. Users can be re-enabled by explicitly setting the enable user attribute via the API.

Force users to change password upon first use

PCI-DSS 8.2.6 requires users to change their password for first time use and upon an administrative password reset. Within the identity user API, *create user* and *update user* are considered administrative password changes. Whereas, *change password for user* is a self-service password change. Once this feature is enabled, new users, and users that have had their password reset, will be required to change their password upon next authentication (first use), before being able to access any services.

Prior to enabling this feature, you may want to exempt some users that you do not wish to be required to change their password. You can mark a user as exempt by setting the user options attribute *ignore_change_password_upon_first_use*.

Warning

Failure to mark service users as exempt from this requirement will result in your service account passwords becoming expired after being reset.

When ready, you can configure it so that users are forced to change their password upon first use by setting `change_password_upon_first_use`:

```
[security_compliance]
change_password_upon_first_use = True
```

Configuring password expiration

Passwords can be configured to expire within a certain number of days by setting the `password_expires_days`:

```
[security_compliance]
password_expires_days = 90
```

Once set, any new password changes have an expiration date based on the date/time of the password change plus the number of days defined here. Existing passwords will not be impacted. If you want existing passwords to have an expiration date, you would need to run a SQL script against the password table in the database to update the expires_at column.

If there exists a user whose password you do not want to expire, keystone supports setting that via the user option *ignore_password_expiry*.

Configuring password strength requirements

You can set password strength requirements, such as requiring numbers in passwords or setting a minimum password length, by adding a regular expression to the `password_regex` setting:

```
[security_compliance]
password_regex = ^(?=.*\d)(?=.*[a-zA-Z]).{7,}$
```

The above example is a regular expression that requires a password to have:

- One (1) letter

- One (1) digit
- Minimum length of seven (7) characters

If you do set the `password_regex`, you should provide text that describes your password strength requirements. You can do this by setting the `password_regex_description`:

```
[security_compliance]
password_regex_description = Passwords must contain at least 1 letter, 1
                             digit, and be a minimum length of 7
                             characters.
```

It is imperative that the `password_regex_description` matches the actual regex. If the `password_regex` and the `password_regex_description` do not match, it will cause user experience to suffer since this description will be returned to users to explain why their requested password was insufficient.

Note

You must ensure the `password_regex_description` accurately and completely describes the `password_regex`. If the two options are out of sync, the help text could inaccurately describe the password requirements being applied to the password. This would lead to a poor user experience.

Requiring a unique password history

The password history requirements controls the number of passwords for a user that must be unique before an old password can be reused. You can enforce this by setting the `unique_last_password_count`:

```
[security_compliance]
unique_last_password_count= 5
```

The above example does not allow a user to create a new password that is the same as any of their last four previous passwords.

Similarly, you can set the number of days that a password must be used before the user can change it by setting the `minimum_password_age`:

```
[security_compliance]
minimum_password_age = 1
```

In the above example, once a user changes their password, they would not be able to change it again for one day. This prevents users from changing their passwords immediately in order to wipe out their password history and reuse an old password.

Note

When you set `password_expires_days`, the value for the `minimum_password_age` should be less than the `password_expires_days`. Otherwise, users would not be able to change their passwords before they expire.

Prevent Self-Service Password Changes

If there exists a user who should not be able to change her own password via the keystone password change API, keystone supports setting that via the user option *lock_password*.

This is typically used in the case where passwords are managed externally to keystone.

8.2.7 Performance and scaling

Before you begin tuning Keystone for performance and scalability, you should first know that Keystone is just a two tier horizontally-scalable web application, and the most effective methods for scaling it are going to be the same as for any other similarly designed web application: give it more processes, more memory, scale horizontally, and load balance the result.

With that said, there are many opportunities for tuning the performance of Keystone, many of which are actually trade-offs between performance and security that you need to judge for yourself, and tune accordingly.

Keystone configuration options that affect performance

These are all of the options in `keystone.conf` that have a direct impact on performance. See the help descriptions for these options for more specific details on how and why you might want to tune these options for yourself.

- `[DEFAULT] max_project_tree_depth`: Reduce this number to increase performance, increase this number to cater to more complicated hierarchical multitenancy use cases.
- `[DEFAULT] max_password_length`: Reduce this number to increase performance, increase this number to allow for more secure passwords.
- `[cache] enable`: Enable this option to increase performance, but you also need to configure other options in the `[cache]` section to actually utilize caching.
- `[token] provider`: All supported token providers have been primarily driven by performance considerations. UUID and Fernet both require online validation (cacheable HTTP calls back to keystone to validate tokens). Fernet has the highest scalability characteristics overall, but requires more work to validate, and therefore enabling caching (`[cache] enable`) is absolutely critical.
- `[fernet] max_active_keys`: If youre using Fernet tokens, decrease this option to improve performance, increase this option to support more advanced key rotation strategies.

Keystonemiddleware configuration options that affect performance

This configuration actually lives in the Paste pipelines of services consuming token validation from keystone (i.e.: nova, cinder, swift, etc.).

- `cache`: When keystones *auth_token* middleware is deployed with a swift cache, use this option to have *auth_token* middleware share a caching backend with swift. Otherwise, use the `memcached_servers` option instead.
- `memcached_servers`: Set this option to share a cache across `keystonemiddleware.auth_token` processes.

- `token_cache_time`: Increase this option to improve performance, decrease this option to respond to token revocation events more quickly (thereby increasing security).
- `revocation_cache_time`: Increase this option to improve performance, decrease this option to respond to token revocation events more quickly (thereby increasing security).
- `memcache_security_strategy`: Do not set this option to improve performance, but set it to improve security where youre sharing memcached with other processes.
- `include_service_catalog`: Disable this option to improve performance, if the protected service does not require a service catalog.

8.2.8 URL safe naming of projects and domains

In the future, keystone may offer the ability to identify a project in a hierarchy via a URL style of naming from the root of the hierarchy (for example specifying projectA/projectB/projectC as the project name in an authentication request). In order to prepare for this, keystone supports the optional ability to ensure both projects and domains are named without including any of the reserved characters specified in section 2.2 of rfc3986.

The safety of the names of projects and domains can be controlled via two configuration options:

```
[resource]
project_name_url_safe = off
domain_name_url_safe = off
```

When set to `off` (which is the default), no checking is done on the URL safeness of names. When set to `new`, an attempt to create a new project or domain with an unsafe name (or update the name of a project or domain to be unsafe) will cause a status code of 400 (Bad Request) to be returned. Setting the configuration option to `strict` will, in addition to preventing the creation and updating of entities with unsafe names, cause an authentication attempt which specifies a project or domain name that is unsafe to return a status code of 401 (Unauthorized).

It is recommended that installations take the steps necessary to where they can run with both options set to `strict` as soon as is practical.

8.2.9 Limiting list return size

Keystone provides a method of setting a limit to the number of entities returned in a collection, which is useful to prevent overly long response times for list queries that have not specified a sufficiently narrow filter. This limit can be set globally by setting `list_limit` in the default section of `keystone.conf`, with no limit set by default. Individual driver sections may override this global value with a specific limit, for example:

```
[resource]
list_limit = 100
```

If a response to `list_{entity}` call has been truncated, then the response status code will still be 200 (OK), but the `truncated` attribute in the collection will be set to `true`.

8.2.10 Endpoint Filtering

Endpoint Filtering enables creation of ad-hoc catalogs for each project-scoped token request.

Configure the endpoint filter catalog driver in the `[catalog]` section. For example:

```
[catalog]
driver = catalog_sql
```

In the `[endpoint_filter]` section, set `return_all_endpoints_if_no_filter` to `False` to return an empty catalog if no associations are made. For example:

```
[endpoint_filter]
return_all_endpoints_if_no_filter = False
```

See API Specification for Endpoint Filtering for the details of API definition.

8.2.11 Endpoint Policy

The Endpoint Policy feature provides associations between service endpoints and policies that are already stored in the Identity server and referenced by a policy ID.

Configure the endpoint policy backend driver in the `[endpoint_policy]` section. For example:

```
[endpoint_policy]
driver = sql
```

See API Specification for Endpoint Policy for the details of API definition.

8.3 Keystone Operations

Guides for managing day-to-day operations of keystone and understanding your deployment.

8.3.1 Upgrading Keystone

As of the Newton release, keystone supports two different approaches to upgrading across releases. The traditional approach requires a significant outage to be scheduled for the entire duration of the upgrade process. The more modern approach results in zero downtime, but is more complicated due to a longer upgrade procedure.

Note

The details of these steps are entirely dependent on the details of your specific deployment, such as your chosen application server and database management system. Use it only as a guide when implementing your own upgrade process.

Before you begin

Plan your upgrade:

- Read and ensure you understand the release notes for the next release.
- Resolve any outstanding deprecation warnings in your logs. Some deprecation cycles are as short as a single release, so its possible to break a deployment if you leave *any* outstanding warnings. It might be a good idea to re-read the release notes for the previous release (or two!).
- Prepare your new configuration files, including `keystone.conf`, `logging.conf`, `policy.yaml`, `keystone-paste.ini`, and anything else in `/etc/keystone/`, by customizing the corresponding files from the next release.

Upgrading with downtime

This is a high-level description of our upgrade strategy built around `keystone-manage db_sync`. It assumes that you are willing to have downtime of your control plane during the upgrade process and presents minimal risk. With keystone unavailable, no other OpenStack services will be able to authenticate requests, effectively preventing the rest of the control plane from functioning normally.

1. Stop all keystone processes. Otherwise, youll risk multiple releases of keystone trying to write to the database at the same time, which may result in data being inconsistently written and read.
2. Make a backup of your database. Keystone does not support downgrading the database, so restoring from a full backup is your only option for recovery in the event of an upgrade failure.
3. Upgrade all keystone nodes to the next release.
4. Update your configuration files (`/etc/keystone/`) with those corresponding from the latest release.
5. Run `keystone-manage db_sync` from any single node to upgrade both the database schema and run any corresponding database migrations.
6. (*New in Newton*) Run `keystone-manage doctor` to diagnose symptoms of common deployment issues and receive instructions for resolving them.
7. Start all keystone processes.

Upgrading with minimal downtime

If you run a multi-node keystone cluster that uses a replicated database, like a Galera cluster, it is possible to upgrade with minimal downtime. This method also optimizes recovery time from a failed upgrade. This section assumes familiarity with the base case (*Upgrading with downtime*) outlined above. In these steps the nodes will be divided into `first` and `other` nodes.

1. Backup your database. There is no way to rollback the upgrade of keystone and this is your worst-case fallback option.
2. Disable keystone on all nodes but the `first` node. This can be done via a variety of mechanisms that will depend on the deployment. If you are unable to disable a service or place a service into maintenance mode in your load balancer, you can stop the keystone processes.
3. Stop the database service on one of the `other` nodes in the cluster. This will isolate the old dataset on a single node in the cluster. In the event of a failed update this data can be used to rebuild the cluster without having to restore from backup.

4. Update the configuration files on the `first` node.
5. Upgrade keystone on the `first` node. keystone is now down for your cloud.
6. Run `keystone-manage db_sync` on the `first` node. As soon as this finishes, keystone is now working again on a single node in the cluster.
7. keystone is now upgraded on a single node. Your load balancers will be sending all traffic to this single node. This is your chance to run ensure keystone up and running, and not broken. If keystone is broken, see the *Rollback after a failed upgrade* section below.
8. Once you have verified that keystone is up and running, begin the upgrade on the `other` nodes. This entails updating configuration files and upgrading the code. The `db_sync` does not need to be run again.
9. On the node where you stopped the database service, be sure to restart it and ensure that it properly rejoins the cluster.

Using this model, the outage window is minimized because the only time when your cluster is totally offline is between loading the newer version of keystone and running the `db_sync` command. Typically the outage with this method can be measured in tens of seconds especially if automation is used.

Rollback after a failed upgrade

If the upgrade fails, only a single node has been affected. This makes the recovery simpler and quicker. If issues are not discovered until the entire cluster is upgraded, a full shutdown and restore from backup will be required. That will take much longer than just fixing a single node with an old copy of the database still available. This process will be dependent on your architecture and it is highly recommended that youve practiced this in a development environment before trying to use it for the first time.

1. Isolate the bad node. Shutdown keystone and the database services on the upgraded bad node.
2. Bootstrap the database cluster from the node holding the old data. This may require wiping the data first on any nodes who are not holding old data.
3. Enable keystone on the old nodes in your load balancer or if the processes were stopped, restart them.
4. Validate that keystone is working.
5. Downgrade the code and config files on the bad node.

This process should be doable in a matter of minutes and will minimize cloud downtime if it is required.

Upgrading without downtime

Added in version 10.0.0: (Newton)

Upgrading without downtime is only supported in deployments upgrading *from* Newton or a newer release.

If upgrading a Mitaka deployment to Newton, the commands described here will be available as described below, but the `keystone-manage db_sync --expand` command will incur downtime (similar to running `keystone-manage db_sync`), as it runs legacy (downtime-incurring) migrations prior to running schema expansions.

Changed in version 21.0.0: (Yoga)

The migration tooling was changed from *SQLAlchemy-Migrate* to *Alembic*. As part of this change, the data migration phase of the database upgrades was dropped.

This is a high-level description of our upgrade strategy built around additional options in `keystone-manage db_sync`. Although it is much more complex than the upgrade process described above, it assumes that you are not willing to have downtime of your control plane during the upgrade process. With this upgrade process, end users will still be able to authenticate to receive tokens normally, and other OpenStack services will still be able to authenticate requests normally.

1. Make a backup of your database. keystone does not support downgrading the database, so restoring from a full backup is your only option for recovery in the event of an upgrade failure.
2. Stop the keystone processes on the first node (or really, any arbitrary node). This node will serve to orchestrate database upgrades.
3. Upgrade your first node to the next release, but do not start any keystone processes.
4. Update your configuration files on the first node (`/etc/keystone/`) with those corresponding to the latest release.
5. Run `keystone-manage doctor` on the first node to diagnose symptoms of common deployment issues and receive instructions for resolving them.
6. Run `keystone-manage db_sync --expand` on the first node to expand the database schema to a superset of what both the previous and next release can utilize, and create triggers to facilitate the live migration process.

   Warning

   For MySQL, using the `keystone-manage db_sync --expand` command requires that you either grant your keystone user `SUPER` privileges, or run `set global log_bin_trust_function_creators=1;` in mysql beforehand.

   At this point, new columns and tables may exist in the database, but will *not* all be populated in such a way that the next release will be able to function normally.

   As the previous release continues to write to the old schema, database triggers will live migrate the data to the new schema so it can be read by the next release.

   Note

   Prior to Yoga, data migrations were treated separatly and required the use of the `keystone-manage db_sync --migrate` command after applying the expand migrations. This is no longer necessary and the `keystone-manage db_sync --migrate` command is now a no-op.

7. Update your configuration files (`/etc/keystone/`) on all nodes (except the first node, which youve already done) with those corresponding to the latest release.
8. Upgrade all keystone nodes to the next release, and restart them one at a time. During this step, youll have a mix of releases operating side by side, both writing to the database.

   As the next release begins writing to the new schema, database triggers will also migrate the data to the old schema, keeping both data schemas in sync.

9. Run `keystone-manage db_sync --contract` to remove the old schema and all data migration triggers.

   When this process completes, the database will no longer be able to support the previous release.

Using `db_sync check`

Added in version 12.0.0: (Pike)

Changed in version 21.0.0: (Yoga)

Previously this command would return `3` if data migrations were required. Data migrations are now part of the expand schema migrations, therefore this step is no longer necessary.

In order to check the current state of your rolling upgrades, you may run the command `keystone-manage db_sync --check`. This will inform you of any outstanding actions you have left to take as well as any possible upgrades you can make from your current version. Here are a list of possible return codes.

- A return code of `0` means you are currently up to date with the latest migration script version and all `db_sync` commands are complete.
- A return code of `1` generally means something serious is wrong with your database and operator intervention will be required.
- A return code of `2` means that an upgrade from your current database version is available, your database is not currently under version control, or the database is already under control. Your first step is to run `keystone-manage db_sync --expand`.
- A return code of `4` means that the expansion and data migration stages are complete, and the next step is to run `keystone-manage db_sync --contract`.

8.3.2 Case-Insensitivity in keystone

Keystone currently handles the case-sensitivity for the naming of each resource a bit differently, depending on the resource itself, and the backend used. For example, depending on whether a user is backed by local SQL or LDAP, the case-sensitivity can be different. When it is case-insensitive, the casing will be preserved. For instance, a project with the name myProject will not end up changing to either all lower or upper case.

Resources in keystone

Below are examples of case-insensitivity in keystone for users, projects, and roles.

Users

If a user with the name MyUser already exists, then the following call which creates a new user by the name of myuser will return a `409 Conflict`:

```
POST /v3/users
```

```
{
    "user": {
        "name": "myuser"
    }
}
```

Projects

If a project with the name Foobar already exists, then the following call which creates a new project by the name of foobar will return a `409 Conflict`:

```
POST /v3/projects
```

```
{
    "project": {
        "name": "foobar"
    }
}
```

Project Tags

While project names are case-insensitive, project tags are case-sensitive. A tag with the value of `mytag` is different than `MyTag`, and both values can be stored in the same project.

Roles

Role names are case-insensitive. for example, when keystone bootstraps default roles, it creates admin, member, and reader. If another role, Member (note the upper case M) is created, keystone will return a `409 Conflict` since it considers the name Member equivalent to member. Note that case is preserved in this event.

Note

As of the Rocky release, keystone will create three default roles when *keystone-manage bootstrap* is run: (`admin`, `member`, `reader`). For existing deployments, this can cause issues if an existing role matches one of these roles. Even if the casing is not an exact match (`member` vs `Member`), it will report an error since roles are considered case-insensitive.

Backends

For each of these examples, we will refer to an existing project with the name mYpRoJeCt and user with the name mYuSeR. The examples here are exaggerated to help display the case handling for each backend.

MySQL & SQLite

By default, MySQL/SQLite are case-insensitive but case-preserving for *varchar*. This means that setting a project name of mYpRoJeCt will cause attempting to create a new project named myproject to fail with keystone returning a `409 Conflict`. However, the original value of mYpRoJeCt will still be returned since case is preserved.

Users will be treated the same, if another user is added with the name myuser, keystone will respond with `409 Conflict` since another user with the (same) name exists (mYuSeR).

PostgreSQL

PostgreSQL is case-sensitive by default, so if a project by the name of myproject is created with the existing mYpRoJeCt, it will be created successfully.

LDAP

By default, LDAP DNs are case-insensitive, so the example with users under MySQL will apply here as well.

8.3.3 Managing trusts

A trust is an OpenStack Identity extension that enables delegation and, optionally, impersonation through `keystone`. See the *user guide on using trusts*.

Removing Expired Trusts

In the SQL trust stores expired and soft deleted trusts, that are not automatically removed. These trusts can be removed with:

```
   $ keystone-manage trust_flush [options]

OPTIONS (optional):

       --project-id <string>:
                   To purge trusts of given project-id.
       --trustor-user-id <string>:
                   To purge trusts of given trustor-id.
       --trustee-user-id <string>:
                   To purge trusts of given trustee-id.
       --date <string>:
```

(continues on next page)

(continued from previous page)

```
                    To purge trusts older than date. If no date is supplied
                    keystone-manage will use the system clock time at runtime.
```

8.4 All about keystone tokens

Everything you need to know about keystone tokens.

8.4.1 Keystone tokens

Tokens are used to authenticate and authorize your interactions with OpenStack APIs. Tokens come in many scopes, representing various authorization and sources of identity.

Authorization scopes

Tokens are used to relay information about your role assignments. Its not uncommon for a user to have multiple role assignments, sometimes spanning projects, domains, or the entire system. These are referred to as authorization scopes, where a token has a single scope of operation (e.g., a project, domain, or the system). For example, a token scoped to a project cant be reused to do something else in a different project.

Each level of authorization scope is useful for certain types of operations in certain OpenStack services, and are not interchangeable.

Unscoped tokens

An unscoped token does not contain a service catalog, roles, or authorization scope (e.g., project, domain, or system attributes within the token). Their primary use case is simply to prove your identity to keystone at a later time (usually to generate scoped tokens), without repeatedly presenting your original credentials.

The following conditions must be met to receive an unscoped token:

- You must not specify an authorization scope in your authentication request (for example, on the command line with arguments such as `--os-project-name` or `--os-domain-id`),
- Your identity must not have a default project associated with it that you also have role assignments, and thus authorization, upon.

Project-scoped tokens

Projects are containers for resources, like volumes or instances. Project-scoped tokens express your authorization to operate in a specific tenancy of the cloud and are useful for things like spinning up compute resources or carving off block storage. They contain a service catalog, a set of roles, and information about the project.

Most end-users need role assignments on projects to consume resources in a deployment.

Domain-scoped tokens

Domains are namespaces for projects, users, and groups. A domain-scoped token expresses your authorization to operate on the contents of a domain or the domain itself.

While some OpenStack services are still adopting the domain concept, domains are fully supported in keystone. This means users with authorization on a domain have the ability to manage things within the domain. For example, a domain administrator can create new users and projects within that domain.

Domain-scoped tokens contain a service catalog, roles, and information about the domain.

People who need to manage users and projects typically need domain-level access.

System-scoped tokens

Some OpenStack APIs fit nicely within the concept of projects (e.g., creating an instance) or domains (e.g., creating a new user), but there are also APIs that affect the entire deployment system (e.g. modifying endpoints, service management, or listing information about hypervisors). These operations are typically reserved for operators and require system-scoped tokens, which represents the role assignments a user has to operate on the deployment as a whole. The term *system* refers to the deployment system, which is a collection of hardware (e.g., compute nodes) and services (e.g., nova, cinder, neutron, barbican, keystone) that provide Infrastructure-as-a-Service.

System-scoped tokens contain a service catalog, roles, and information about the *system*. System role assignments and system-scoped tokens are typically reserved for operators and cloud administrators.

Token providers

The token type issued by keystone is configurable through the `/etc/keystone/keystone.conf` file. Currently, there are two supported token providers, `fernet` and `jws`.

Fernet tokens

The fernet token format was introduced in the OpenStack Kilo release and now is the default token provider in Keystone. Unlike the other token types mentioned in this document, fernet tokens do not need to be persisted in a back end. `AES256` encryption is used to protect the information stored in the token and integrity is verified with a `SHA256 HMAC` signature. Only the Identity service should have access to the keys used to encrypt and decrypt fernet tokens. Like UUID tokens, fernet tokens must be passed back to the Identity service in order to validate them. For more information on the fernet token type, see the *Fernet - Frequently Asked Questions*.

A deployment might consider using the fernet provider as opposed to JWS tokens if they are concerned about public expose of the payload used to build tokens.

JWS tokens

The JSON Web Signature (JWS) token format is a type of JSON Web Token (JWT) and it was implemented in the Stein release. JWS tokens are signed, meaning the information used to build the token ID is not opaque to users and can it can be decoded by anyone. JWS tokens are ephemeral, or non-persistent, which means they wont bloat the database or require replication across nodes. Since the JWS token provider uses asymmetric keys, the tokens are signed with private keys and validated with public keys. The JWS token provider implementation only supports the `ES256` JSON Web Algorithm (JWA), which is an Elliptic Curve Digital Signature Algorithm (ECDSA) using the P-256 curve and a SHA-256 hash algorithm.

A deployment might consider using JWS tokens as opposed to fernet tokens if there are security concerns about sharing symmetric encryption keys across hosts. Note that a major difference between the two providers is that JWS tokens are not opaque and can be decoded by anyone with the token ID. Fernet tokens are opaque in that the token ID is ciphertext. Despite the JWS token payload being readable by anyone, keystone reserves the right to make backwards incompatible changes to the token payload itself, which is not an API contract. We only recommend validating the token against keystones authentication API to inspect its associated metadata. We strongly discourage relying on decoded payloads for information about tokens.

More information about JWTs can be found in the specification. Summary

Feature	*Status*	**Fernet tokens**	**JWS tokens**
Create unscoped token	mandatory	✓	✓
Create system-scoped token	mandatory	✓	✓
Create project-scoped token	mandatory	✓	✓
Create domain-scoped token	optional	✓	✓
Create trust-scoped token	optional	✓	✓
Create a token given an OAuth access token	optional	✓	✓
Revoke a token	optional	✓	✓

Details

- **Create unscoped token Status: mandatory.**

  CLI commands:

  - `openstack --os-username=<username> --os-user-domain-name=<domain> --os-password=<password> token issue`

  Notes: All token providers must be capable of issuing tokens without an explicit scope of authorization.

  Driver Support:

  - **Fernet tokens:** `complete`
  - **JWS tokens:** `complete`

- **Create system-scoped token Status: mandatory.**

  CLI commands:

  - `openstack --os-username=<username> --os-user-domain-name=<domain> --os-system-scope all token issue`

Notes: All token providers must be capable of issuing system-scoped tokens.

Driver Support:

- **Fernet tokens:** `complete`
- **JWS tokens:** `complete`

- **Create project-scoped token Status: mandatory.**

  CLI commands:

  - `openstack --os-username=<username> --os-user-domain-name=<domain> --os-password=<password> --os-project-name=<project> --os-project-domain-name=<domain> token issue`

  Notes: All token providers must be capable of issuing project-scoped tokens.

  Driver Support:

  - **Fernet tokens:** `complete`
  - **JWS tokens:** `complete`

- **Create domain-scoped token Status: optional.**

  CLI commands:

  - `openstack --os-username=<username> --os-user-domain-name=<domain> --os-password=<password> --os-domain-name=<domain> token issue`

  Notes: Domain-scoped tokens are not required for all use cases, and for some use cases, projects can be used instead.

  Driver Support:

  - **Fernet tokens:** `complete`
  - **JWS tokens:** `complete`

- **Create trust-scoped token Status: optional.**

  CLI commands:

  - `openstack --os-username=<username> --os-user-domain-name=<domain> --os-password=<password> --os-trust-id=<trust> token issue`

  Notes: Tokens scoped to a trust convey only the user impersonation and project-based authorization attributes included in the delegation.

  Driver Support:

  - **Fernet tokens:** `complete`
  - **JWS tokens:** `complete`

- **Create a token given an OAuth access token Status: optional.**

  Notes: OAuth access tokens can be exchanged for keystone tokens.

  Driver Support:

  - **Fernet tokens:** `complete`
  - **JWS tokens:** `complete`

- **Revoke a token Status: optional.**

  CLI commands:

  - `openstack token revoke`

  Notes: Tokens may be individually revoked, such as when a user logs out of Horizon. Under certain circumstances, its acceptable for more than just a single token may be revoked as a result of this operation (such as when the revoked token was previously used to create additional tokens).

  Driver Support:

  - **Fernet tokens:** `complete`
  - **JWS tokens:** `complete`

Notes:

- **This document is a continuous work in progress**

8.4.2 Fernet - Frequently Asked Questions

The following questions have been asked periodically since the initial release of the fernet token format in Kilo.

What is a fernet token?

A fernet token is a bearer token that represents user authentication. Fernet tokens contain a limited amount of identity and authorization data in a MessagePacked payload. The payload is then wrapped as a Fernet message for transport, where Fernet provides the required web safe characteristics for use in URLs and headers. The data inside a fernet token is protected using symmetric encryption keys, or fernet keys.

What is a fernet key?

A fernet key is used to encrypt and decrypt fernet tokens. Each key is actually composed of two smaller keys: a 128-bit AES encryption key and a 128-bit SHA256 HMAC signing key. The keys are held in a key repository that keystone passes to a library that handles the encryption and decryption of tokens.

What are the different types of keys?

A key repository is required by keystone in order to create fernet tokens. These keys are used to encrypt and decrypt the information that makes up the payload of the token. Each key in the repository can have one of three states. The state of the key determines how keystone uses a key with fernet tokens. The different types are as follows:

Primary key:
: There is only ever one primary key in a key repository. The primary key is allowed to encrypt and decrypt tokens. This key is always named as the highest index in the repository.

Secondary key:
: A secondary key was at one point a primary key, but has been demoted in place of another primary key. It is only allowed to decrypt tokens. Since it was the primary at some point in time, its

existence in the key repository is justified. Keystone needs to be able to decrypt tokens that were created with old primary keys.

Staged key:
The staged key is a special key that shares some similarities with secondary keys. There can only ever be one staged key in a repository and it must exist. Just like secondary keys, staged keys have the ability to decrypt tokens. Unlike secondary keys, staged keys have never been a primary key. In fact, they are opposites since the staged key will always be the next primary key. This helps clarify the name because they are the next key staged to be the primary key. This key is always named as `0` in the key repository.

So, how does a staged key help me and why do I care about it?

The fernet keys have a natural lifecycle. Each key starts as a staged key, is promoted to be the primary key, and then demoted to be a secondary key. New tokens can only be encrypted with a primary key. Secondary and staged keys are never used to encrypt token. The staged key is a special key given the order of events and the attributes of each type of key. The staged key is the only key in the repository that has not had a chance to encrypt any tokens yet, but it is still allowed to decrypt tokens. As an operator, this gives you the chance to perform a key rotation on one keystone node, and distribute the new key set over a span of time. This does not require the distribution to take place in an ultra short period of time. Tokens encrypted with a primary key can be decrypted, and validated, on other nodes where that key is still staged.

Where do I put my key repository?

The key repository is specified using the `key_repository` option in the keystone configuration file. The keystone process should be able to read and write to this location but it should be kept secret otherwise. Currently, keystone only supports file-backed key repositories.

```
[fernet_tokens]
key_repository = /etc/keystone/fernet-keys/
```

What is the recommended way to rotate and distribute keys?

The **keystone-manage** command line utility includes a key rotation mechanism. This mechanism will initialize and rotate keys but does not make an effort to distribute keys across keystone nodes. The distribution of keys across a keystone deployment is best handled through configuration management tooling, however ensure that the new primary key is distributed first. Use **keystone-manage fernet_rotate** to rotate the key repository.

Do fernet tokens still expire?

Yes, fernet tokens can expire just like any other keystone token formats.

Why should I choose fernet tokens over UUID tokens?

Even though fernet tokens operate very similarly to UUID tokens, they do not require persistence or leverage the configured token persistence driver in any way. The keystone token database no longer suffers bloat as a side effect of authentication. Pruning expired tokens from the token database is no longer required when using fernet tokens. Because fernet tokens do not require persistence, they do not have to be replicated. As long as each keystone node shares the same key repository, fernet tokens can be created and validated instantly across nodes.

Why should I choose fernet tokens over PKI or PKIZ tokens?

The arguments for using fernet over PKI and PKIZ remain the same as UUID, in addition to the fact that fernet tokens are much smaller than PKI and PKIZ tokens. PKI and PKIZ tokens still require persistent storage and can sometimes cause issues due to their size. This issue is mitigated when switching to fernet because fernet tokens are kept under a 250 byte limit. PKI and PKIZ tokens typically exceed 1600 bytes in length. The length of a PKI or PKIZ token is dependent on the size of the deployment. Bigger service catalogs will result in longer token lengths. This pattern does not exist with fernet tokens because the contents of the encrypted payload is kept to a minimum.

Should I rotate and distribute keys from the same keystone node every rotation?

No, but the relationship between rotation and distribution should be lock-step. Once you rotate keys on one keystone node, the key repository from that node should be distributed to the rest of the cluster. Once you confirm that each node has the same key repository state, you could rotate and distribute from any other node in the cluster.

If the rotation and distribution are not lock-step, a single keystone node in the deployment will create tokens with a primary key that no other node has as a staged key. This will cause tokens generated from one keystone node to fail validation on other keystone nodes.

How do I add new keystone nodes to a deployment?

The keys used to create fernet tokens should be treated like super secret configuration files, similar to an SSL secret key. Before a node is allowed to join an existing cluster, issuing and validating tokens, it should have the same key repository as the rest of the nodes in the cluster.

How should I approach key distribution?

Remember that key distribution is only required in multi-node keystone deployments. If you only have one keystone node serving requests in your deployment, key distribution is unnecessary.

Key distribution is a problem best approached from the deployments current configuration management system. Since not all deployments use the same configuration management systems, it makes sense to explore options around what is already available for managing keys, while keeping the secrecy of the keys in mind. Many configuration management tools can leverage something like `rsync` to manage key distribution.

Key rotation is a single operation that promotes the current staged key to primary, creates a new staged key, and prunes old secondary keys. It is easiest to do this on a single node and verify the rotation took place properly before distributing the key repository to the rest of the cluster. The concept behind the staged key breaks the expectation that key rotation and key distribution have to be done in a single step. With the staged key, we have time to inspect the new key repository before syncing state with the rest of the cluster. Key distribution should be an operation that can run in succession until it succeeds. The following might help illustrate the isolation between key rotation and key distribution.

1. Ensure all keystone nodes in the deployment have the same key repository.
2. Pick a keystone node in the cluster to rotate from.
3. Rotate keys.
    1. Was it successful?
        1. If no, investigate issues with the particular keystone node you rotated keys on. Fernet keys are small and the operation for rotation is trivial. There should not be much room for error in key rotation. It is possible that the user does not have the ability to write new keys to the key repository. Log output from `keystone-manage fernet_rotate` should give more information into specific failures.
        2. If yes, you should see a new staged key. The old staged key should be the new primary. Depending on the `max_active_keys` limit you might have secondary keys that were pruned. At this point, the node that you rotated on will be creating fernet tokens with a primary key that all other nodes should have as the staged key. This is why we checked the state of all key repositories in Step one. All other nodes in the cluster should be able to decrypt tokens created with the new primary key. At this point, we are ready to distribute the new key set.
4. Distribute the new key repository.
    1. Was it successful?
        1. If yes, you should be able to confirm that all nodes in the cluster have the same key repository that was introduced in Step 3. All nodes in the cluster will be creating tokens with the primary key that was promoted in Step 3. No further action is required until the next schedule key rotation.

2. If no, try distributing again. Remember that we already rotated the repository and performing another rotation at this point will result in tokens that cannot be validated across certain hosts. Specifically, the hosts that did not get the latest key set. You should be able to distribute keys until it is successful. If certain nodes have issues syncing, it could be permission or network issues and those should be resolved before subsequent rotations.

How long should I keep my keys around?

The fernet tokens that keystone creates are only secure as the keys creating them. With staged keys the penalty of key rotation is low, allowing you to err on the side of security and rotate weekly, daily, or even hourly. Ultimately, this should be less time than it takes an attacker to break a `AES256` key and a `SHA256` `HMAC`.

Is a fernet token still a bearer token?

Yes, and they follow exactly the same validation path as UUID tokens, with the exception of being written to, and read from, a back end. If someone compromises your fernet token, they have the power to do all the operations you are allowed to do.

What if I need to revoke all my tokens?

To invalidate every token issued from keystone and start fresh, remove the current key repository, create a new key set, and redistribute it to all nodes in the cluster. This will render every token issued from keystone as invalid regardless if the token has actually expired. When a client goes to re-authenticate, the new token will have been created with a new fernet key.

What can an attacker do if they compromise a fernet key in my deployment?

If any key used in the key repository is compromised, an attacker will be able to build their own tokens. If they know the ID of an administrator on a project, they could generate administrator tokens for the project. They will be able to generate their own tokens until the compromised key has been removed from the repository.

I rotated keys and now tokens are invalidating early, what did I do?

Using fernet tokens requires some awareness around token expiration and the key lifecycle. You do not want to rotate so often that secondary keys are removed that might still be needed to decrypt unexpired tokens. If this happens, you will not be able to decrypt the token because the key the was used to encrypt it is now gone. Only remove keys that you know are not being used to encrypt or decrypt tokens.

For example, your token is valid for 24 hours and we want to rotate keys every six hours. We will need to make sure tokens that were created at 08:00 AM on Monday are still valid at 07:00 AM on Tuesday, assuming they were not prematurely revoked. To accomplish this, we will want to make sure we set `max_active_keys=6` in our keystone configuration file. This will allow us to hold all keys that might still be required to validate a previous token, but keeps the key repository limited to only the keys that are needed.

The number of `max_active_keys` for a deployment can be determined by dividing the token lifetime, in hours, by the frequency of rotation in hours and adding two. Better illustrated as:

```
token_expiration = 24
rotation_frequency = 6
max_active_keys = (token_expiration / rotation_frequency) + 2
```

The reason for adding two additional keys to the count is to include the staged key and a buffer key.

Note

If validating expired tokens is needed (for example when services are configured to use ServiceToken auth), the value of `allow_expired_window` option from the `[token]` config section should also be taken into account, so that the formula to calculate the max_active_keys is

max_active_keys = ((token_expiration + allow_expired_window) / rotation_frequency) + 2

This can be shown based on the previous example. We initially setup the key repository at 6:00 AM on Monday, and the initial state looks like:

```
$ ls -la /etc/keystone/fernet-keys/
drwx------ 2 keystone keystone 4096 .
drwxr-xr-x 3 keystone keystone 4096 ..
-rw------- 1 keystone keystone   44 0    (staged key)
-rw------- 1 keystone keystone   44 1    (primary key)
```

All tokens created after 6:00 AM are encrypted with key 1. At 12:00 PM we will rotate keys again, resulting in,

```
$ ls -la /etc/keystone/fernet-keys/
drwx------ 2 keystone keystone 4096 .
drwxr-xr-x 3 keystone keystone 4096 ..
-rw------- 1 keystone keystone   44 0    (staged key)
-rw------- 1 keystone keystone   44 1    (secondary key)
-rw------- 1 keystone keystone   44 2    (primary key)
```

We are still able to validate tokens created between 6:00 - 11:59 AM because the 1 key still exists as a secondary key. All tokens issued after 12:00 PM will be encrypted with key 2. At 6:00 PM we do our next rotation, resulting in:

```
$ ls -la /etc/keystone/fernet-keys/
drwx------ 2 keystone keystone 4096 .
drwxr-xr-x 3 keystone keystone 4096 ..
-rw------- 1 keystone keystone   44 0    (staged key)
-rw------- 1 keystone keystone   44 1    (secondary key)
-rw------- 1 keystone keystone   44 2    (secondary key)
-rw------- 1 keystone keystone   44 3    (primary key)
```

It is still possible to validate tokens issued from 6:00 AM - 5:59 PM because keys 1 and 2 exist as secondary keys. Every token issued until 11:59 PM will be encrypted with key 3, and at 12:00 AM we do our next rotation:

```
$ ls -la /etc/keystone/fernet-keys/
drwx------ 2 keystone keystone 4096 .
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x 3 keystone keystone 4096 ..
-rw------- 1 keystone keystone   44 0    (staged key)
-rw------- 1 keystone keystone   44 1    (secondary key)
-rw------- 1 keystone keystone   44 2    (secondary key)
-rw------- 1 keystone keystone   44 3    (secondary key)
-rw------- 1 keystone keystone   44 4    (primary key)
```

Just like before, we can still validate tokens issued from 6:00 AM the previous day until 5:59 AM today because keys 1 - 4 are present. At 6:00 AM, tokens issued from the previous day will start to expire and we do our next scheduled rotation:

```
$ ls -la /etc/keystone/fernet-keys/
drwx------ 2 keystone keystone 4096 .
drwxr-xr-x 3 keystone keystone 4096 ..
-rw------- 1 keystone keystone   44 0    (staged key)
-rw------- 1 keystone keystone   44 1    (secondary key)
-rw------- 1 keystone keystone   44 2    (secondary key)
-rw------- 1 keystone keystone   44 3    (secondary key)
-rw------- 1 keystone keystone   44 4    (secondary key)
-rw------- 1 keystone keystone   44 5    (primary key)
```

Tokens will naturally expire after 6:00 AM, but we will not be able to remove key 1 until the next rotation because it encrypted all tokens from 6:00 AM to 12:00 PM the day before. Once we do our next rotation, which is at 12:00 PM, the 1 key will be pruned from the repository:

```
$ ls -la /etc/keystone/fernet-keys/
drwx------ 2 keystone keystone 4096 .
drwxr-xr-x 3 keystone keystone 4096 ..
-rw------- 1 keystone keystone   44 0    (staged key)
-rw------- 1 keystone keystone   44 2    (secondary key)
-rw------- 1 keystone keystone   44 3    (secondary key)
-rw------- 1 keystone keystone   44 4    (secondary key)
-rw------- 1 keystone keystone   44 5    (secondary key)
-rw------- 1 keystone keystone   44 6    (primary key)
```

If keystone were to receive a token that was created between 6:00 AM and 12:00 PM the day before, encrypted with the 1 key, it would not be valid because it was already expired. This makes it possible for us to remove the 1 key from the repository without negative validation side-effects.

8.4.3 JWS key rotation

The JWS token provider issues tokens using asymmetric signing. This document attempts to describe how to manage key pairs in a deployment of keystone nodes that need to validate tokens issued by one another.

The inherent benefit of using asymmetric keys is that each keystone server generates its own key pair. The private key is used to sign tokens. Anyone with access to the public key has the ability to verify the token signature. This is a critical step in validating tokens across a cluster of keystone nodes.

It is necessary for operators to sync public keys across all keystone nodes in the deployment. Each keystone server will need a corresponding public key for every node. This only applies to public keys.

Private keys should never leave the server they are generated from.

Initial setup

Before a deployment of keystone servers can issue JWT tokens, each server must set `keystone.conf [token] provider = jws`. Additionally, each API server must have its own asymmetric key pair either generated manually or using `keystone-manage create_jws_keypair`. If youre generating the key pairs manually, they must be usable with the `ES256` JSON Web Algorithm (JWA). It is worth noting that the `keystone-manage create_jws_keypair` command line utility will create an appropriate key pair, but it will not automatically deploy it to the key repository locations defined in `keystone.conf [jwt_tokens]`. It is up to operators to move these files accordingly and resolve possible file name conflicts.

After generating a key pair, the public key from each API server must be shared with every other API server in the deployment. Ensure the private key used to sign JWS tokens is readable by the process running keystone and available in the `keystone.conf [jwt_tokens] jws_private_key_repository` location. Keystone will automatically use a key named `private.pem` to sign tokens and ignore all other keys in the repository. To validate tokens, keystone will iterate all available public keys in `keystone.conf [jwt_tokens] jws_public_key_repository`. At a minimum, this repository needs to have the corresponding public key to the `private.pem` key found in `keystone.conf [jwt_tokens] jws_private_key_repository`.

Continued operations

Depending on the security requirements for your deployment, you might need to rotate out an existing key pair. To do so without prematurely invalidating tokens, follow these steps:

1. Generate a new asymmetric key pair for a given keystone API server (see `keystone-manage create_jws_keypair` for more details)
2. Copy or sync the newly generated public key to the public key repositories of all other keystone API servers, the public key should be placed in `keystone.conf [jwt_tokens] jws_public_key_repository`
3. Copy the new private key to the private key repository on the API server youre performing the rotation on and make sure its named `private.pem`, at this point the server will start signing tokens with the new private key and all other keystone API servers will be able to validate those tokens since they already have a copy of the public key from step #2
4. At this point, you must wait until the last tokens signed with the old private key have expired before you can remove the old corresponding public keys from each keystone API server, note this should be a minimum of `keystone.conf [token] expiration`
5. Once youre confident all tokens signed with the old private key are expired, it is safe to remove the old corresponding public key from each API server in the deployment, which is important in case the original private key was compromised and prevents attackers from using it craft their own tokens

8.4.4 Token provider

OpenStack Identity supports customizable token providers. This is specified in the `[token]` section of the configuration file. The token provider controls the token construction, validation, and revocation operations.

You can register your own token provider by configuring the following property:

Note

More commonly, you can use this option to change the token provider to one of the ones built in. Alternatively, you can use it to configure your own token provider.

- `provider` - token provider driver. Defaults to `fernet`. Implemented by `keystone.token.providers.fernet.Provider`. This is the entry point for the token provider in the `keystone.token.provider` namespace.

Below is the detailed list of the token formats supported by keystone.:

Fernet
: `fernet` tokens do not need to be persisted at all, but require that you run `keystone-manage fernet_setup` (also see the `keystone-manage fernet_rotate` command).

Warning

Fernet tokens are bearer tokens. They must be protected from unnecessary disclosure to prevent unauthorized access.

JWS
: `jws` tokens do not need to be persisted at all, but require that you configure an asymmetric key pair to sign and validate tokens. The key pair can be generated using `keystone-manage create_jws_keypair` or it can be generated out-of-band manually so long as it is compatible with the JWT `ES256` Elliptic Curve Digital Signature Algorithm (ECDSA) using a P-256 curve and a SHA-256 hash algorithm.

Warning

JWS tokens are bearer tokens. They must be protected from unnecessary disclosure to prevent unauthorized access.

8.5 Default Roles

8.5.1 Primer

Like most OpenStack services, keystone protects its API using role-based access control (RBAC).

Users can access different APIs depending on the roles they have on a project, domain, or system, which we refer to as scope.

As of the Rocky release, keystone provides three roles called `admin`, `member`, and `reader` by default. Operators can grant these roles to any actor (e.g., group or user) on any scope (e.g., system, domain, or

project). If you need a refresher on authorization scopes and token types, please refer to the token guide. The following sections describe how each default role behaves with keystones API across different scopes. Additionally, other service developers can use this document as a guide for implementing similar patterns in their services.

Default roles and behaviors across scopes allow operators to delegate more functionality to their team, auditors, customers, and users without maintaining custom policies.

In addition to `admin`, `member`, and `reader` role, from 2023.2 (Bobcat) release keystone will provide the `service` and `manager` roles by default as well. Operators can use the `service` role for service to service API calls instead of using `admin` role for the same. The service role will be separate from `admin`, `member`, `reader` and will not implicate any of these roles. Operators can give the `manager` role to users to within a domain to enable self-service management of users, groups, projects and role assignments within their domain.

8.5.2 Roles Definitions

The default roles provided by keystone via `keystone-manage bootstrap` (except for the `service` role) are related through role implications. The `admin` role implies the `manager` role, the `manager` implies the `member` role, and the `member` role implies the `reader` role. These implications mean users with the `admin` role automatically have the `manager`, `member` and `reader` roles. Additionally, users with the `manager` role automatically have the `member` and `reader` roles. Users with the `member` role automatically have the `reader` role. Implying roles reduces role assignments and forms a natural hierarchy between the default roles. It also reduces the complexity of default policies by making check strings short. For example, a policy that requires `reader` can be expressed as:

```
"identity:list_foo": "role:reader"
```

Instead of:

```
"identity:list_foo": "role:admin or role:manager or role:member or role:reader
↪"
```

Reader

Warning

While its possible to use the `reader` role to perform audits, we highly recommend assessing the viability of using `reader` for auditing from the perspective of the compliance target youre pursuing.

The `reader` role is the least-privileged role within the role hierarchy described here. As such, OpenStack development teams, by default, do not advocate exposing sensitive information to users with the `reader` role, regardless of the scope. We have noted the need for a formal, read-only, role that is useful for inspecting all applicable resources within a particular scope, but it shouldnt be implemented as the lowest level of authorization. This work will come in a subsequent release where we support an elevated read-only role, that implies `reader`, but also exposes sensitive information, where applicable.

This will allow operators to grant third-party auditors a permissive role for viewing sensitive information, specifically for compliance targets that require it.

The `reader` role provides read-only access to resources within the system, a domain, or a project. Depending on the assignment scope, two users with the `reader` role can expect different API behaviors. For example, a user with the `reader` role on the system can list all projects within the deployment. A user with the `reader` role on a domain can only list projects within their domain.

By analyzing the scope of a role assignment, we increase the re-usability of the `reader` role and provide greater functionality without introducing more roles. For example, to accomplish this without analyzing assignment scope, you would need `system-reader`, `domain-reader`, and `project-reader` roles in addition to custom policies for each service.

Its imperative to note that `reader` is the least authoritative role in the hierarchy because assignments using `admin` or `member` ultimately include the `reader` role. We document this explicitly so that `reader` roles are not overloaded with read-only access to sensitive information. For example, a deployment pursuing a specific compliance target may want to leverage the `reader` role to perform the audit. If the audit requires the auditor to evaluate sensitive information, like license keys or administrative metadata, within a given scope, auditors shouldnt expect to perform these operations with the `reader` role. We justify this design decision because sensitive information should be explicitly protected, and not implicitly exposed.

The `reader` role should be implemented and used from the perspective of least-privilege, which may or may not fulfill your auditing use case.

Member

Within keystone, there isnt a distinct advantage to having the `member` role instead of the `reader` role. The `member` role is more applicable to other services. The `member` role works nicely for introducing granularity between `admin` and `reader` roles. Other services might write default policies that require the `member` role to create resources, but the `admin` role to delete them. For example, users with `reader` on a project could list instance, users with `member` on a project can list and create instances, and users with `admin` on a project can list, create, and delete instances. Service developers can use the `member` role to provide more flexibility between `admin` and `reader` on different scopes.

Manager

The `manager` role takes a special place in keystone. It sits between the `admin` and `member` role, allowing limited identity management while being clearly differentiated from the `admin` role both in terms of purpose and privileges. The `manager` role is meant to be assigned in a domain scope and enables users to manage identity assets in a whole domain including users, projects, groups and role assignments. This enables identity self-service management capabilities for users within a domain without the need to assign the most privileged `admin` role to them.

The keystone default policies include a special rule that specifies the list of roles a user with the `manager` role is be able to assign and revoke within the domain scope. This prevents such user from escalating their own privileges or those of others beyond `manager` and for this purpose the list excludes the `admin` role. The list can be adjusted by cloud administrators via policy definitions in case the role model differs. For example, if a new role is introduced for a specific cloud environment, the list can be adjusted to allow users with the `manager` role to also assign it.

Other services might write default policies to enable the `manager` role to have more privileged managing rights or cross-project privileges in a domain.

Admin

We reserve the `admin` role for the most privileged operations within a given scope. It is important to note that having `admin` on a project, domain, or the system carries separate authorization and are not transitive. For example, users with `admin` on the system should be able to manage every aspect of the deployment because theyre operators. Users with `admin` on a project shouldnt be able to manage things outside the project because it would violate the tenancy of their role assignment (this doesnt apply consistently since services are addressing this individually at their own pace).

Service

We reserve the `service` role for Service-to-service communication. The aim of a `service` role is to allow a service to communicate with another service and possibly be granted elevated privileges by the service receiving the request. Before the introduction of the `service` role, a service had to be granted the `admin` role in order to have elevated privileges, which gave a service powers way beyond what was necessary. With the `service` role in place, we can now allow all service-to-service APIs to default to the `service` role only. For example, a policy that requires `service` can be expressed as:

```
"identity:create_foo": "role:service"
```

There might be exception service-to-service APIs which project think are useful to be used by admin or non-admin user then they can take the exceptional decision to default them to user role and `service` role. For example, a policy that requires `service` and `admin` can be expressed as:

```
"identity:create_foo": "role:service or role:admin"
```

Note

Unlike the other default roles, the `service` role is *not* a member of a role hierarchy. It is a standalone role.

Note

As of the Train release, keystone applies the following personas consistently across its API.

8.5.3 System Personas

This section describes authorization personas typically used for operators and deployers. You can find all users with system role assignments using the following query:

```
$ openstack role assignment list --names --system all
+--------+-------------------------+-------------------------+---------+--------
↪+---------+-----------+
| Role   | User                    | Group                   | Project | Domain␣
↪| System | Inherited |
+--------+-------------------------+-------------------------+---------+--------
↪+---------+-----------+
```

(continues on next page)

(continued from previous page)

```
| admin  |                         | system-admins@Default  |          |       ␣
↪| all    | False     |
| admin  | admin@Default           |                        |          |       ␣
↪| all    | False     |
| admin  | operator@Default        |                        |          |       ␣
↪| all    | False     |
| reader |                         | system-support@Default |          |       ␣
↪| all    | False     |
| admin  | operator@Default        |                        |          |       ␣
↪| all    | False     |
| member | system-support@Default  |                        |          |       ␣
↪| all    | False     |
+--------+-------------------------+------------------------+----------+--------
↪+----------+-----------+
```

System Administrators

System administrators are allowed to manage every resource in keystone. System administrators are typically operators and cloud administrators. They can control resources that ultimately affect the behavior of the deployment. For example, they can add or remove services and endpoints in the catalog, create new domains, add federated mappings, and clean up stale resources, like a users application credentials or trusts.

You can find *system administrators* in your deployment with the following assignments:

```
$ openstack role assignment list --names --system all --role admin
+-------+-----------------+------------------------+---------+--------+-------
↪-+-----------+
| Role  | User            | Group                  | Project | Domain |␣
↪System | Inherited |
+-------+-----------------+------------------------+---------+--------+-------
↪-+-----------+
| admin |                 | system-admins@Default  |         |        | all   ␣
↪ | False     |
| admin | admin@Default   |                        |         |        | all   ␣
↪ | False     |
| admin | operator@Default |                       |         |        | all   ␣
↪ | False     |
+-------+-----------------+------------------------+---------+--------+-------
↪-+-----------+
```

System Members & System Readers

In keystone, *system members* and *system readers* are very similar and have the same authorization. Users with these roles on the system can view all resources within keystone. They can list role assignments, users, projects, and group memberships, among other resources.

The *system reader* persona is useful for members of a support team or auditors if the audit doesnt require access to sensitive information. You can find *system members* and *system readers* in your deployment with the following assignments:

```
$ openstack role assignment list --names --system all --role member --role␣
↪reader
+--------+-------------------------+----------------------+--------+--------
↪+--------+-----------+
| Role   | User                    | Group                | Project | Domain␣
↪| System | Inherited |
+--------+-------------------------+----------------------+--------+--------
↪+--------+-----------+
| reader |                         | system-support@Default |         |       ␣
↪| all    | False     |
| admin  | operator@Default        |                      |         |       ␣
↪| all    | False     |
| member | system-support@Default  |                      |         |       ␣
↪| all    | False     |
+--------+-------------------------+----------------------+--------+--------
↪+--------+-----------+
```

Warning

Filtering system role assignments is currently broken and is being tracked as a bug.

8.5.4 Domain Personas

This section describes authorization personas for people who manage their own domains, which contain projects, users, and groups. You can find all users with role assignments on a specific domain using the following query:

```
$ openstack role assignment list --names --domain foobar
+---------+-----------------+---------------------+---------+--------+-------
↪-+-----------+
| Role    | User            | Group               | Project | Domain |␣
↪System | Inherited |
+---------+-----------------+---------------------+---------+--------+-------
↪-+-----------+
| reader  | support@Default |                     |         | foobar |       ␣
↪ | False     |
| admin   | jsmith@Default  |                     |         | foobar |       ␣
↪ | False     |
| admin   |                 | foobar-admins@foobar |         | foobar |       ␣
```

(continues on next page)

(continued from previous page)

```
↪ | False     |
| manager | alice@foobar    |                      |         | foobar |      ␣
↪ | False     |
| member  | jdoe@foobar     |                      |         | foobar |      ␣
↪ | False     |
+---------+-----------------+----------------------+---------+--------+-------
↪-+-----------+
```

Domain Administrators

Domain administrators can manage most aspects of the domain or its contents. These users can create new projects and users within their domain. They can inspect the role assignments users have on projects within their domain.

Domain administrators arent allowed to access system-specific resources or resources outside their domain. Users that need control over project, group, and user creation are a great fit for *domain administrators*.

You can find *domain administrators* in your deployment with the following role assignment:

```
$ openstack role assignment list --names --domain foobar --role admin
+-------+----------------+----------------------+---------+--------+--------+-
↪----------+
| Role  | User           | Group                | Project | Domain | System |␣
↪Inherited |
+-------+----------------+----------------------+---------+--------+--------+-
↪----------+
| admin | jsmith@Default |                      |         | foobar |        |␣
↪False     |
| admin |                | foobar-admins@foobar |         | foobar |        |␣
↪False     |
+-------+----------------+----------------------+---------+--------+--------+-
↪----------+
```

Domain Managers

Domain managers can only manage specific resources related to identity management within their domain. This includes creating new users, projects and groups as well as updating and deleting them. They can also assign and revoke roles between those or in relation to the domain. Furthermore, they can inspect role assignments within the domain.

Domain managers cannot change any aspects of the domain itself. The role assignments they can apply within their domain is limited to a specific list of applicable roles and in the default configuration, this excludes the `admin` role to prevent privilege escalation.

You can find *domain managers* in your deployment with the following role assignment:

```
$ openstack role assignment list --names --domain foobar --role manager
+---------+-----------------+----------------------+---------+--------+-------
```

(continues on next page)

(continued from previous page)

```
↪-+-----------+
| Role    | User            | Group                | Project | Domain |␣
↪System | Inherited |
+---------+-----------------+----------------------+---------+--------+-------
↪-+-----------+
| manager | alice@foobar    |                      |         | foobar |      ␣
↪ | False     |
+---------+-----------------+----------------------+---------+--------+-------
↪-+-----------+
```

Domain Members & Domain Readers

Domain members and domain readers have the same relationship as system members and system readers. Theyre allowed to view resources and information about their domain. They arent allowed to access system-specific information or information about projects, groups, and users outside their domain.

The domain member and domain reader use-cases are great for support teams, monitoring the details of an account, or auditing resources within a domain assuming the audit doesnt validate sensitive information. You can find domain members and domain readers with the following role assignments:

```
$ openstack role assignment list --names --role member --domain foobar
+--------+-------------+-------+---------+--------+--------+-----------+
| Role   | User        | Group | Project | Domain | System | Inherited |
+--------+-------------+-------+---------+--------+--------+-----------+
| member | jdoe@foobar |       |         | foobar |        | False     |
+--------+-------------+-------+---------+--------+--------+-----------+
$ openstack role assignment list --names --role reader --domain foobar
+--------+-----------------+-------+---------+--------+--------+-----------+
| Role   | User            | Group | Project | Domain | System | Inherited |
+--------+-----------------+-------+---------+--------+--------+-----------+
| reader | support@Default |       |         | foobar |        | False     |
+--------+-----------------+-------+---------+--------+--------+-----------+
```

8.5.5 Project Personas

This section describes authorization personas for users operating within a project. These personas are commonly used by end users. You can find all users with role assignments on a specific project using the following query:

```
$ openstack role assignment list --names --project production
+--------+---------------+--------------------------+------------------+--
↪------+--------+-----------+
| Role   | User          | Group                    | Project          |␣
↪Domain | System | Inherited |
+--------+---------------+--------------------------+------------------+--
↪------+--------+-----------+
| admin  | jsmith@Default |                          | production@foobar | ␣
↪      |        | False     |
```

(continues on next page)

(continued from previous page)

```
| admin  |                | production-admins@foobar   | production@foobar | ␣
↪      |        | False     |
| member |                | foobar-operators@Default   | production@foobar | ␣
↪      |        | False     |
| reader | alice@Default  |                            | production@foobar | ␣
↪      |        | False     |
| reader |                | production-support@Default | production@foobar | ␣
↪      |        | False     |
+--------+----------------+----------------------------+-------------------+--
↪------+--------+-----------+
```

Project Administrators

Project administrators can only view and modify data within the project they have authorization on. Theyre able to view information about their projects and set tags on their projects. Theyre not allowed to view system or domain resources, as that would violate the tenancy of their role assignment. Since the majority of the resources in keystones API are system and domain-specific, *project administrators* dont have much authorization.

You can find *project administrators* in your deployment with the following role assignment:

```
$ openstack role assignment list --names --project production --role admin
+-------+----------------+--------------------------+-------------------+-----
↪---+--------+-----------+
| Role  | User           | Group                    | Project           |␣
↪Domain | System | Inherited |
+-------+----------------+--------------------------+-------------------+-----
↪---+--------+-----------+
| admin | jsmith@Default |                          | production@foobar |    ␣
↪    |        | False     |
| admin |                | production-admins@foobar | production@foobar |    ␣
↪    |        | False     |
+-------+----------------+--------------------------+-------------------+-----
↪---+--------+-----------+
```

Project Members & Project Readers

Project members and *project readers* can discover information about their projects. They can access important information like resource limits for their project, but theyre not allowed to view information outside their project or view system-specific information.

You can find *project members* and *project readers* in your deployment with the following role assignments:

```
$ openstack role assignment list --names --project production --role member
+--------+------+--------------------------+-------------------+--------+-----
↪---+-----------+
| Role   | User | Group                    | Project           | Domain |␣
↪System | Inherited |
```

(continues on next page)

(continued from previous page)

```
+--------+------+--------------------------+------------------+--------+-----
↪---+-----------+
| member |      | foobar-operators@Default | production@foobar |        |  ␣
↪   | False     |
+--------+------+--------------------------+------------------+--------+-----
↪---+-----------+
$ openstack role assignment list --names --project production --role reader
+--------+--------------+---------------------------+------------------+---
↪-----+--------+-----------+
| Role   | User         | Group                     | Project          |␣
↪Domain | System | Inherited |
+--------+--------------+---------------------------+------------------+---
↪-----+--------+-----------+
| reader | alice@Default |                          | production@foobar |  ␣
↪     |        | False     |
| reader |               | production-support@Default | production@foobar |  ␣
↪     |        | False     |
+--------+--------------+---------------------------+------------------+---
↪-----+--------+-----------+
```

8.5.6 Writing Policies

If the granularity provided above doesnt meet your specific use-case, you can still override policies and maintain them manually. You can read more about how to do that in oslo.policy usage documentation.

8.6 Advanced Keystone Features

Guides to lesser-known features of keystone.

8.6.1 Unified Limits

Warning

The unified limits API is currently labeled as experimental and can change in backwards incompatible ways. After we get feedback on the intricacies of the API and no longer expect to make API breaking changes, the API will be marked as stable.

As of the Queens release, keystone has the ability to store and relay information known as a limit. Limits can be used by services to enforce quota on resources across OpenStack. This section describes the basic concepts of limits, how the information can be consumed by services, and how operators can manage resource quota across OpenStack using limits.

What is a limit?

A limit is a threshold for resource management and helps control resource utilization. A process for managing limits allows for reallocation of resources to different users or projects as needs change. Some information needed to establish a limit may include:

- project_id
- domain_id
- API service type (e.g. compute, network, object-storage)
- a resource type (e.g. ram_mb, vcpus, security-groups)
- a default limit
- a project specific limit i.e resource limit
- user_id (optional)
- a region (optional depending on the service)

Note

The *default limit* of registered limit and the *resource limit* of project limit now are limited from *-1* to *2147483647* (integer). *-1* means no limit and *2147483647* is the max value for user to define limits. The length of unified limits *resource type* now is limited from *1* to *255* (string).

Since keystone is the source of truth for nearly everything in the above list, limits are a natural fit as a keystone resource. Two different limit resources exist in this design. The first is a registered limit and the second is a project limit.

Registered limits

A registered limit accomplishes two important things in order to enforce quota across multi-tenant, distributed systems. First, it establishes resource types and associates them to services. Second, it sets a default resource limit for all projects. The first part maps specific resource types to the services that provide them. For example, a registered limit can map *vcpus*, to the compute service. The second part sets a default of 20 *vcpus* per project. This provides all the information needed for basic quota enforcement for any resource provided by a service.

Domain limits

A domain limit is a limit associated to a specific domain and it acts as an override for a registered limit. Similar to registered limits, domain limits require a resource type and a service. Additionally, a registered limit must exist before you can create a domain-specific override. For example, lets assume a registered limit exists for *vcpus* provided by the compute service. It wouldnt be possible to create a domain limit for *cores* on the compute service. Domain limits can only override limits that have already been registered. In a general sense, registered limits are likely established when a new service or cloud is deployed. Domain limits are used continuously to manage the flow of resource allocation.

Domain limits may affect the limits of projects within the domain. This is particularly important to keep in mind when choosing an enforcement model, documented below.

Project limits

Project limits have the same properties as domain limits, but are specific to projects instead of domains. You must register a limit before creating a project-specific override. Just like with domain limits, the flow of resources between related projects may vary depending on the configured enforcement model. The support enforcement models below describe how limit validation and enforcement behave between related projects and domains.

Together, registered limits, domain limits, and project limits give deployments the ability to restrict resources across the deployment by default, while being flexible enough to freely marshal resources across projects.

Limits and usage

When we talk about a quota system, were really talking about two systems. A system for setting and maintaining limits, the theoretical maximum usage, and a system for enforcing that usage does not exceed limits. While they are coupled, they are distinct.

Up to this point, weve established that keystone is the system for maintaining limit information. Keystones responsibility is to ensure that any changes to limits are consistent with related limits currently stored in keystone.

Individual services maintain and enforce usage. Services check enforcement against the current limits at the time a user requests a resource. Usage reflects the actual resource allocation in units to a consumer.

Given the above, the following is a possible and legal flow:

- User Jane is in project Foo
- Project Foo has a default CPU limit of 20
- User Jane allocated 18 CPUs in project Foo
- Administrator Kelly sets project Foo CPU limit to 10
- User Jane can no longer allocate instance resources in project Foo, until she (or others in the project) have deleted at least 9 CPUs to get under the new limit

The following would be another permutation:

- User Jane is in project Foo
- Project Foo has a default CPU limit of 20
- User Jane allocated 20 CPUs in project Foo
- User Jane attempts to create another instance, which results in a failed resource request since the request would violate usage based on the current limit of CPUs
- User Jane requests more resources
- Administrator Kelly adjust the project limit for Foo to be 30 CPUs
- User Jane resends her request for an instance, which succeeds since the usage for project Foo is under the project limit of 30 CPUs

This behavior lets administrators set the policy of what the future should be when convenient, and prevent those projects from creating any more resources that would exceed the limits in question. Members of a

project can fix this for themselves by bringing down the project usage to where there is now headroom. If they dont, at some point the administrators can more aggressively delete things themselves.

Enforcement models

Project resources in keystone can be organized in hierarchical structures, where projects can be nested. As a result, resource limits and usage should respect that hierarchy if present. Its possible to think of different cases where limits or usage assume different characteristics, regardless of the project structure. For example, if a projects usage for a particular resource hasnt been met, should the projects underneath that project assume those limits? Should they not assume those limits? These opinionated models are referred to as enforcement models. This section is dedicated to describing different enforcement models that are implemented.

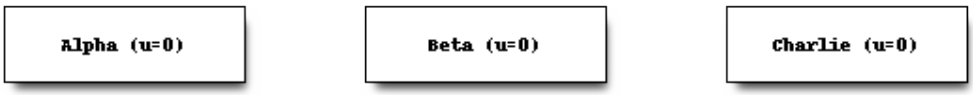
It is important to note that enforcement must be consistent across the entire deployment. Grouping certain characteristics into a model makes referring to behaviors consistent across services. Operators should be aware that switching between enforcement models may result in backwards incompatible changes. We recommend extremely careful planning and understanding of various enforcement models if youre planning on switching from one model to another in a deployment.

Keystone exposes a `GET /limits/model` endpoint that returns the enforcement model selected by the deployment. This allows limit information to be discoverable and preserves interoperability between OpenStack deployments with different enforcement models.

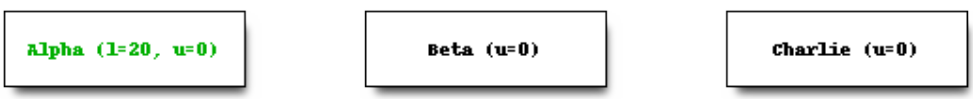
Flat

Flat enforcement ignores all aspects of a project hierarchy. Each project is considered a peer to all other projects. The limits associated to the parents, siblings, or children have no affect on a particular project. This model exercises the most isolation between projects because there are no assumptions between limits, regardless of the hierarchy. Validation of limits via the API will allow operations that might not be considered accepted in other models.

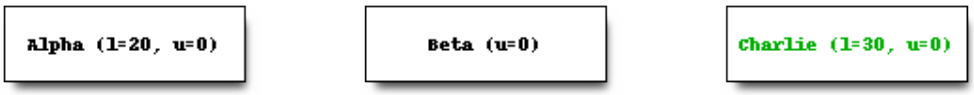
For example, assume project *Charlie* is a child of project *Beta*, which is a child of project *Alpha*. All projects assume a default limit of 10 cores via a registered limit. The labels in the diagrams below use shorthand notation for *limit* and *usage* as *l* and *u*, respectively:



Each project may use up to 10 cores because of the registered limit and none of the projects have an override. Using flat enforcement, youre allowed to `UPDATE LIMIT on Alpha to 20`:

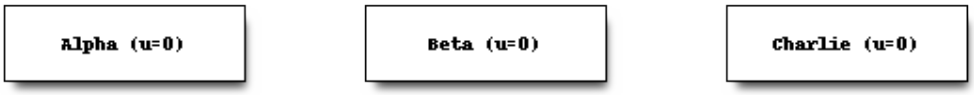


Youre also allowed to `UPDATE LIMIT on Charlie to 30`, even though *Charlie* is a sub-project of both *Beta* and *Alpha*.

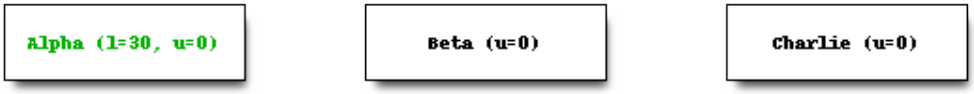


This is allowed with flat enforcement because the hierarchy is not taken into consideration during limit validation. Child projects may have a higher limit than a parent project.

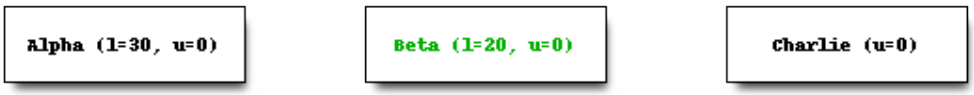
Conversely, you can simulate hierarchical enforcement by adjusting limits through the project tree manually. For example, lets still assume 10 is the default limit imposed by an existing registered limit:



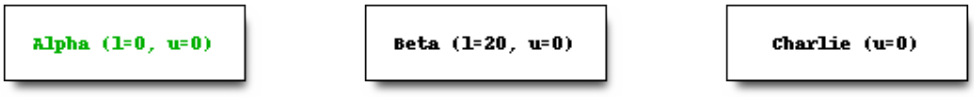
You may set a project-specific override to `UPDATE LIMIT on Alpha to 30`:



Next you can `UPDATE LIMIT on Beta to 20`:



Theoretically, the entire project tree consisting of *Alpha*, *Beta*, and *Charlie* is limited to 60 cores. If youd like to ensure only 30 cores are used within the entire hierarchy, you can `UPDATE LIMIT on Alpha to 0`:



You should use this model if you:

- Have project hierarchies greater than two levels

- Want extremely strict control of project usage and dont want resource usage to bleed across projects or domains

Advantages

- Allows you to model specific and strict limits
- Works with any project hierarchy or depth
- Usage is only calculated for the project in question

Disadvantages

- Resources arent allowed to flow gracefully between projects in a hierarchy
- Requires intervention and verification to move resources across projects
- Project limit validation isnt performed with respect to other projects or domains

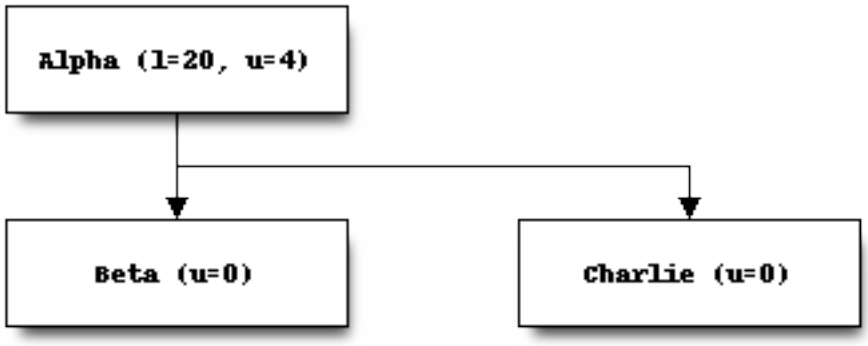
Strict Two Level

The `strict_two_level` enforcement model assumes the project hierarchy does not exceed two levels. The top layer can consist of projects or domains. For example, project *Alpha* can have a sub-project called *Beta* within this model. Project *Beta* cannot have a sub-project. The hierarchy is restrained to two layers. *Alpha* can also be a domain that contains project *Beta*, but *Beta* cannot have a sub-project. Regardless of the top layer consisting of projects or domains, the hierarchical depth is limited to two layers.

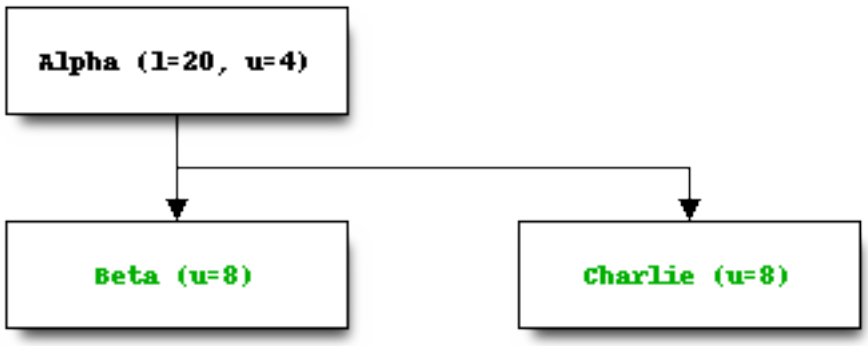
Resource utilization is allowed to flow between projects in the hierarchy, depending on the limits. This property allows for more flexibility than the `flat` enforcement model. The model is strict in that operators can set limits on parent projects or domains and the limits of the children may never exceed the parent.

For example, assume domain *Alpha* contains two projects, *Beta* and *Charlie*. Projects *Beta* and *Charlie* are siblings so the hierarchy maintains a depth of two. A system administrator sets the limit of a resource on *Alpha* to 20. Both projects *Beta* and *Charlie* can consume resources until the total usage of *Alpha*, *Beta*, and *Charlie* reach 20. At that point, no more resources should be allocated to the tree. System administrators can also reserve portions of domain *Alpha*s resource in sub-projects directly. Using the previous example, project *Beta* could have a limit of 12 resources, implicitly leaving 8 resources for *Charlie* to consume.

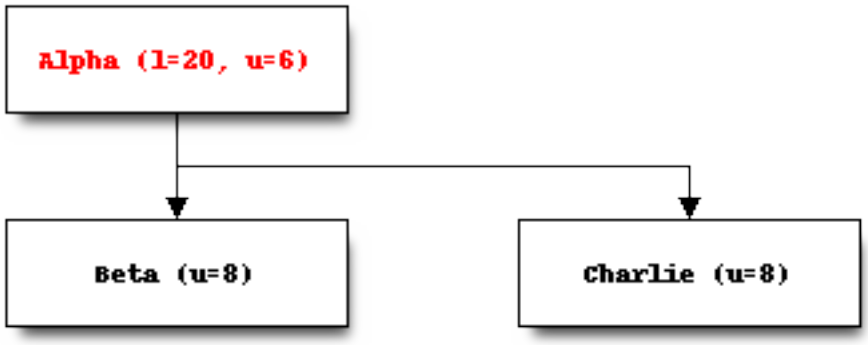
The following diagrams illustrate the behaviors described above, using projects named *Alpha*, *Beta*, *Charlie*, and *Delta*. Assume the resource in question is cores and the default registered limit for cores is 10. Also assume we have the following project hierarchy where *Alpha* has a limit of 20 cores and its usage is currently 4:



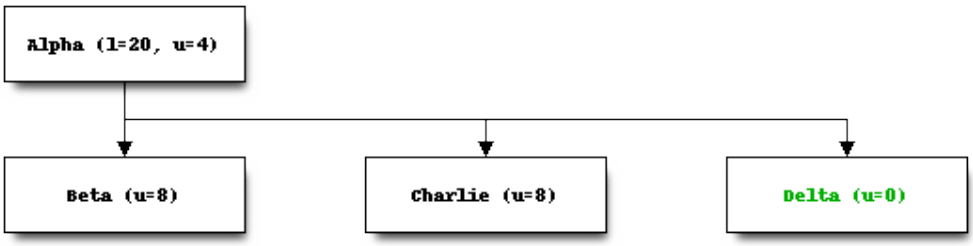
Technically, both *Beta* and *Charlie* can use up to 8 cores each:



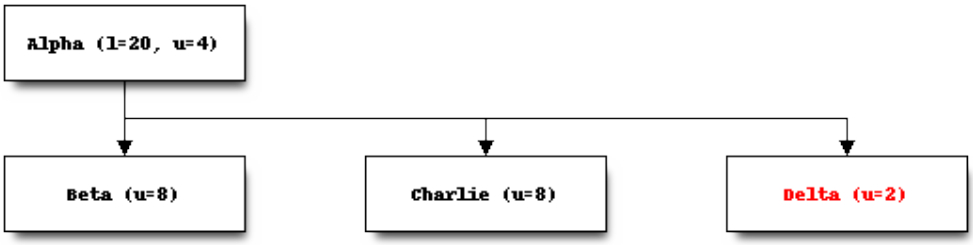
If *Alpha* attempts to claim two cores the usage check will fail because the service will fetch the hierarchy from keystone using `oslo.limit` and check the usage of each project in the hierarchy to see that the total usage of *Alpha*, *Beta*, and *Charlie* is equal to the limit of the tree, set by *Alpha.limit*:



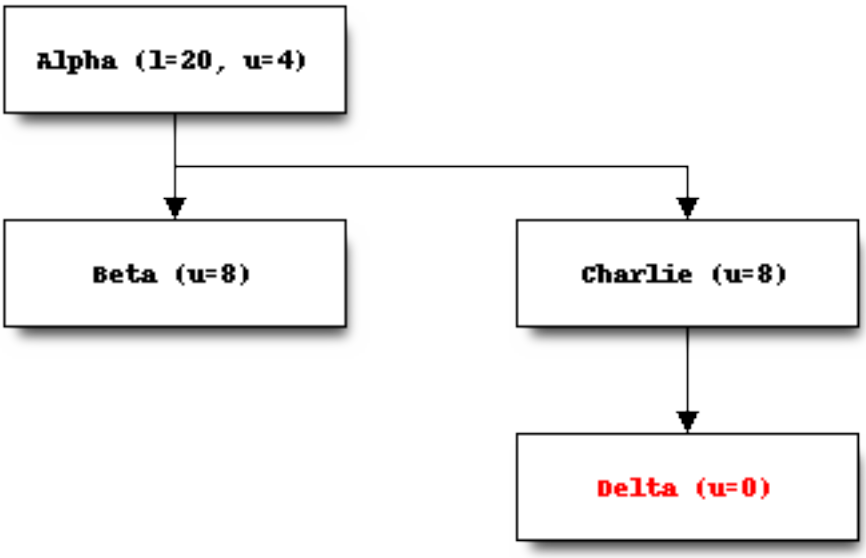
Despite the usage of the tree being equal to the limit, we can still add children to the tree:



Even though the project can be created, the current usage of cores across the tree prevents *Delta* from claiming any cores:



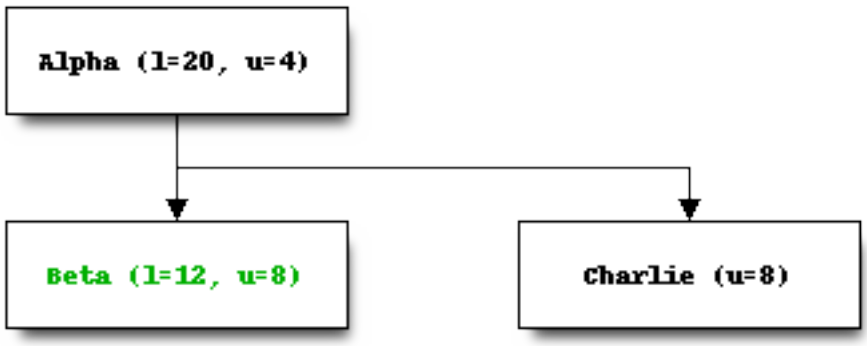
Creating a grandchild of project *Alpha* is forbidden because it violates the two-level hierarchical constraint:



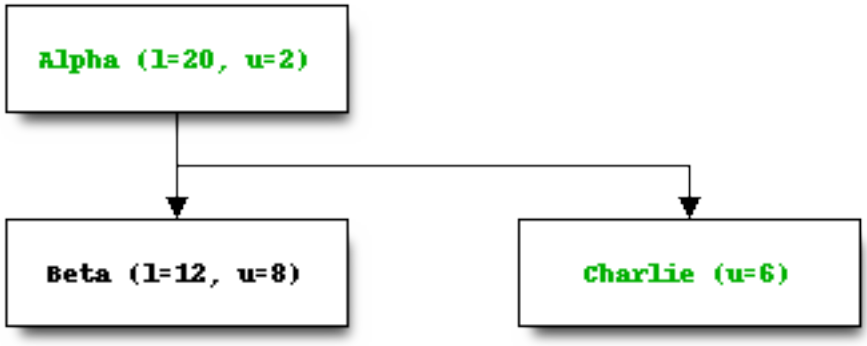
This is a fundamental constraint of this design because it provides a very clear escalation path. When a

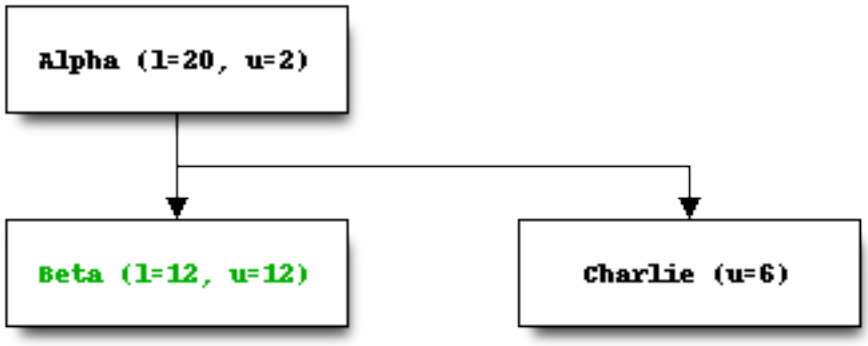
request fails because the tree limit has been exceeded, a user has all the information they need to provide meaningful context in a support ticket (e.g., their project ID and the parent project ID). An administrator should be able to reshuffle usage accordingly. Providing this information in tree structures with more than a depth of two is much harder, but may be implemented with a separate model.

Granting *Beta* the ability to claim more cores can be done by giving *Beta* a project-specific override for cores

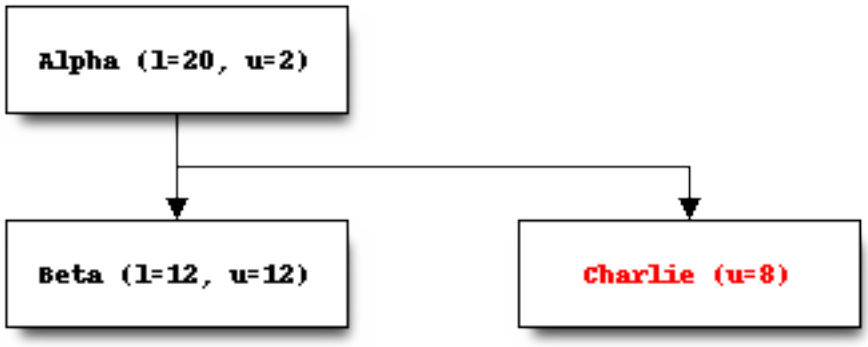


Note that regardless of this update, any subsequent requests to claim more cores in the tree will be rejected since the usage is equal to the limit of the *Alpha*. *Beta* can claim cores if they are released from *Alpha* or *Charlie*:

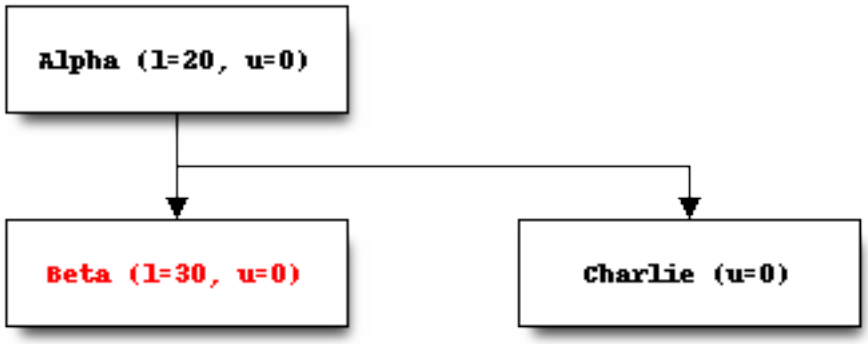


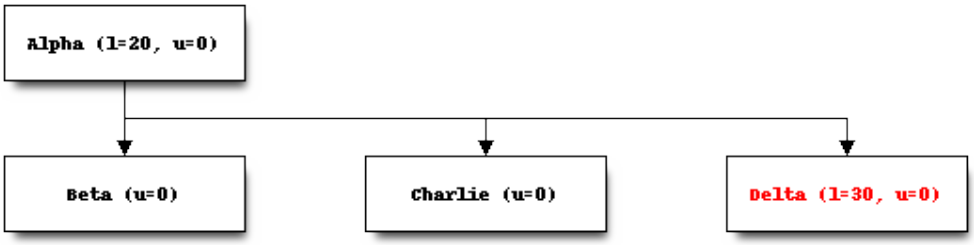


While *Charlie* is still under its default allocation of 10 cores, it wont be able to claim any more cores because the total usage of the tree is equal to the limit of *Alpha*, thus preventing *Charlie* from reclaiming the cores it had:

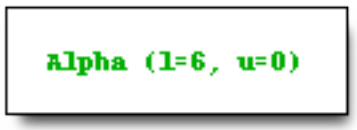


Creating or updating a project with a limit that exceeds the limit of *Alpha* is forbidden. Even though it is possible for the sum of all limits under *Alpha* to exceed the limit of *Alpha*, the total usage is capped at *Alpha.limit*. Allowing children to have explicit overrides greater than the limit of the parent would result in strange user experience and be misleading since the total usage of the tree would be capped at the limit of the parent:

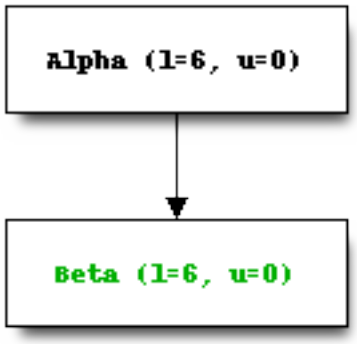




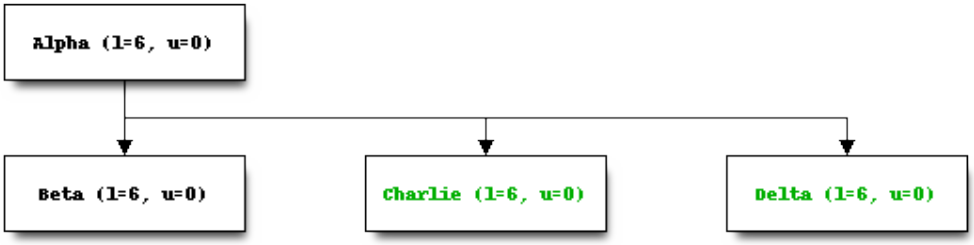
Finally, lets still assume the default registered limit for cores is 10, but were going to create project *Alpha* with a limit of 6 cores.



When we create project *Beta*, which is a child of project *Alpha*, the limit API ensures that project *Beta* doesnt assume the default of 10, despite the registered limit of 10 cores. Instead, the child assumes the parents limit since no single child limit should exceed the limit of the parent:



This behavior is consistent regardless of the number of children added under project *Alpha*.



Creating limit overrides while creating projects seems counter-productive given the whole purpose of a registered default, but it also seems unlikely to throttle a parent project by specifying its default to be less than a registered default. This behavior maintains consistency with the requirement that the sum of all child limits may exceed the parent limit, but the limit of any one child may not.

You should use this model if you:

- Want resources to flow between projects and domains within a hierarchy
- Dont have a project depth greater than two levels
- Are not concerned about usage calculation performance or dont have project trees that are wide

Advantages

- Allows resources to flow between projects and domains within a strict two-level hierarchy
- Limits are validated when they are created and updated

Disadvantages

- Project depth cannot exceed two levels
- Performance may suffer in wide and flat project hierarchies during usage calculation

8.6.2 Resource Options

A resource option is an attribute that can be optionally set on an entity in keystone. These options are used to control specific features or behaviors within keystone. This allows flexibility on a per-resource basis as opposed to settings a configuration file value that controls a behavior for all resources in a deployment.

This flexibility can be useful for deployments is setting different authentication requirements for users. For example, operators can use resource options to set the number of failed authentication attempts on a per-user basis as opposed to setting a global value that is applied to all users.

The purpose of this document is to formally document the supported resource options used in keystone, their intended behavior, and how to use them.

User Options

The following options are available on user resources. If left undefined, they are assumed to be false or disabled.

These can be set either in the initial user creation (`POST /v3/users`) or by updating an existing user to include new options (`PATCH /v3/users/{user_id}`):

```
{
    "user": {
        "options": {
            "ignore_lockout_failure_attempts": true
        }
    }
}
```

Note

User options of the `Boolean` type can be set to `True`, `False`, or `None`; if the option is set to `None`, it is removed from the users data structure.

ignore_user_inactivity

Type: `Boolean`

Opt into ignoring global inactivity lock settings defined in `keystone.conf [security_compliance]` on a per-user basis. Setting this option to `True` will make users not set as disabled even after the globally configured inactivity period is reached.

```
{
    "user": {
        "options": {
            "ignore_user_inactivity": true
        }
    }
}
```

Note

Setting this option for users which are already disabled will not make them automatically enabled. Such users must be enabled manually after setting this option to True for them.

See the security compliance documentation for more details.

ignore_change_password_upon_first_use

Type: `Boolean`

Control if a user should be forced to change their password immediately after they log into keystone for the first time. This can be useful for deployments that auto-generate passwords but want to ensure a user picks a new password when they start using the deployment.

```
{
    "user": {
        "options": {
            "ignore_change_password_upon_first_use": true
        }
    }
}
```

See the *security compliance documentation* for more details.

ignore_password_expiry

Type: `Boolean`

Opt into ignoring global password expiration settings defined in `keystone.conf [security_compliance]` on a per-user basis. Setting this option to `True` will allow users to continue using passwords that may be expired according to global configuration values.

```
{
    "user": {
        "options": {
            "ignore_password_expiry": true
        }
    }
}
```

See the *security compliance documentation* for more details.

ignore_lockout_failure_attempts

Type: `Boolean`

If `True`, opt into ignoring the number of times a user has authenticated and locking out the user as a result.

```
{
    "user": {
        "options": {
            "ignore_lockout_failure_attempts": true
        }
    }
}
```

See the *security compliance documentation* for more details.

lock_password

Type: `Boolean`

If set to `True`, this option disables the ability for users to change their password through self-service APIs.

```
{
    "user": {
        "options": {
            "lock_password": true
        }
    }
}
```

See the *security compliance documentation* for more details.

multi_factor_auth_enabled

Type: `Boolean`

Specify if a user has multi-factor authentication enabled on their account. This will result in different behavior at authentication time and the user may be presented with different authentication requirements based on multi-factor configuration.

```
{
    "user": {
        "options": {
            "multi_factor_auth_enabled": true
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

See *Multi-Factor Authentication* for further details.

multi_factor_auth_rules

Type: `List of Lists of Strings`

Define a list of strings that represent the methods required for a user to authenticate.

```
{
    "user": {
        "options": {
            "multi_factor_auth_rules": [
                ["password", "totp"],
                ["password", "u2f"]
            ]
        }
    }
}
```

See *Multi-Factor Authentication* for further details.

Role Options

The following options are available on role resources. If left undefined, they are assumed to be false or disabled.

immutable

Type: `Boolean`

Specify whether a role is immutable. An immutable role may not be deleted or modified except to remove the `immutable` option.

```
{
    "role": {
        "options": {
            "immutable": true
        }
    }
}
```

Project Options

The following options are available on project resources. If left undefined, they are assumed to be false or disabled.

immutable

Type: `Boolean`

Specify whether a project is immutable. An immutable project may not be deleted or modified except to remove the `immutable` option.

```
{
    "project": {
        "options": {
            "immutable": true
        }
    }
}
```

Domain Options

The following options are available on domain resources. If left undefined, they are assumed to be false or disabled.

immutable

Type: `Boolean`

Specify whether a domain is immutable. An immutable domain may not be deleted or modified except to remove the `immutable` option.

```
{
    "domain": {
        "options": {
            "immutable": true
        }
    }
}
```

8.6.3 Credential Encryption

As of the Newton release, keystone encrypts all credentials stored in the default `sql` backend. Credentials are encrypted with the same mechanism used to encrypt Fernet tokens, `fernet`. Keystone provides only one type of credential encryption but the encryption provider is pluggable in the event you wish to supply a custom implementation.

This document details how credential encryption works, how to migrate existing credentials in a deployment, and how to manage encryption keys for credentials.

Configuring credential encryption

The configuration for credential encryption is straightforward. There are only two configuration options needed:

```
[credential]
provider = fernet
key_repository = /etc/keystone/credential-keys/
```

`[credential] provider` defaults to the only option supplied by keystone, `fernet`. There is no reason to change this option unless you wish to provide a custom credential encryption implementation. The `[credential] key_repository` location is a requirement of using `fernet` but will default to the `/etc/keystone/credential-keys/` directory. Both `[credential] key_repository` and `[fernet_tokens] key_repository` define locations for keys used to encrypt things. One holds the keys to encrypt and decrypt credentials and the other holds keys to encrypt and decrypt tokens. It is imperative that these repositories are managed separately and they must not share keys. Meaning they cannot share the same directory path. The `[credential] key_repository` is only allowed to have three keys. This is not configurable and allows for credentials to be re-encrypted periodically with a new encryption key for the sake of security.

How credential encryption works

The implementation of this feature did not change any existing credential API contracts. All changes are transparent to the user unless youre inspecting the credential backend directly.

When creating a credential, keystone will encrypt the `blob` attribute before persisting it to the backend. Keystone will also store a hash of the key that was used to encrypt the information in that credential. Since Fernet is used to encrypt credentials, a key repository consists of multiple keys. Keeping track of which key was used to encrypt each credential is an important part of encryption key management. Why this is important is detailed later in the *Encryption key management* section.

When updating an existing credentials `blob` attribute, keystone will encrypt the new `blob` and update the key hash.

When listing or showing credentials, all `blob` attributes are decrypted in the response. Neither the cipher text, nor the hash of the key used to encrypt the `blob` are exposed through the API. Furthermore, the key is only used internally to keystone.

Encryption key management

Key management of `[credential] key_repository` is handled with three `keystone-manage` commands:

1. `keystone-manage credential_setup`
2. `keystone-manage credential_rotate`
3. `keystone-manage credential_migrate`

`keystone-manage credential_setup` will populate `[credential] key_repository` with new encryption keys. This must be done in order for proper credential encryption to work, with the exception of the null key. This step should only be done once.

`keystone-manage credential_rotate` will create and rotate a new encryption key in the `[credential] key_repository`. This will only be done if all credential key hashes match the hash of the current primary key. If any credential has been encrypted with an older key, or secondary key, the rotation will fail. Failing the rotation is necessary to prevent overrotation, which would leave some credentials indecipherable since the key used to encrypt it no longer exists. If this step fails, it is possible to forcibly re-key all credentials using the same primary key with `keystone-manage credential_migrate`.

`keystone-manage credential_migrate` will check the backend for credentials whose key hash doesnt match the hash of the current primary key. Any credentials with a key hash mismatching the current primary key will be re-encrypted with the current primary key. The new cipher text and key hash will be updated in the backend.

8.6.4 Health Check

Health check mechanism allows an operator to configure the endpoint URL that will provide information to a load balancer if the given API endpoint at the node should be available or not.

Its enabled by default in Keystone using the functions from *oslo.middleware*. And the URL is `/healthcheck`.

For more information and configuration options for the middleware see oslo.middleware.

8.6.5 Keystone Event Notifications

Keystone provides notifications about usage data so that 3rd party applications can use the data for billing, monitoring, or quota purposes. This document describes the current inclusions and exclusions for Keystone notifications.

Keystone currently supports two notification formats: a Basic Notification, and a Cloud Auditing Data Federation (CADF) Notification. The supported operations between the two types of notification formats are documented below.

Common Notification Structure

Notifications generated by Keystone are generated in JSON format. An external application can format them into ATOM format and publish them as a feed. Currently, all notifications are immediate, meaning they are generated when a specific event happens. Notifications all adhere to a specific top level format:

```
{
    "event_type": "identity.<resource_type>.<operation>",
    "message_id": "<message_id>",
    "payload": {},
    "priority": "INFO",
    "publisher_id": "identity.<hostname>",
    "timestamp": "<timestamp>"
}
```

Where `<resource_type>` is a Keystone resource, such as user or project, and `<operation>` is a Keystone operation, such as created, deleted.

The key differences between the two notification formats (Basic and CADF), lie within the `payload` portion of the notification.

The `priority` of the notification being sent is not configurable through the Keystone configuration file. This value is defaulted to INFO for all notifications sent in Keystones case.

Auditing with CADF

Keystone uses the PyCADF library to emit CADF notifications, these events adhere to the DMTF CADF specification. This standard provides auditing capabilities for compliance with security, operational, and business processes and supports normalized and categorized event data for federation and aggregation.

CADF notifications include additional context data around the `resource`, the `action` and the `initiator`.

CADF notifications may be emitted by changing the `notification_format` to `cadf` in the configuration file.

The `payload` portion of a CADF Notification is a CADF `event`, which is represented as a JSON dictionary. For example:

```
{
    "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
    "initiator": {
        "typeURI": "service/security/account/user",
        "host": {
            "agent": "curl/7.22.0(x86_64-pc-linux-gnu)",
            "address": "127.0.0.1"
        },
        "id": "<initiator_id>"
    },
    "target": {
        "typeURI": "<target_uri>",
        "id": "openstack:1c2fc591-facb-4479-a327-520dade1ea15"
    },
```

(continues on next page)

(continued from previous page)

```
    "observer": {
        "typeURI": "service/security",
        "id": "openstack:3d4a50a9-2b59-438b-bf19-c231f9c7625a"
    },
    "eventType": "activity",
    "eventTime": "2014-02-14T01:20:47.932842+00:00",
    "action": "<action>",
    "outcome": "success",
    "id": "openstack:f5352d7b-bee6-4c22-8213-450e7b646e9f",
}
```

Where the following are defined:

- <initiator_id>: ID of the user that performed the operation
- <target_uri>: CADF specific target URI, (i.e.: data/security/project)
- <action>: The action being performed, typically: <operation>. <resource_type>

Note

The eventType property of the CADF payload is different from the event_type property of a notifications. The former (eventType) is a CADF keyword which designates the type of event that is being measured, this can be: *activity*, *monitor* or *control*. Whereas the latter (event_type) is described in previous sections as: *identity.<resource_type>.<operation>*

Additionally there may be extra keys present depending on the operation being performed, these will be discussed below.

Reason

There is a specific reason object that will be present for the following PCI-DSS related events:

PCI-DSS Section	reason-Code	reasonType
8.1.6 Limit repeated access attempts by locking out the user after more than X failed attempts.	401	Maximum number of <number> login attempts exceeded.
8.2.3 Passwords must meet the established criteria.	400	Password does not meet expected requirements: <regex_description>
8.2.4 Password must be changed every X days.	401	Password for <user> expired and must be changed
8.2.5 Do not let users reuse the last X passwords.	400	Changed password cannot be identical to the last <number> passwords.
Other - Prevent passwords from being changed for a minimum of X days.	401	Cannot change password before minimum age <number> days is met

The reason object will contain the following keys:

- `reasonType`: Description of the PCI-DSS event
- `reasonCode`: HTTP response code for the event

For more information, see *Security compliance and PCI-DSS* for configuring PCI-DSS in keystone.

Supported Events

The following table displays the compatibility between resource types and operations.

Resource Type	Supported Operations	typeURI
group	create,update,delete	data/security/group
project	create,update,delete	data/security/project
role	create,update,delete	data/security/role
domain	create,update,delete	data/security/domain
user	create,update,delete	data/security/account/user
trust	create,delete	data/security/trust
region	create,update,delete	data/security/region
endpoint	create,update,delete	data/security/endpoint
service	create,update,delete	data/security/service
policy	create,update,delete	data/security/policy
role assignment	add,remove	data/security/account/user
None	authenticate	data/security/account/user

Example Notification - Project Create

The following is an example of a notification that is sent when a project is created. This example can be applied for any `create`, `update` or `delete` event that is seen in the table above. The <`action`> and `typeURI` fields will be change.

The difference to note is the inclusion of the `resource_info` field which contains the <`resource_id`> that is undergoing the operation. Thus creating a common element between the CADF and Basic notification formats.

```
{
    "event_type": "identity.project.created",
    "message_id": "0156ee79-b35f-4cef-ac37-d4a85f231c69",
    "payload": {
        "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
        "initiator": {
            "typeURI": "service/security/account/user",
            "host": {
                "agent": "curl/7.22.0(x86_64-pc-linux-gnu)",
                "address": "127.0.0.1"
            },
            "id": "c9f76d3c31e142af9291de2935bde98a"
        },
        "target": {
            "typeURI": "data/security/project",
```

(continues on next page)

(continued from previous page)


```
            "id": "openstack:1c2fc591-facb-4479-a327-520dade1ea15"
        },
        "observer": {
            "typeURI": "service/security",
            "id": "openstack:3d4a50a9-2b59-438b-bf19-c231f9c7625a"
        },
        "eventType": "activity",
        "eventTime": "2014-02-14T01:20:47.932842+00:00",
        "action": "created.project",
        "outcome": "success",
        "id": "openstack:f5352d7b-bee6-4c22-8213-450e7b646e9f",
        "resource_info": "671da331c47d4e29bb6ea1d270154ec3"
    },
    "priority": "INFO",
    "publisher_id": "identity.host1234",
    "timestamp": "2013-08-29 19:03:45.960280"
}
```


Example Notification - Authentication

The following is an example of a notification that is sent when a user authenticates with Keystone.

Note that this notification will be emitted if a user successfully authenticates, and when a user fails to authenticate.


```
{
    "event_type": "identity.authenticate",
    "message_id": "1371a590-d5fd-448f-b3bb-a14dead6f4cb",
    "payload": {
        "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
        "initiator": {
            "typeURI": "service/security/account/user",
            "host": {
                "agent": "curl/7.22.0(x86_64-pc-linux-gnu)",
                "address": "127.0.0.1"
            },
            "id": "c9f76d3c31e142af9291de2935bde98a"
        },
        "target": {
            "typeURI": "service/security/account/user",
            "id": "openstack:1c2fc591-facb-4479-a327-520dade1ea15"
        },
        "observer": {
            "typeURI": "service/security",
            "id": "openstack:3d4a50a9-2b59-438b-bf19-c231f9c7625a"
        },
        "eventType": "activity",
        "eventTime": "2014-02-14T01:20:47.932842+00:00",
        "action": "authenticate",
```


(continues on next page)

(continued from previous page)

```
        "outcome": "success",
        "id": "openstack:f5352d7b-bee6-4c22-8213-450e7b646e9f"
    },
    "priority": "INFO",
    "publisher_id": "identity.host1234",
    "timestamp": "2014-02-14T01:20:47.932842"
}
```

Example Notification - Federated Authentication

The following is an example of a notification that is sent when a user authenticates with Keystone via Federation.

This example is similar to the one seen above, however the `initiator` portion of the `payload` contains a new `credential` section.

```
{
    "event_type": "identity.authenticate",
    "message_id": "1371a590-d5fd-448f-b3bb-a14dead6f4cb",
    "payload": {
        "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
        "initiator": {
            "credential": {
                "type": "http://docs.oasis-open.org/security/saml/v2.0",
                "token": "671da331c47d4e29bb6ea1d270154ec3",
                "identity_provider": "ACME",
                "user": "c9f76d3c31e142af9291de2935bde98a",
                "groups": [
                    "developers"
                ]
            },
            "typeURI": "service/security/account/user",
            "host": {
                "agent": "curl/7.22.0(x86_64-pc-linux-gnu)",
                "address": "127.0.0.1"
            },
            "id": "c9f76d3c31e142af9291de2935bde98a"
        },
        "target": {
            "typeURI": "service/security/account/user",
            "id": "openstack:1c2fc591-facb-4479-a327-520dade1ea15"
        },
        "observer": {
            "typeURI": "service/security",
            "id": "openstack:3d4a50a9-2b59-438b-bf19-c231f9c7625a"
        },
        "eventType": "activity",
        "eventTime": "2014-02-14T01:20:47.932842+00:00",
        "action": "authenticate",
```

(continues on next page)

(continued from previous page)

```
        "outcome": "success",
        "id": "openstack:f5352d7b-bee6-4c22-8213-450e7b646e9f"
    },
    "priority": "INFO",
    "publisher_id": "identity.host1234",
    "timestamp": "2014-02-14T01:20:47.932842"
}
```

Example Notification - Role Assignment

The following is an example of a notification that is sent when a role is granted or revoked to a project or domain, for a user or group.

It is important to note that this type of notification has many new keys that convey the necessary information. Expect the following in the `payload`: `role`, `inherited_to_project`, `project` or `domain`, `user` or `group`. With the exception of `inherited_to_project`, each will represent the unique identifier of the resource type.

```
{
    "event_type": "identity.role_assignment.created",
    "message_id": "a5901371-d5fd-b3bb-448f-a14dead6f4cb",
    "payload": {
        "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
        "initiator": {
            "typeURI": "service/security/account/user",
            "host": {
                "agent": "curl/7.22.0(x86_64-pc-linux-gnu)",
                "address": "127.0.0.1"
            },
            "id": "c9f76d3c31e142af9291de2935bde98a"
        },
        "target": {
            "typeURI": "service/security/account/user",
            "id": "openstack:1c2fc591-facb-4479-a327-520dade1ea15"
        },
        "observer": {
            "typeURI": "service/security",
            "id": "openstack:3d4a50a9-2b59-438b-bf19-c231f9c7625a"
        },
        "eventType": "activity",
        "eventTime": "2014-08-20T01:20:47.932842+00:00",
        "role": "0e6b990380154a2599ce6b6e91548a68",
        "project": "24bdcff1aab8474895dbaac509793de1",
        "inherited_to_projects": false,
        "group": "c1e22dc67cbd469ea0e33bf428fe597a",
        "action": "created.role_assignment",
        "outcome": "success",
        "id": "openstack:f5352d7b-bee6-4c22-8213-450e7b646e9f"
    },
```

(continues on next page)

(continued from previous page)

```
    "priority": "INFO",
    "publisher_id": "identity.host1234",
    "timestamp": "2014-08-20T01:20:47.932842"
}
```

Example Notification - Expired Password

The following is an example of a notification that is sent when a user attempts to authenticate but their password has expired.

In this example, the `payload` contains a `reason` portion which contains both a `reasonCode` and `reasonType`.

```
{
    "priority": "INFO",
    "_unique_id": "222441bdc958423d8af6f28f9c558614",
    "event_type": "identity.authenticate",
    "timestamp": "2016-11-11 18:31:11.290821",
    "publisher_id": "identity.host1234",
    "payload": {
        "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
        "initiator": {
            "typeURI": "service/security/account/user",
            "host": {
                "address": "127.0.0.1"
            },
            "id": "73a19db6-e26b-5313-a6df-58d297fa652e"
        },
        "target": {
            "typeURI": "service/security/account/user",
            "id": "c23e6cb7-abe0-5e42-b7f7-4c4104ea77b0"
        },
        "observer": {
            "typeURI": "service/security",
            "id": "9bdddeda6a0b451e9e0439646e532afd"
        },
        "eventType": "activity",
        "eventTime": "2016-11-11T18:31:11.156356+0000",
        "reason": {
            "reasonCode": 401,
            "reasonType": "The password is expired and needs to be reset for␣
↪user: ed1ab0b40f284fb48fea9e25d0d157fc"
        },
        "action": "authenticate",
        "outcome": "failure",
        "id": "78cd795f-5850-532f-9ab1-5adb04e30c0f"
    },
    "message_id": "9a97e9d0-fef1-4852-8e82-bb693358bc46"
}
```

Basic Notifications

All basic notifications contain a limited amount of information, specifically, just the resource type, operation, and resource id.

The `payload` portion of a Basic Notification is a single key-value pair.

```
{
    "resource_info": <resource_id>
}
```

Where <resource_id> is the unique identifier assigned to the `resource_type` that is undergoing the <operation>.

Supported Events

The following table displays the compatibility between resource types and operations.

Resource Type	Supported Operations
group	create,update,delete
project	create,update,delete
role	create,update,delete
domain	create,update,delete
user	create,update,delete
trust	create,delete
region	create,update,delete
endpoint	create,update,delete
service	create,update,delete
policy	create,update,delete

Note, `trusts` are an immutable resource, they do not support `update` operations.

Example Notification

This is an example of a notification sent for a newly created user:

```
{
    "event_type": "identity.user.created",
    "message_id": "0156ee79-b35f-4cef-ac37-d4a85f231c69",
    "payload": {
        "resource_info": "671da331c47d4e29bb6ea1d270154ec3"
    },
    "priority": "INFO",
    "publisher_id": "identity.host1234",
    "timestamp": "2013-08-29 19:03:45.960280"
}
```

If the operation fails, the notification wont be sent, and no special error notification will be sent. Information about the error is handled through normal exception paths.

Recommendations for consumers

One of the most important notifications that Keystone emits is for project deletions (`event_type = identity.project.deleted`). This event should indicate to the rest of OpenStack that all resources (such as virtual machines) associated with the project should be deleted.

Projects can also have update events (`event_type = identity.project.updated`), wherein the project has been disabled. Keystone ensures this has an immediate impact on the accessibility of the projects resources by revoking tokens with authorization on the project, but should **not** have a direct impact on the projects resources (in other words, virtual machines should **not** be deleted).

Opting out of certain notifications

There are many notifications that Keystone emits and some deployers may only care about certain events. In Keystone there is a way to opt-out of certain notifications. In `/etc/keystone/keystone.conf` you can set `opt_out` to the event you wish to opt-out of. It is possible to opt-out of multiple events.

Example:

```
[DEFAULT]
notification_opt_out = identity.user.created
notification_opt_out = identity.role_assignment.created
notification_opt_out = identity.authenticate.pending
```

This will opt-out notifications for user creation, role assignment creation and successful authentications. For a list of event types that can be used, refer to: Telemetry Measurements.

By default, messages for the following authentication events are suppressed since they are too noisy: `identity.authenticate.success`, `identity.authenticate.pending` and `identity.authenticate.failed`.

8.7 Authentication Mechanisms

Keystone supports various methods of authentication beyond the standard local user and password method.

8.7.1 Multi-Factor Authentication

Configuring MFA

MFA is configured on a per user basis via the user options *multi_factor_auth_rules* and *multi_factor_auth_enabled*. Until these are set the user can authenticate with any one of the enabled auth methods.

MFA rules

The MFA rules allow an admin to force a user to use specific forms of authentication or combinations of forms of authentication to get a token.

The rules are specified as follows via the user option *multi_factor_auth_rules*:

```
[["password", "totp"], ["password", "custom-auth-method"]]
```

They are a list of lists. The elements of the sub-lists must be strings and are intended to mirror the required authentication method names (e.g. `password`, `totp`, etc) as defined in the `keystone.conf` file in the `[auth] methods` option. Each list of methods specifies a rule.

If the auth methods provided by a user match (or exceed) the auth methods in the list, that rule is used. The first rule found (rules will not be processed in a specific order) that matches will be used. If a user has the ruleset defined as `[["password", "totp"]]` the user must provide both password and totp auth methods (and both methods must succeed) to receive a token. However, if a user has a ruleset defined as `[["password"], ["password", "totp"]]` the user may use the `password` method on its own but would be required to use both `password` and `totp` if `totp` is specified at all.

Any auth methods that are not defined in `keystone.conf` in the `[auth] methods` option are ignored when the rules are processed. Empty rules are not allowed. If a rule is empty due to no-valid auth methods existing within it, the rule is discarded at authentication time. If there are no rules or no valid rules for the user, authentication occurs in the default manner: any single configured auth method is sufficient to receive a token.

Note

The `token` auth method typically should not be specified in any MFA Rules. The `token` auth method will include all previous auth methods for the original auth request and will match the appropriate ruleset. This is intentional, as the `token` method is used for rescoping/changing active projects.

Enabling MFA

Before the MFA rules take effect on a user, MFA has to be enabled for that user via the user option *multi_factor_auth_enabled*. By default this is unset, and the rules will not take effect until configured.

In the case a user should be exempt from MFA Rules, regardless if they are set, the User-Option may be set to `False`.

Using MFA

See *Multi-Factor Authentication* in the user guide for some examples.

Supported multi-factor authentication methods

TOTP is the only suggested second factor along with password for now, but there are plans to include more in future.

TOTP

This is a simple 6 digit passcode generated by both the server and client from a known shared secret.

This used in a multi-step fashion is the most common 2-factor method used these days.

See: *Time-based One-time Password (TOTP)*

8.7.2 Time-based One-time Password (TOTP)

Configuring TOTP

TOTP is not enabled in Keystone by default. To enable it add the `totp` authentication method to the `[auth]` section in `keystone.conf`:

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

For a user to have access to TOTP, he must have configured TOTP credentials in Keystone and a TOTP device (i.e. Google Authenticator).

TOTP uses a base32 encoded string for the secret. The secret must be at least 128 bits (16 bytes). The following python code can be used to generate a TOTP secret:

```
import base64
message = b'1234567890123456'
print(base64.b32encode(message).rstrip(b'='))
```

Example output:

```
GEZDGNBVGY3TQOJQGEZDGNBVGY
```

This generated secret can then be used to add new totp credentials to a specific user.

Create a TOTP credential

Create `totp` credentials for user:

```
USER_ID=b7793000f8d84c79af4e215e9da78654
SECRET=GEZDGNBVGY3TQOJQGEZDGNBVGY

curl -i \
  -H "Content-Type: application/json" \
  -d '
{
```

(continues on next page)

(continued from previous page)

```
    "credential": {
        "blob": "'$SECRET'",
        "type": "totp",
        "user_id": "'$USER_ID'"
    }
}' \
  http://localhost:5000/v3/credentials ; echo
```

Google Authenticator

On a device install Google Authenticator and inside the app click on Set up account and then click on Enter provided key. In the input fields enter account name and secret. Optionally a QR code can be generated programmatically to avoid having to type the information.

QR code

Create TOTP QR code for device:

```
import qrcode

secret='GEZDGNBVGY3TQOJQGEZDGNBVGY'
uri = 'otpauth://totp/{name}?secret={secret}&issuer={issuer}'.format(
    name='name',
    secret=secret,
    issuer='Keystone')

img = qrcode.make(uri)
img.save('totp.png')
```

In Google Authenticator app click on Set up account and then click on Scan a barcode, and then scan the totp.png image. This should create a new TOTP entry in the application.

Authenticate with TOTP

Google Authenticator will generate a 6 digit PIN (passcode) every few seconds. Use the passcode and your user ID to authenticate using the `totp` method.

Tokens

Get a token with default scope (may be unscoped) using totp:

```
USER_ID=b7793000f8d84c79af4e215e9da78654
PASSCODE=012345

curl -i \
  -H "Content-Type: application/json" \
```

(continues on next page)

(continued from previous page)

```
  -d '
{ "auth": {
        "identity": {
            "methods": [
                "totp"
            ],
            "totp": {
                "user": {
                    "id": "'$USER_ID'",
                    "passcode": "'$PASSCODE'"
                }
            }
        }
    }
}' \
  http://localhost:5000/v3/auth/tokens ; echo
```

8.7.3 Federated Identity

Introduction to Keystone Federation

What is keystone federation?

Identity federation is the ability to share identity information across multiple identity management systems. In keystone, this is implemented as an authentication method that allows users to authenticate directly with another identity source and then provides keystone with a set of user attributes. This is useful if your organization already has a primary identity source since it means users dont need a separate set of credentials for the cloud. It is also useful for connecting multiple clouds together, as we can use a keystone in another cloud as an identity source. Using *LDAP as an identity backend* is another way for keystone to obtain identity information from an external source, but it requires keystone to handle passwords directly rather than offloading authentication to the external source.

Keystone supports two configuration models for federated identity. The most common configuration is with *keystone as a Service Provider (SP)*, using an external Identity Provider, such as a Keycloak or Google, as the identity source and authentication method. The second type of configuration is *Keystone to Keystone*, where two keystones are linked with one acting as the identity source.

This document discusses identity federation involving a secondary identity management that acts as the source of truth concerning the users it contains, specifically covering the SAML2.0 and OpenID Connect protocols, although keystone can work with other protocols. A similar concept is *external authentication* whereby keystone is still the source of truth about its users but authentication is handled externally. Yet another closely related topic is *tokenless authentication* which uses some of the same constructs as described here but allows services to validate users without using keystone tokens.

Glossary

Service Provider (SP)
A Service Provider is the service providing the resource an end-user is requesting. In our case, this is keystone, which provides keystone tokens that we use on other OpenStack services. We do NOT call the other OpenStack services service providers. The specific service we care about in this context is the token service, so that is our Service Provider.

Identity Provider (IdP)
An Identity Provider is the service that accepts credentials, validates them, and generates a yay/nay response. It returns this response along with some other attributes about the user, such as their username, their display name, and whatever other details it stores and youve configured your Service Provider to accept.

Entity ID or Remote ID
An Entity ID or a Remote ID are both names for a unique identifier string for either a Service Provider or an Identity Provider. It usually takes the form of a URN, but the URN does not need to be a resolvable URL. Remote IDs are globally unique. Two Identity Providers cannot be associated with the same remote ID. Keystone uses the remote ID retrieved from the HTTPD environment variables to match the incoming request with a trusted Identity Provider and render the appropriate authorization mapping.

SAML2.0
SAML2.0 is an XML-based federation protocol. It is commonly used in internal-facing organizations, such as a university or business in which IT services are provided to members of the organization.

OpenID Connect (OpenIDC)
OpenID Connect is a JSON-based federation protocol built on OAuth 2.0. Its used more often by public-facing services like Google.

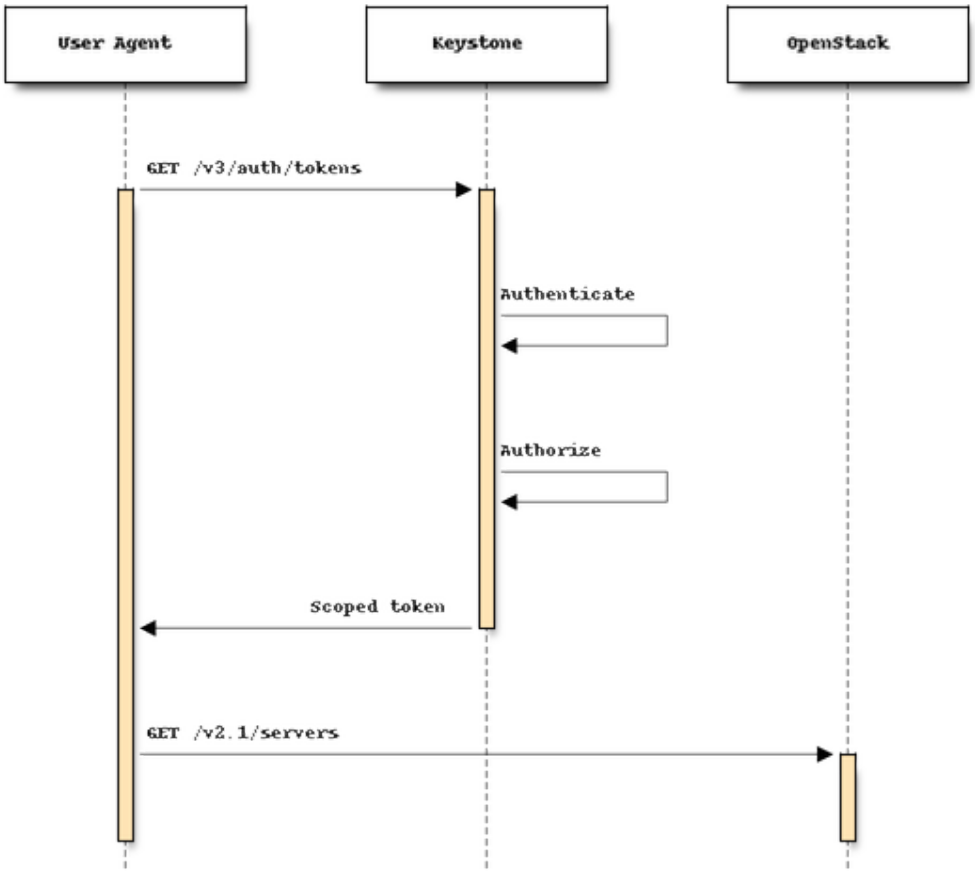
Assertion
An assertion is a formatted statement from the Identity Provider that asserts that a user is authenticated and provides some attributes about the user. The Identity Provider always signs the assertion and typically encrypts it as well.

Single Sign-On (SSO)
Single Sign-On is a mechanism related to identity federation whereby a user may log in to their identity management system and be granted a token or ticket that allows them access to multiple Service Providers.

Authentication Flows

Understanding the flow of information as a user moves through the authentication process is key to being able to debug later on.

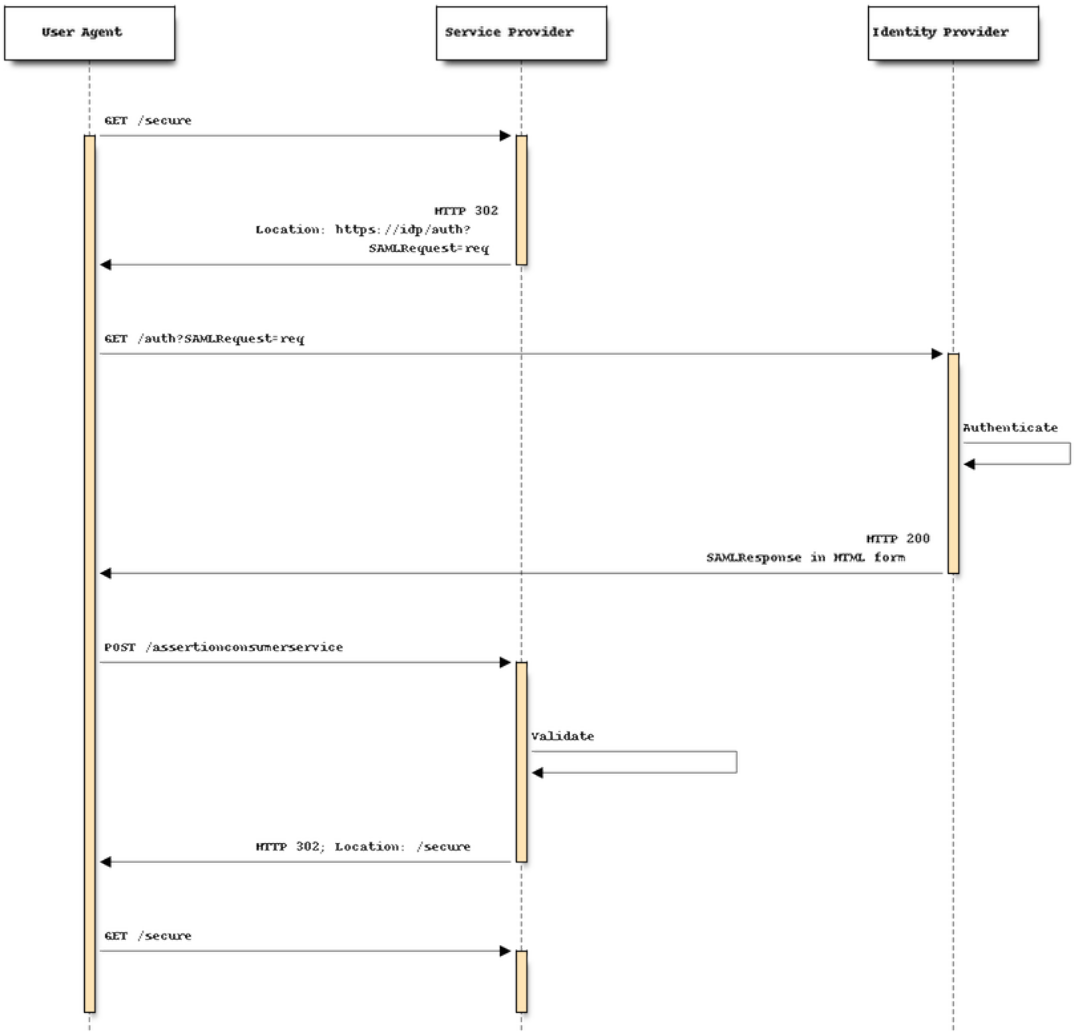
Normal keystone



In a normal keystone flow, the user requests a scoped token directly from keystone. Keystone accepts their credentials and checks them against its local storage or against its LDAP backend. Then it checks the scope that the user is requesting, ensuring they have the correct role assignments, and produces a scoped token. The user can use the scoped token to do something else in OpenStack, like request servers, but everything that happens after the token is produced is irrelevant to this discussion.

SAML2.0

SAML2.0 WebSSO



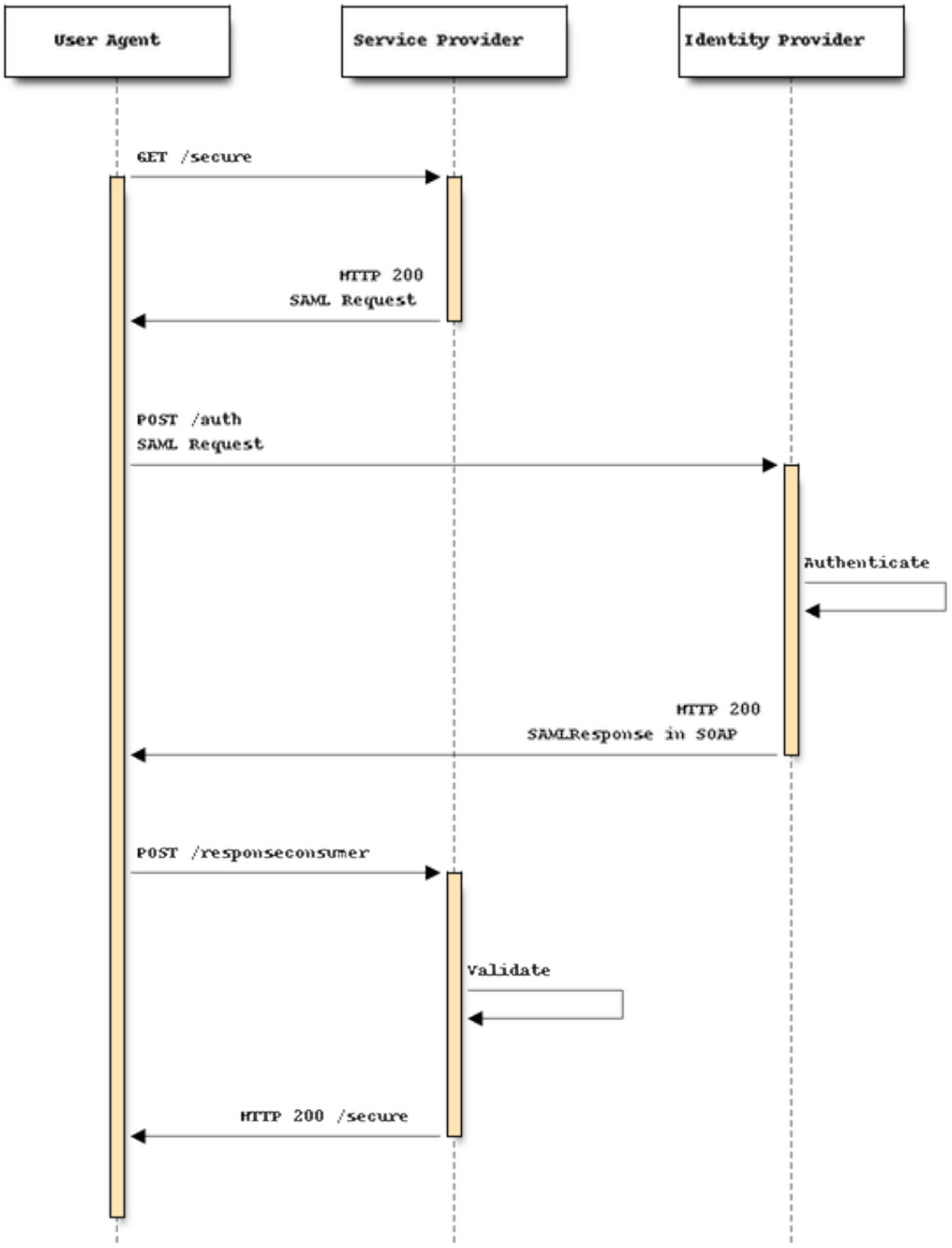
This diagram shows a standard WebSSO authentication flow, not one involving keystone. WebSSO is one of a few SAML2.0 profiles. It is based on the idea that a web browser will be acting as an intermediary and so the flow involves concepts that a browser can understand and act on, like HTTP redirects and HTML forms.

First, the user uses their web browser to request some secure resource from the Service Provider. The Service Provider detects that the user isnt authenticated yet, so it generates a SAML Request which it base64 encodes, and then issues an HTTP redirect to the Identity Provider.

The browser follows the redirect and presents the SAML Request to the Identity Provider. The user is prompted to authenticate, probably by filling out a username and password in a login page. The Identity Provider responds with an HTTP success and generates a SAML Response with an HTML form.

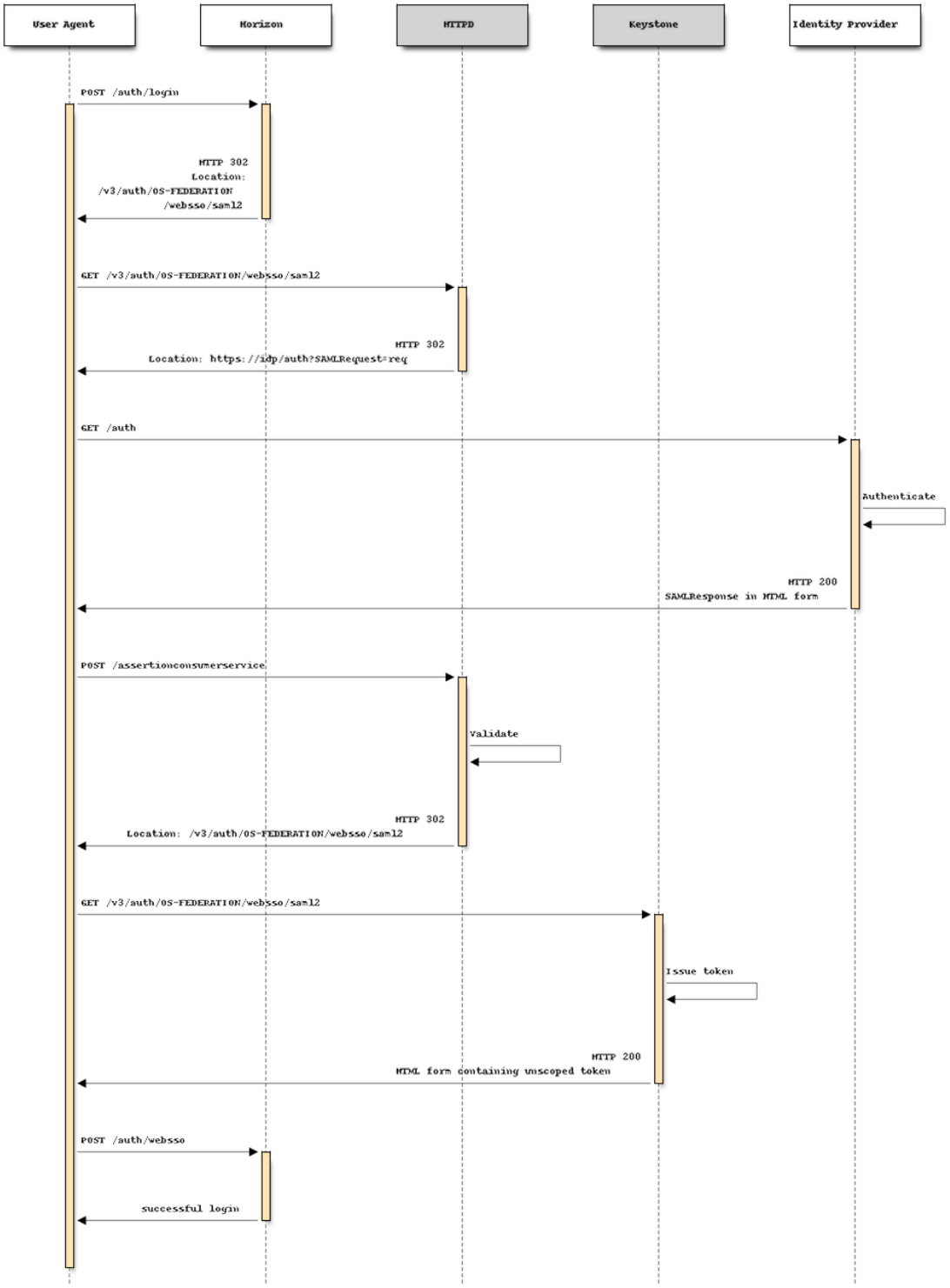
The browser automatically POSTs the form back to the Service Provider, which validates the SAML Response. The Service Provider finally issues another redirect back to the original resource the user had requested.

SAML2.0 ECP



ECP is another SAML profile. Generally the flow is similar to the WebSSO flow, but it is designed for a client that natively understands SAML, for example the keystoneauth library (and therefore also the python-openstackclient CLI tool). ECP is slightly different from the browser-based flow and is not supported by all SAML2.0 IdPs, and so getting WebSSO working does not necessarily mean ECP is working correctly, or vice versa. ECP support must often be turned on explicitly in the Identity Provider.

WebSSO with keystone and horizon



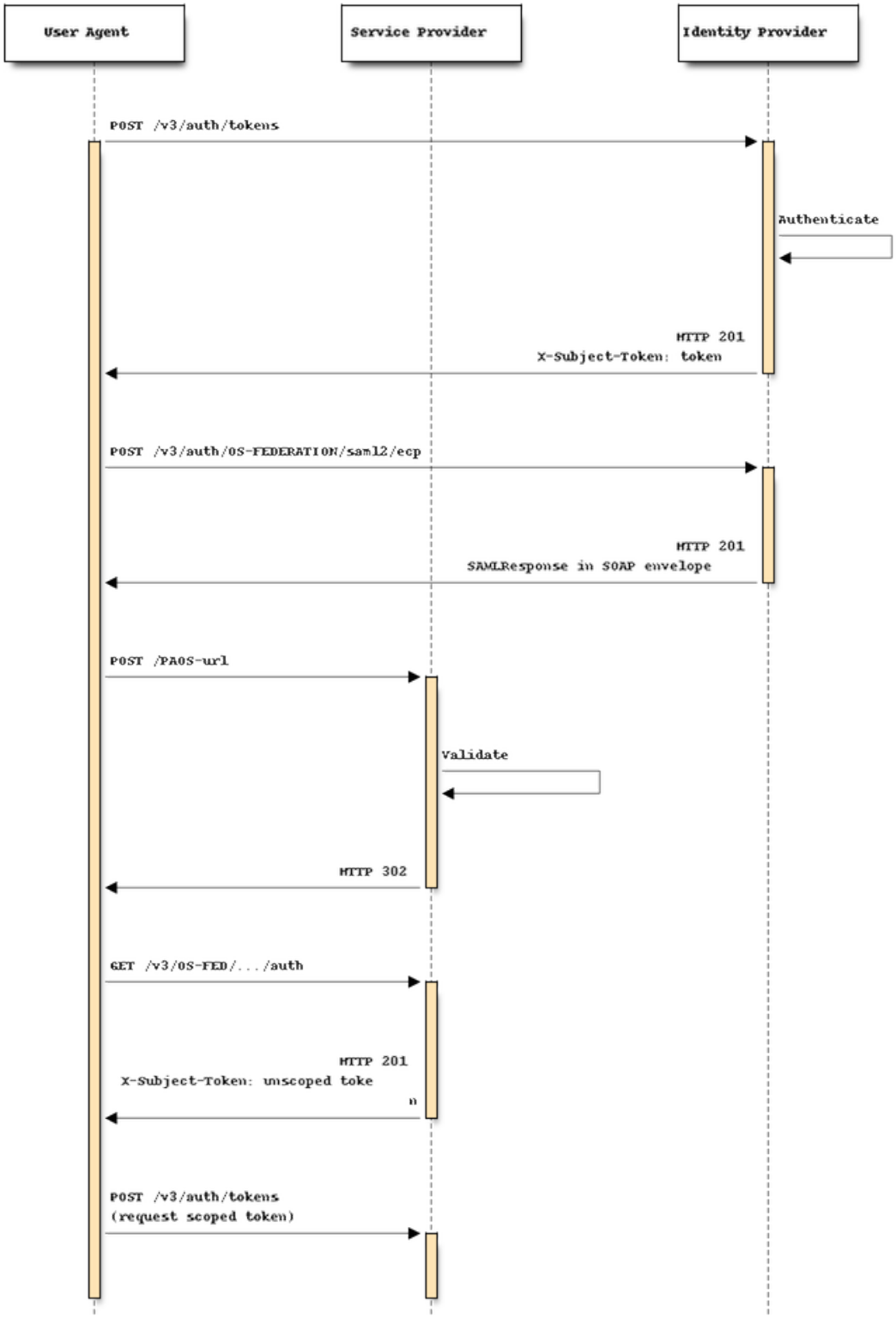
Keystone is not a web front-end, which means horizon needs to handle some parts of being a Service Provider to implement WebSSO.

In the diagram above, horizon is added, and keystone and HTTPD are split out from each other to distinguish which parts each are responsible for, though typically both together are referred to as the Service Provider.

In this model, the user requests to log in to horizon by selecting a federated authentication method from a dropdown menu. Horizon automatically generates a keystone URL based on the Identity Provider and protocol selected and redirects the browser to keystone. That location is equivalent to the /secure resource in the *SAML2.0 WebSSO* diagram. The browser follows the redirect, and the HTTPD module detects that the user isnt logged in yet and issues another redirect to the Identity Provider with a SAML Request. At this point, the flow is the same as in the normal WebSSO model. The user logs into the Identity Provider, a SAML Response is POSTed back to the Service Provider, where the HTTPD module validates the response and issues a redirect back to the location that horizon had originally requested, which is a special federation auth endpoint. At this point keystone is able to grant an unscoped token, which it hands off as another HTML form. The browser will POST that back to horizon, which triggers the normal login process, picking a project to scope to and getting a scoped token from keystone.

Note that horizon is acting as a middleman, since it knows the endpoint of the secure resource it requests from keystone.

Keystone to Keystone



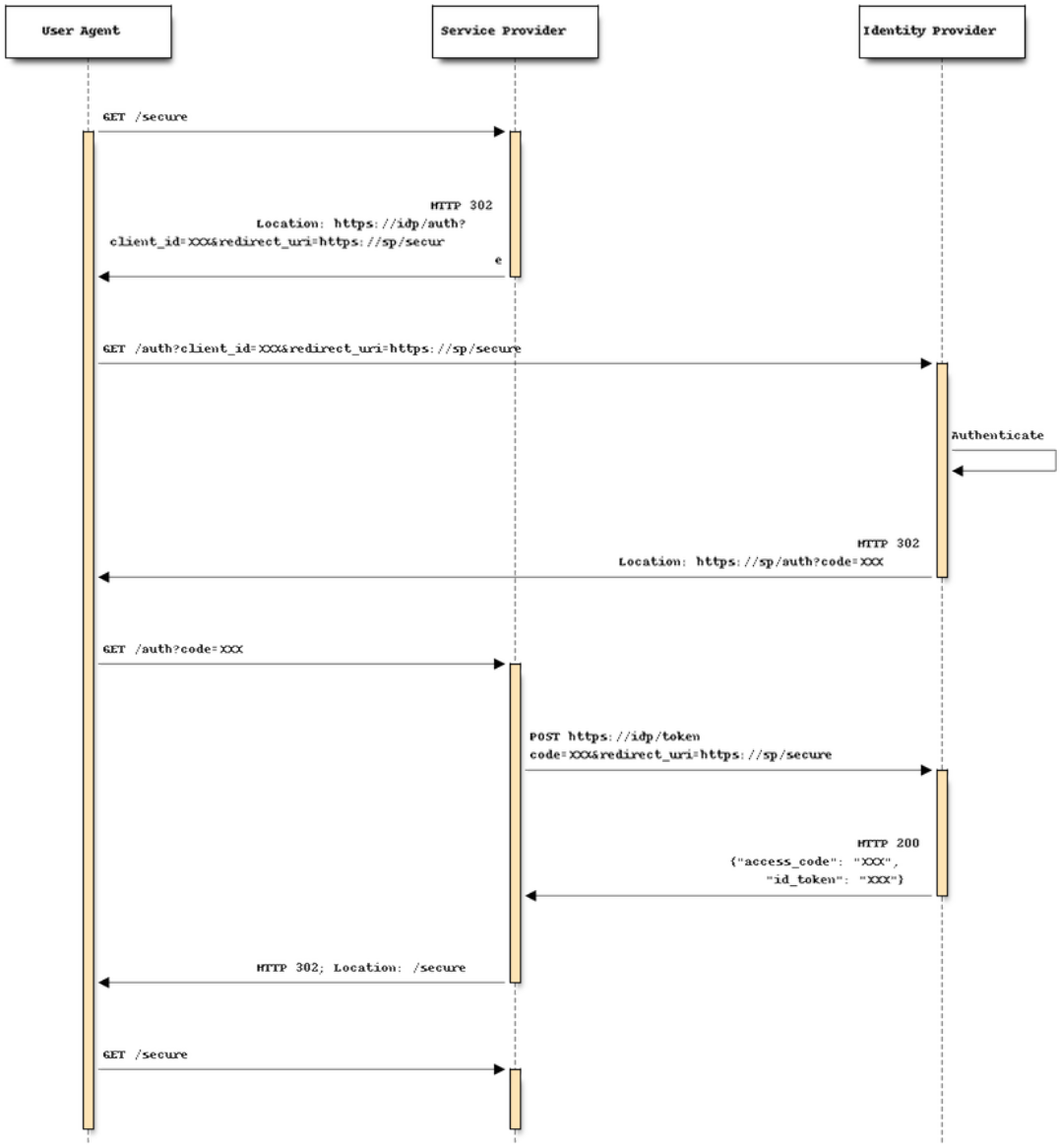
When keystone is used as an Identity Provider in a Keystone to Keystone configuration, the auth flow is nonstandard. It is similar to an IdP-initiated auth flow. In this case, the user goes directly to the Identity Provider first before requesting any resource from the Service Provider. The user will get a token from

keystone, then use that to request a SAML Response via ECP. When it gets that response back, it POSTs that to the Service Provider, which will grant a token for it.

Notice that the Service Provider has to accept data from the Identity Provider and therefore needs to have a way of trusting it. The Identity Provider, on the other hand, never has to accept data from the Service Provider. There is no back and forth, the user simply completes the auth process on one side and presents the result to the other side.

OpenID Connect

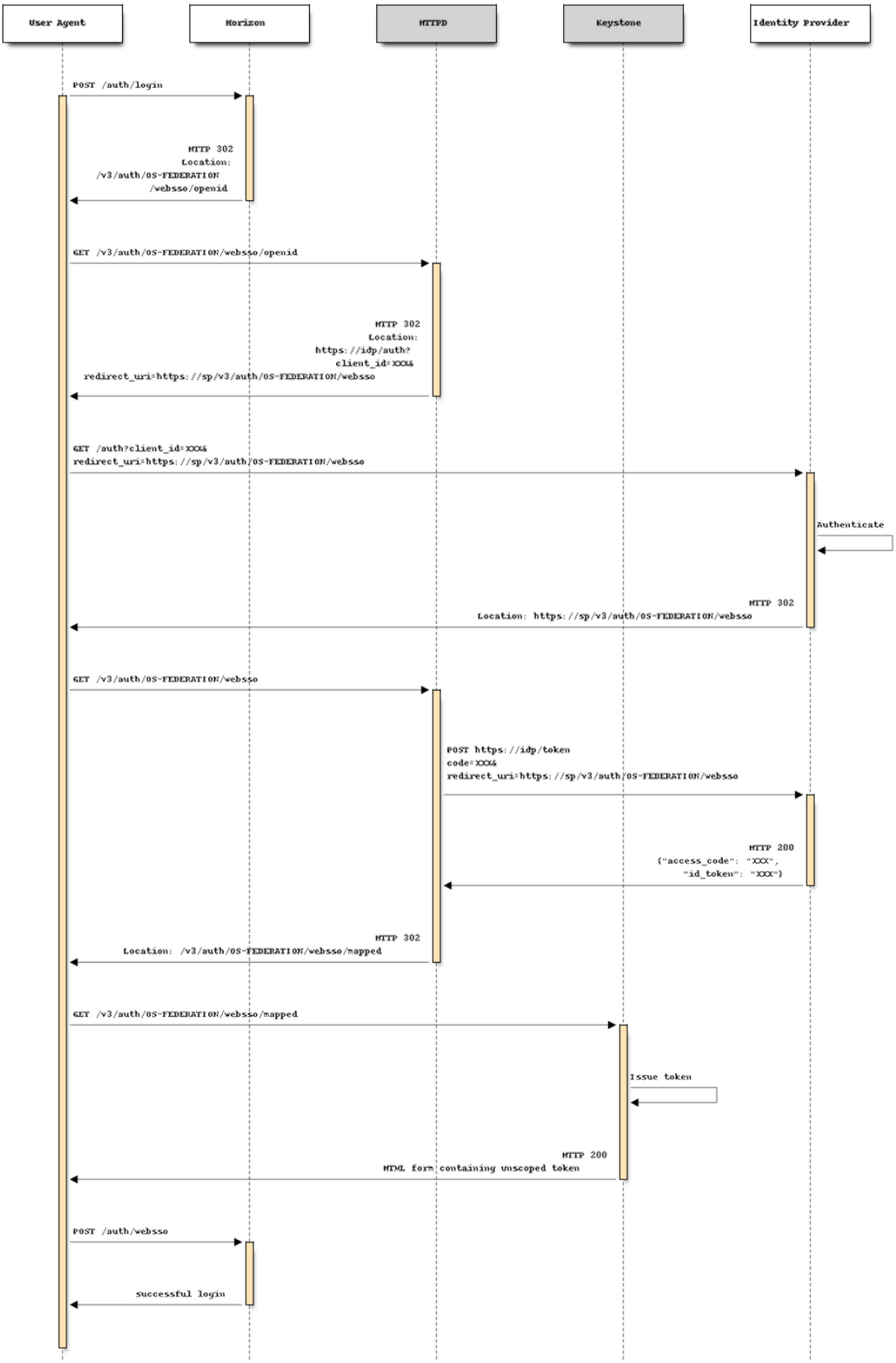
OpenID Connect Authentication Flow



OpenID Connect is different from any SAML2.0 flow because the negotiation is not handled entirely through the client. The Service Provider must make a request directly to the Identity Provider, which means this flow would not be appropriate if the Service Provider and Identity Provider are in segregated networks.

When the user requests a secure resource from the Service Provider, they are redirected to the Identity Provider to log in. The Identity Provider then redirects the user back to the Service Provider using a known redirect URI and providing an authorization code. The Service Provider must then make a back-channel request directly to the Identity Provider using the provided code, and exchange it for an ID token.

OpenID Connect with keystone and horizon



From horizon and keystones point of view, the authentication flow is the same for OpenID Connect as it

is for SAML2.0. It is only the HTTPD OpenIDC module that must handle the flow in accordance with the spec.

Configuring Keystone for Federation

Keystone as a Service Provider (SP)

Prerequisites

If you are not familiar with the idea of federated identity, see the *Introduction to Keystone Federation* first.

In this section, we will configure keystone as a Service Provider, consuming identity properties issued by an external Identity Provider, such as SAML assertions or OpenID Connect claims. For testing purposes, we recommend using samltest.id as a SAML Identity Provider, or Google as an OpenID Connect Identity Provider, and the examples here will references those providers. If you plan to set up *Keystone as an Identity Provider (IdP)*, it is easiest to set up keystone with a dummy SAML provider first and then reconfigure it to point to the keystone Identity Provider later.

The following configuration steps were performed on a machine running Ubuntu 16.04 and Apache 2.4.18.

To enable federation, youll need to run keystone behind a web server such as Apache rather than running the WSGI application directly with uWSGI or Gunicorn. See the installation guide for *SUSE*, *RedHat* or *Ubuntu* to configure the Apache web server for keystone.

Throughout the rest of the guide, you will need to decide on three pieces of information and use them consistently throughout your configuration:

1. The protocol name. This must be a valid keystone auth method and must match one of: `saml2`, `openid`, `mapped` or a *custom auth method* for which you must *register as an external driver*.
2. The identity provider name. This can be arbitrary.
3. The entity ID of the service provider. This should be a URN but need not resolve to anything.

You will also need to decide what HTTPD module to use as a Service Provider. This guide provides examples for `mod_shib` and `mod_auth_mellon` as SAML service providers, and `mod_auth_openidc` as an OpenID Connect Service Provider.

Note

In this guide, the keystone Service Provider is configured on a host called sp.keystone.example.org listening on the standard HTTPS port. All keystone paths will start with the keystone version prefix, `/v3`. If you have configured keystone to listen on port 5000, or to respond on the path `/identity` (for example), take this into account in your own configuration.

Creating federation resources in keystone

You need to create three resources via the keystone API to identify the Identity Provider to keystone and align remote user attributes with keystone objects:

- *Create an Identity Provider*
- *Create a Mapping*
- *Create a Protocol*

See also the keystone federation API reference.

Create an Identity Provider

Create an Identity Provider object in keystone, which represents the Identity Provider we will use to authenticate end users:

```
$ openstack identity provider create --remote-id https://samltest.id/saml/idp␣
↪samltest
```

The value for the `remote-id` option is the unique identifier provided by the Identity Provider, called the *entity ID* or the *remote ID*. For a SAML Identity Provider, it can found by querying its metadata endpoint:

```
$ curl -s https://samltest.id/saml/idp | grep -o 'entityID=".*"'
entityID="https://samltest.id/saml/idp"
```

For an OpenID Connect IdP, it is the Identity Providers Issuer Identifier. A remote ID must be globally unique: two identity providers cannot be associated with the same remote ID. The remote ID will usually appear as a URN but need not be a resolvable URL.

The local name, called `samltest` in our example, is decided by you and will be used by the mapping and protocol, and later for authentication.

Note

An identity provider keystone object may have multiple `remote-ids` specified, this allows the same *keystone* identity provider resource to be used with multiple external identity providers. For example, an identity provider resource `university-idp`, may have the following `remote_ids`: `['university-x', 'university-y', 'university-z']`. This removes the need to configure N identity providers in keystone.

See also the API reference on identity providers.

Create a Mapping

Next, create a mapping. A mapping is a set of rules that link the attributes of a remote user to user properties that keystone understands. It is especially useful for granting remote users authorization to keystone resources, either by associating them with a local keystone group and inheriting its role assignments, or dynamically provisioning projects within keystone based on these rules.

Note

By default, group memberships that a user gets from a mapping are only valid for the duration of the token. It is possible to persist these groups memberships for a limited period of time. To enable this, either set the `authorization_ttl` attribute of the identity provider, or the ``[federation] default_authorization_ttl` in the keystone.conf file. This value is in minutes, and will result in a lag from when a user is removed from a group in the identity provider, and when that will happen in keystone. Please consider your security requirements carefully.

An Identity Provider has exactly one mapping specified per protocol. Mapping objects can be used multiple times by different combinations of Identity Provider and Protocol.

As a simple example, create a mapping with a single rule to map all remote users to a local user in a single group in keystone:

```
$ cat > rules.json <<EOF
[
    {
        "local": [
            {
                "user": {
                    "name": "{0}"
                },
                "group": {
                    "domain": {
                        "name": "Default"
                    },
                    "name": "federated_users"
                }
            }
        ],
        "remote": [
            {
                "type": "REMOTE_USER"
            }
        ]
    }
]
EOF
$ openstack mapping create --rules rules.json samltest_mapping
```

This mapping rule evaluates the `REMOTE_USER` variable set by the HTTPD auth module and uses it to fill in the name of the local user in keystone. It also ensures all remote users become effective members of the `federated_users` group, thereby inheriting the groups role assignments.

In this example, the `federated_users` group must exist in the keystone Identity backend and must have a role assignment on some project, domain, or system in order for federated users to have an authorization in keystone. For example, to create the group:

```
$ openstack group create federated_users
```

Create a project these users should be assigned to:

```
$ openstack project create federated_project
```

Assign the group a `member` role in the project:

```
$ openstack role add --group federated_users --project federated_project␣
↪member
```

Mappings can be quite complex. A detailed guide can be found on the *Mapping Combinations* page.

See also the API reference on mapping rules.

Create a Protocol

Now create a federation protocol. A federation protocol object links the Identity Provider to a mapping.

You can create a protocol like this:

```
$ openstack federation protocol create saml2 \
--mapping samltest_mapping --identity-provider samltest
```

As mentioned in *Prerequisites*, the name you give the protocol is not arbitrary, it must be a valid auth method.

See also the API reference for federation protocols.

Configuring an HTTPD auth module

This guide currently only includes examples for the Apache web server, but it possible to use SAML, OpenIDC, and other auth modules in other web servers. See the installation guides for running keystone behind Apache for *SUSE*, *RedHat* or *Ubuntu*.

Configure protected endpoints

There is a minimum of one endpoint that must be protected in the VirtualHost configuration for the keystone service:

```
<Location /v3/OS-FEDERATION/identity_providers/IDENTITYPROVIDER/protocols/
↪PROTOCOL/auth>
  Require valid-user
  AuthType [...]
  ...
</Location>
```

This is the endpoint for federated users to request an unscoped token.

If configuring WebSSO, you should also protect one or both of the following endpoints:

```
<Location /v3/auth/OS-FEDERATION/websso/PROTOCOL>
  Require valid-user
  AuthType [...]
  ...
</Location>
<Location /v3/auth/OS-FEDERATION/identity_providers/IDENTITYPROVIDER/
↪protocols/PROTOCOL/websso>
  Require valid-user
  AuthType [...]
  ...
</Location>
```

The first example only specifies a protocol, and keystone will use the incoming remote ID to determine the Identity Provider. The second specifies the Identity Provider directly, which must then be supplied to horizon when configuring *horizon for WebSSO*.

The path must exactly match the path that will be used to access the keystone service. For example, if the identity provider you created in *Create an Identity Provider* is `samltest` and the protocol you created in *Create a Protocol* is `saml2`, then the Locations will be:

```
<Location /v3/OS-FEDERATION/identity_providers/samltest/protocols/saml2/auth>
  Require valid-user
  AuthType [...]
  ...
</Location>
<Location /v3/auth/OS-FEDERATION/websso/saml2>
  Require valid-user
  AuthType [...]
  ...
</Location>
<Location /v3/auth/OS-FEDERATION/identity_providers/samltest/protocols/saml2/
↪websso>
  Require valid-user
  AuthType [...]
  ...
</Location>
```

However, if you have configured the keystone service to use a virtual path such as `/identity`, that part of the path should be included:

```
<Location /identity/v3/OS-FEDERATION/identity_providers/samltest/protocols/
↪saml2/auth>
  Require valid-user
  AuthType [...]
  ...
</Location>
...
```

Configure the auth module

If your Identity Provider is a SAML IdP, there are two main Apache modules that can be used as a SAML Service Provider: *mod_shib* and *mod_auth_mellon*. For an OpenID Connect Identity Provider, *mod_auth_openidc* is used. You can also use other auth modules such as kerberos, X.509, or others. Check the documentation for the provider you choose for detailed installation and configuration guidance.

Depending on the Service Provider module youve chosen, you will need to install the applicable Apache module package and follow additional configuration steps. This guide contains examples for two major federation protocols:

- SAML2.0 - see guides for the following implementations:
  - *Set up mod_shib*.
  - *Set up mod_auth_mellon*.
- OpenID Connect: *Set up mod_auth_openidc*.

Configuring Keystone

While the Apache module does the majority of the heavy lifting, minor changes are needed to allow keystone to allow and understand federated authentication.

Add the Auth Method

Add the authentication methods to the `[auth]` section in `keystone.conf`. The auth method here must have the same name as the protocol you created in *Create a Protocol*. You should also remove `external` as an allowable method.

```
[auth]
methods = password,token,saml2,openid
```

Configure the Remote ID Attribute

Keystone is mostly apathetic about what HTTPD auth module you choose to configure for your Service Provider, but must know what header key to look for from the auth module to determine the Identity Providers remote ID so it can associate the incoming request with the Identity Provider resource. The key name is decided by the auth module choice:

- For `mod_shib`: use `Shib-Identity-Provider`
- For `mod_auth_mellon`: the attribute name is configured with the `MellonIdP` parameter in the VirtualHost configuration, if set to e.g. `IDP` then use `MELLON_IDP`
- For `mod_auth_openidc`: the attribute name is related to the `OIDCClaimPrefix` parameter in the Apache configuration, if set to e.g. `OIDC-` use `HTTP_OIDC_ISS`

It is recommended that this option be set on a per-protocol basis by creating a new section named after the protocol:

```
[saml2]
remote_id_attribute = Shib-Identity-Provider
[openid]
remote_id_attribute = HTTP_OIDC_ISS
```

Alternatively, a generic option may be set at the `[federation]` level.

```
[federation]
remote_id_attribute = HTTP_OIDC_ISS
```

Add a Trusted Dashboard (WebSSO)

If you intend to configure horizon as a WebSSO frontend, you must specify the URLs of trusted horizon servers. This value may be repeated multiple times. This setting ensures that keystone only sends token data back to trusted servers. This is performed as a precaution, specifically to prevent man-in-the-middle (MITM) attacks. The value must exactly match the origin address sent by the horizon server, including any trailing slashes.

```
[federation]
trusted_dashboard = https://horizon1.example.org/auth/websso/
trusted_dashboard = https://horizon2.example.org/auth/websso/
```

Add the Callback Template (WebSSO)

If you intend to configure horizon as a WebSSO frontend, and if not already done for you by your distributions keystone package, copy the sso_callback_template.html template into the location specified by the `[federation]/sso_callback_template` option in `keystone.conf`. You can also use this template as an example to create your own custom HTML redirect page.

Restart the keystone WSGI service or the Apache frontend service after making changes to your keystone configuration.

```
# systemctl restart apache2
```

Configuring Horizon as a WebSSO Frontend

Note

Consult horizons official documentation for details on configuring horizon.

Keystone on its own is not capable of supporting a browser-based Single Sign-on authentication flow such as the SAML2.0 WebSSO profile, therefore we must enlist horizons assistance. Horizon can be configured to support SSO by enabling it in horizons `local_settings.py` configuration file and adding the possible authentication choices that will be presented to the user on the login screen.

Ensure the *WEBSSO_ENABLED* option is set to *True* in horizons local_settings.py file, this will provide users with an updated login screen for horizon.

```
WEBSSO_ENABLED = True
```

Configure the options for authenticating that a user may choose from at the login screen. The pairs configured in this list map a user-friendly string to an authentication option, which may be one of:

- The string `credentials` which forces horizon to present its own username and password fields that the user will use to authenticate as a local keystone user
- The name of a protocol that you created in *Create a Protocol*, such as `saml2` or `openid`, which will cause horizon to call keystones WebSSO API without an Identity Provider to authenticate the user
- A string that maps to an Identity Provider and Protocol combination configured in `WEBSSO_IDP_MAPPING` which will cause horizon to call keystones WebSSO API specific to the given Identity Provider.

```
WEBSSO_CHOICES = (
    ("credentials", _("Keystone Credentials")),
    ("openid", _("OpenID Connect")),
    ("saml2", _("Security Assertion Markup Language")),
    ("myidp_openid", "Acme Corporation - OpenID Connect"),
    ("myidp_saml2", "Acme Corporation - SAML2")
)

WEBSSO_IDP_MAPPING = {
    "myidp_openid": ("myidp", "openid"),
    "myidp_saml2": ("myidp", "saml2")
}
```

The initial selection of the dropdown menu can also be configured:

```
WEBSSO_INITIAL_CHOICE = "credentials"
```

Remember to restart the web server when finished configuring horizon:

```
# systemctl restart apache2
```

Authenticating

Use the CLI to authenticate with a SAML2.0 Identity Provider

The `python-openstackclient` can be used to authenticate a federated user in a SAML Identity Provider to keystone.

Note

The SAML Identity Provider must be configured to support the ECP authentication profile.

To use the CLI tool, you must have the name of the Identity Provider resource in keystone, the name of the federation protocol configured in keystone, and the ECP endpoint for the Identity Provider. If you

are the cloud administrator, the name of the Identity Provider and protocol was configured in *Create an Identity Provider* and *Create a Protocol* respectively. If you are not the administrator, you must obtain this information from the administrator.

The ECP endpoint for the Identity Provider can be obtained from its metadata without involving an administrator. This endpoint is the `urn:oasis:names:tc:SAML:2.0:bindings:SOAP` binding in the metadata document:

```
$ curl -s https://samltest.id/saml/idp | grep urn:oasis:names:tc:SAML:2.
↪0:bindings:SOAP
    <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"␣
↪Location="https://samltest.id/idp/profile/SAML2/SOAP/ECP"/>
```

Find available scopes

If you are a new user and are not aware of what resources you have access to, you can use an unscoped query to list the projects or domains you have been granted a role assignment on:

```
export OS_AUTH_TYPE=v3samlpassword
export OS_IDENTITY_PROVIDER=samltest
export OS_IDENTITY_PROVIDER_URL=https://samltest.id/idp/profile/SAML2/SOAP/ECP
export OS_PROTOCOL=saml2
export OS_USERNAME=morty
export OS_PASSWORD=panic
export OS_AUTH_URL=https://sp.keystone.example.org/v3
export OS_IDENTITY_API_VERSION=3
openstack federation project list
openstack federation domain list
```

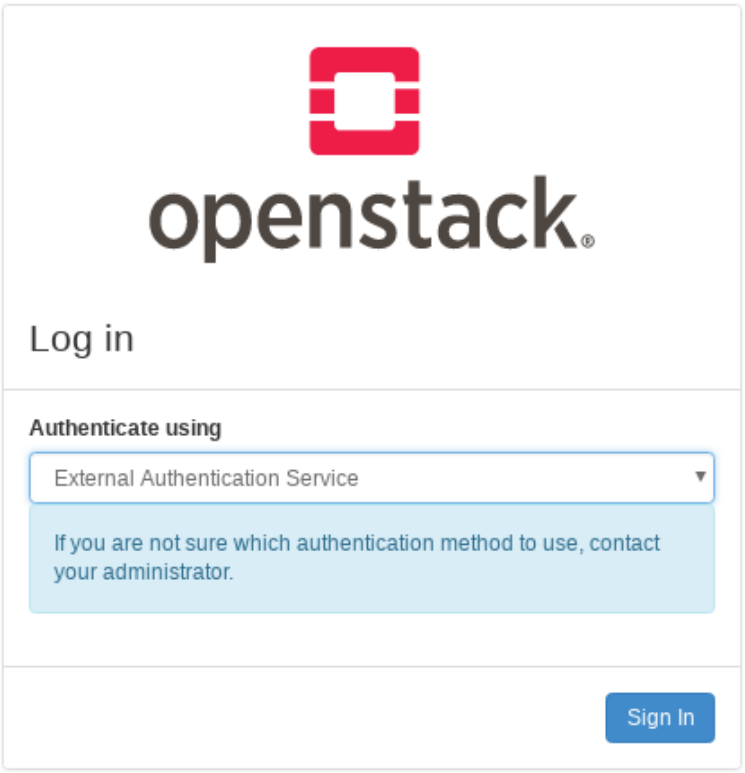
Get a scoped token

If you already know the project, domain or system you wish to scope to, you can directly request a scoped token:

```
export OS_AUTH_TYPE=v3samlpassword
export OS_IDENTITY_PROVIDER=samltest
export OS_IDENTITY_PROVIDER_URL=https://samltest.id/idp/profile/SAML2/SOAP/ECP
export OS_PROTOCOL=saml2
export OS_USERNAME=morty
export OS_PASSWORD=panic
export OS_AUTH_URL=https://sp.keystone.example.org/v3
export OS_IDENTITY_API_VERSION=3
export OS_PROJECT_NAME=federated_project
export OS_PROJECT_DOMAIN_NAME=Default
openstack token issue
```

Use horizon to authenticate with an external Identity Provider

When horizon is configured to enable WebSSO, a dropdown menu will appear on the login screen before the user has authenticated. Select an authentication method from the menu to be redirected to your Identity Provider for authentication.



Keystone as an Identity Provider (IdP)

Prerequisites

When keystone is configured as an Identity Provider, it is often referred to as *Keystone to Keystone*, because it enables federation between multiple OpenStack clouds using the SAML2.0 protocol.

If you are not familiar with the idea of federated identity, see the *introduction* first.

When setting up *Keystone to Keystone*, it is easiest to *configure a keystone Service Provider* first with a sandbox Identity Provider such as samltest.id.

This feature requires installation of the xmlsec1 tool via your distribution packaging system (for instance apt or yum)

```
# apt-get install xmlsec1
```

Note

In this guide, the keystone Identity Provider is configured on a host called idp.keystone.example.org listening on the standard HTTPS port. All keystone paths will start with the keystone version prefix,

`/v3`. If you have configured keystone to listen on port 5000, or to respond on the path `/identity` (for example), take this into account in your own configuration.

Configuring Metadata

Since keystone is acting as a SAML Identity Provider, its metadata must be configured in the `[saml]` section (not to be confused with an optional `[saml2]` section which you may have configured in *Configure the Remote Id Attribute* while setting up keystone as Service Provider) of `keystone.conf` so that it can served by the metadata API.

The two parameters that **must** be set in order for keystone to generate metadata are `idp_entity_id` and `idp_sso_endpoint`:

```
[saml]
idp_entity_id=https://idp.keystone.example.org/v3/OS-FEDERATION/saml2/idp
idp_sso_endpoint=https://idp.keystone.example.org/v3/OS-FEDERATION/saml2/sso
```

`idp_entity_id` sets the Identity Provider entity ID, which is a string of your choosing that uniquely identifies the Identity Provider to any Service Provider.

`idp_sso_endpoint` is required to generate valid metadata, but its value is currently not used because keystone as an Identity Provider does not support the SAML2.0 WebSSO auth profile. This may change in the future which is why there is no default value provided and must be set by the operator.

For completeness, the following Organization and Contact configuration options should also be updated to reflect your organization and administrator contact details.

```
idp_organization_name=example_company
idp_organization_display_name=Example Corp.
idp_organization_url=example.com
idp_contact_company=example_company
idp_contact_name=John
idp_contact_surname=Smith
idp_contact_email=jsmith@example.com
idp_contact_telephone=555-555-5555
idp_contact_type=technical
```

It is important to take note of the default `certfile` and `keyfile` options, and adjust them if necessary:

```
certfile=/etc/keystone/ssl/certs/signing_cert.pem
keyfile=/etc/keystone/ssl/private/signing_key.pem
```

You must generate a PKI key pair and copy the files to these paths. You can use the `openssl` tool to do so. Keystone does not provide a utility for this.

Check the `idp_metadata_path` setting and adjust it if necessary:

```
idp_metadata_path=/etc/keystone/saml2_idp_metadata.xml
```

To create metadata for your keystone IdP, run the `keystone-manage` command and redirect the output to a file. For example:

```
# keystone-manage saml_idp_metadata > /etc/keystone/saml2_idp_metadata.xml
```

Finally, restart the keystone WSGI service or the web server frontend:

```
# systemctl restart apache2
```

Creating a Service Provider Resource

Create a Service Provider resource to represent your Service Provider as an object in keystone:

```
$ openstack service provider create keystonesp \
--service-provider-url https://sp.keystone.example.org/Shibboleth.sso/SAML2/
↪ECP
--auth-url https://sp.keystone.example.org/v3/OS-FEDERATION/identity_
↪providers/keystoneidp/protocols/saml2/auth
```

The `--auth-url` is the federated auth endpoint for a specific Identity Provider and protocol name, here named `keystoneidp` and `saml2`.

The `--service-provider-url` is the `urn:oasis:names:tc:SAML:2.0:bindings:PAOS` binding for the Assertion Consumer Service of the Service Provider. It can be obtained from the Service Provider metadata:

```
$ curl -s https://sp.keystone.example.org/Shibboleth.sso/Metadata | grep␣
↪urn:oasis:names:tc:SAML:2.0:bindings:PAOS
<md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.
↪0:bindings:PAOS" Location="https://sp.keystone.example.org/Shibboleth.sso/
↪SAML2/ECP" index="4"/>
```

Authenticating

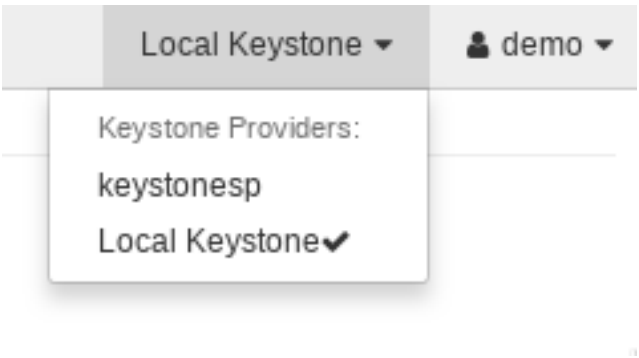
Use the CLI to authenticate with Keystone-to-Keystone

Use `python-openstackclient` to authenticate with the IdP and then get a scoped token from the SP.

```
export OS_USERNAME=demo
export OS_PASSWORD=nomoresecret
export OS_AUTH_URL=https://idp.keystone.example.org/v3
export OS_IDENTITY_API_VERSION=3
export OS_PROJECT_NAME=federated_project
export OS_PROJECT_DOMAIN_NAME=Default
export OS_SERVICE_PROVIDER=keystonesp
export OS_REMOTE_PROJECT_NAME=federated_project
export OS_REMOTE_PROJECT_DOMAIN_NAME=Default
openstack token issue
```

Use Horizon to switch clouds

No additional configuration is necessary to enable horizon for Keystone to Keystone. Log into the horizon instance for the Identity Provider using your regular local keystone credentials. Once logged in, you will see a Service Provider dropdown menu which you can use to switch your dashboard view to another cloud.



Setting Up OpenID Connect

See *Keystone as a Service Provider (SP)* before proceeding with these OpenIDC-specific instructions.

These examples use Google as an OpenID Connect Identity Provider. The Service Provider must be added to the Identity Provider in the Google API console.

Configuring Apache HTTPD for mod_auth_openidc

Note

You are advised to carefully examine the mod_auth_openidc documentation.

Install the Module

Install the Apache module package. For example, on Ubuntu:

```
# apt-get install libapache2-mod-auth-openidc
```

The package and module name will differ between distributions.

Configure mod_auth_openidc

In the Apache configuration for the keystone VirtualHost, set the following OIDC options:

```
OIDCClaimPrefix "OIDC-"
OIDCResponseType "id_token"
OIDCScope "openid email profile"
OIDCProviderMetadataURL https://accounts.google.com/.well-known/openid-
↪configuration
```

(continues on next page)

(continued from previous page)

```
OIDCOAuthVerifyJwksUri https://www.googleapis.com/oauth2/v3/certs
OIDCClientID <openid_client_id>
OIDCClientSecret <openid_client_secret>
OIDCCryptoPassphrase <random string>
OIDCRedirectURI https://sp.keystone.example.org/v3/redirect_uri
```

`OIDCScope` is the list of attributes that the user will authorize the Identity Provider to send to the Service Provider. `OIDCClientID` and `OIDCClientSecret` must be generated and obtained from the Identity Provider. `OIDCProviderMetadataURL` is a URL from which the Service Provider will fetch the Identity Providers metadata. `OIDCOAuthVerifyJwksUri` is a URL from which the Service Provider will download the public key from the Identity Provider to check if the users access token is valid or not, this configuration must be used while using the AuthType `auth-openidc`, when using the AuthType `openid-connect` and the OIDCProviderMetadataURL is configured, this property will not be necessary. `OIDCRedirectURI` is a vanity URL that must point to a protected path that does not have any content, such as an extension of the protected federated auth path. It should not match any Keystone API endpoints or mod_auth_openidc will handle requests to the endpoint instead of Keystone. This can lead to unusual errors and behaviors from Keystone.

Note

If using a mod_wsgi version less than 4.3.0, then the *OIDCClaimPrefix* must be specified to have only alphanumerics or a dash (-). This is because mod_wsgi blocks headers that do not fit this criteria.

Configure Protected Endpoints

Configure each protected path to use the `openid-connect` AuthType:

```
<Location ~ "/redirect_uri">
    Require valid-user
    AuthType openid-connect
</Location>
<Location /v3/OS-FEDERATION/identity_providers/google/protocols/openid/auth>
    Require valid-user
    AuthType openid-connect
</Location>
```

Note

To add support to Bearer Access Token authentication flow that is used by applications that do not adopt the browser flow, such the OpenStack CLI, you will need to change the AuthType from `openid-connect` to `auth-openidc`.

Do the same for the WebSSO auth paths if using horizon:

```
<Location /v3/auth/OS-FEDERATION/websso/openid>
    Require valid-user
    AuthType openid-connect
```

(continues on next page)

(continued from previous page)

```
</Location>
<Location /v3/auth/OS-FEDERATION/identity_providers/google/protocols/openid/
↪websso>
    Require valid-user
    AuthType openid-connect
</Location>
```

Remember to reload Apache after altering the VirtualHost:

```
# systemctl reload apache2
```

Note

When creating *mapping rules*, in keystone, note that the remote attributes will be prefixed, with `HTTP_`, so for instance, if you set `OIDCClaimPrefix` to `OIDC-`, then a typical remote value to check for is: `HTTP_OIDC_ISS`.

Configuring Multiple Identity Providers

To configure multiples Identity Providers in your environment you will need to set your OIDC options like the following options:

```
OIDCClaimPrefix "OIDC-"
OIDCResponseType "id_token"
OIDCScope "openid email profile"
OIDCMetadataDir <IDP metadata directory>
OIDCCryptoPassphrase <random string>
OIDCRedirectURI https://sp.keystone.example.org/redirect_uri
OIDCOAuthVerifyCertFiles <kid>#</path/to-cert.pem> <kid2>#</path/to-cert2.pem>
↪ <kidN>#</path/to-certN.pem>
```

The `OIDCOAuthVerifyCertFiles` is a tuple separated with *space* containing the key-id (kid) of the Issuers public key and a path to the Issuer certificate. The separator `#` is used to split the (`kid`) and the public certificate address

The metadata folder configured in the option `OIDCMetadataDir` must have all your Identity Providers configurations, the name of the files will be the name (with path) of the Issuers like:

```
- <IDP metadata directory>
  |
  - accounts.google.com.client
  |
  - accounts.google.com.conf
  |
  - accounts.google.com.provider
  |
  - keycloak.example.org%2Fauth%2Frealms%2Fidp.client
  |
```

(continues on next page)

(continued from previous page)

```
  - keycloak.example.org%2Fauth%2Frealms%2Fidp.conf
  |
  - keycloak.example.org%2Fauth%2Frealms%2Fidp.provider
```

Note

The name of the file must be url-encoded if needed, as the Apache2 mod_auth_openidc will get the raw value from the query parameter `iss` from the http request and check if there is a metadata with this name, as the query parameter is url-encoded, so the metadata file name need to be encoded too. For example, if you have an Issuer with `/` in the URL, then you need to escape it to `%2F` by applying a URL escape in the file name.

The content of these files must be a JSON like

`accounts.google.com.client`:

```
{
  "client_id":"<openid_client_id>",
  "client_secret":"<openid_client_secret>"
}
```

The `.client` file handles the SP credentials in the Issuer.

`accounts.google.com.conf`:

This file will be a JSON that overrides some of OIDC options. The options that are able to be overridden are listed in the OpenID Connect Apache2 plugin documentation.

If you do not want to override the config values, you can leave this file as an empty JSON like `{}`.

`accounts.google.com.provider`:

This file will contain all specifications about the IdentityProvider. To simplify, you can just use the JSON returned in the `.well-known` endpoint:

```
{
  "issuer": "https://accounts.google.com",
  "authorization_endpoint": "https://accounts.google.com/o/oauth2/v2/auth",
  "token_endpoint": "https://oauth2.googleapis.com/token",
  "userinfo_endpoint": "https://openidconnect.googleapis.com/v1/userinfo",
  "revocation_endpoint": "https://oauth2.googleapis.com/revoke",
  "jwks_uri": "https://www.googleapis.com/oauth2/v3/certs",
  "response_types_supported": [
    "code",
    "token",
    "id_token",
    "code token",
    "code id_token",
    "token id_token",
    "code token id_token",
    "none"
  ],
```

(continues on next page)

(continued from previous page)

```
  "subject_types_supported": [
   "public"
  ],
  "id_token_signing_alg_values_supported": [
   "RS256"
  ],
  "scopes_supported": [
   "openid",
   "email",
   "profile"
  ],
  "token_endpoint_auth_methods_supported": [
   "client_secret_post",
   "client_secret_basic"
  ],
  "claims_supported": [
   "aud",
   "email",
   "email_verified",
   "exp",
   "family_name",
   "given_name",
   "iat",
   "iss",
   "locale",
   "name",
   "picture",
   "sub"
  ],
  "code_challenge_methods_supported": [
   "plain",
   "S256"
  ]
}
```

Continue configuring keystone

Continue configuring keystone

Setting Up Mellon

See *Keystone as a Service Provider (SP)* before proceeding with these Mellon-specific instructions.

Configuring Apache HTTPD for mod_auth_mellon

Note

You are advised to carefully examine the mod_auth_mellon documentation.

Follow the steps outlined at: Keystone install guide for SUSE, RedHat or Ubuntu.

Install the Module

Install the Apache module package. For example, on Ubuntu:

```
# apt-get install libapache2-mod-auth-mellon
```

The package and module name will differ between distributions.

Configure mod_auth_mellon

Unlike `mod_shib`, all of `mod_auth_mellon`s configuration is done in Apache, not in a separate config file. Set up the shared settings in a single `<Location>` directive near the top in your keystone VirtualHost file, before your protected endpoints:

```
<Location /v3>
    MellonEnable "info"
    MellonSPPrivateKeyFile /etc/apache2/mellon/sp.keystone.example.org.key
    MellonSPCertFile /etc/apache2/mellon/sp.keystone.example.org.cert
    MellonSPMetadataFile /etc/apache2/mellon/sp-metadata.xml
    MellonIdPMetadataFile /etc/apache2/mellon/idp-metadata.xml
    MellonEndpointPath /v3/mellon
    MellonIdP "IDP"
</Location>
```

Configure Protected Endpoints

Configure each protected path to use the `Mellon` AuthType:

```
<Location /v3/OS-FEDERATION/identity_providers/samltest/protocols/saml2/auth>
   Require valid-user
   AuthType Mellon
   MellonEnable auth
</Location>
```

Do the same for the WebSSO auth paths if using horizon as a single sign-on frontend:

```
<Location /v3/auth/OS-FEDERATION/websso/saml2>
   Require valid-user
   AuthType Mellon
   MellonEnable auth
</Location>
<Location /v3/auth/OS-FEDERATION/identity_providers/samltest/protocols/saml2/
↪websso>
   Require valid-user
   AuthType Mellon
   MellonEnable auth
</Location>
```

Configure the Mellon Service Provider Metadata

Mellon provides a script called `mellon_create_metadata.sh``_ which generates the values for the config directives ``MellonSPPrivateKeyFile`, `MellonSPCertFile`, and `MellonSPMetadataFile`. Run the script:

```
$ ./mellon_create_metadata.sh \
https://sp.keystone.example.org/mellon \
http://sp.keystone.example.org/v3/OS-FEDERATION/identity_providers/samltest/
↪protocols/saml2/auth/mellon
```

The first parameter is used as the entity ID, a URN of your choosing that must uniquely identify the Service Provider to the Identity Provider. The second parameter is the full URL for the endpoint path corresponding to the parameter `MellonEndpointPath`.

After generating the keypair and metadata, copy the files to the locations given by the `MellonSPPrivateKeyFile` and `MellonSPCertFile` settings in your Apache configuration.

Upload the Service Providers Metadata file which you just generated to your Identity Provider. This is the file used as the value of the *MellonSPMetadataFile* in the config. The IdP may provide a webpage where you can upload the file, or you may be required to submit the file using *wget* or *curl*. Please check your IdP documentation for details.

Exchange Metadata

Fetch your Identity Providers Metadata file and copy it to the path specified by the `MellonIdPMetadataFile` setting in your Apache configuration.

```
$ wget -O /etc/apache2/mellon/idp-metadata.xml https://samltest.id/saml/idp
```

Remember to reload Apache after finishing configuring Mellon:

```
# systemctl reload apache2
```

Continue configuring keystone

Continue configuring keystone

Setting up Shibboleth

See *Keystone as a Service Provider (SP)* before proceeding with these Shibboleth-specific instructions.

Note

The examples below are for Ubuntu 16.04, for which only version 2 of the Shibboleth Service Provider is available. Version 3 is available for other distributions and the configuration should be identical to version 2.

Configuring Apache HTTPD for mod_shib

Note

You are advised to carefully examine the mod_shib Apache configuration documentation.

Configure keystone under Apache, following the steps in the install guide for SUSE, RedHat or Ubuntu.

Install the Module

Install the Apache module package. For example, on Ubuntu:

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

The package and module name will differ between distributions.

Configure Protected Endpoints

In the Apache configuration for the keystone VirtualHost, set an additional `<Location>` which is not part of keystones API:

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

If you are using `mod_proxy`, for example to proxy requests to the `/identity` path to keystones UWSGI service, you must exempt this Shibboleth endpoint from it:

```
Proxypass Shibboleth.sso !
```

Configure each protected path to use the `shibboleth` AuthType:

```
<Location /v3/OS-FEDERATION/identity_providers/samltest/protocols/saml2/auth>
    Require valid-user
    AuthType shibboleth
    ShibRequestSetting requireSession 1
    ShibExportAssertion off
    <IfVersion < 2.4>
        ShibRequireSession On
        ShibRequireAll On
    </IfVersion>
</Location>
```

Do the same for the WebSSO auth paths if using horizon as a single sign-on frontend:

```
<Location /v3/auth/OS-FEDERATION/websso/saml2>
    Require valid-user
    AuthType shibboleth
    ShibRequestSetting requireSession 1
    ShibExportAssertion off
    <IfVersion < 2.4>
        ShibRequireSession On
        ShibRequireAll On
    </IfVersion>
</Location>
<Location /v3/auth/OS-FEDERATION/identity_providers/samltest/protocols/saml2/
↪websso>
    Require valid-user
    AuthType shibboleth
    ShibRequestSetting requireSession 1
    ShibExportAssertion off
    <IfVersion < 2.4>
        ShibRequireSession On
        ShibRequireAll On
    </IfVersion>
</Location>
```

Remember to reload Apache after altering the VirtualHost:

```
# systemctl reload apache2
```

Configuring mod_shib

Note

You are advised to examine Shibboleth Service Provider Configuration documentation

Generate a keypair

For all SAML Service Providers, a PKI key pair must be generated and exchanged with the Identity Provider. The `mod_shib` package on the Ubuntu distribution provides a utility to generate the key pair:

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

which will generate a key pair under `/etc/shibboleth`. In other cases, the package might generate the key pair automatically upon installation.

Configure metadata

`mod_shib` also has its own configuration file at `/etc/shibboleth/shibboleth2.xml` that must be altered, as well as its own daemon. First, give the Service Provider an entity ID. This is a URN that you choose that must be globally unique to the Identity Provider:

```
<ApplicationDefaults entityID="https://sp.keystone.example.org/shibboleth"
    REMOTE_USER="eppn persistent-id targeted-id">
```

Depending on your Identity Provider, you may also want to change the REMOTE_USER setting, more on that in a moment.

Set the entity ID of the Identity Provider (this is the same as the value you provided for `--remote-id` in *Identity Provider*):

```
<SSO entityID="https://samltest.id/saml/idp">
```

Additionally, if you want to enable ECP (required for Keystone-to-Keystone), the SSO tag for this entity must also have the ECP flag set:

```
<SSO entityID="https://samltest.id/saml/idp" ECP="true">
```

Tell Shibboleth where to find the metadata of the Identity Provider. You could either tell it to fetch it from a URI or point it to a local file. For example, pointing to a local file:

```
<MetadataProvider type="XML" file="/etc/shibboleth/samltest-metadata.xml" />
```

or pointing to a remote location:

```
<MetadataProvider type="XML" url="https://samltest.id/saml/idp"
    backingFile="samltest-metadata.xml" />
```

When you are finished configuring `shibboleth2.xml`, restart the `shibd` daemon:

```
# systemctl restart shibd
```

Check the `shibd` logs in `/var/log/shibboleth/shibd.log` and `/var/log/shibboleth/shibd_warn.log` for errors or warnings.

Configure allowed attributes

Note

For more information see the attributes documentation

By default, `mod_shib` does not pass all attributes received from the Identity Provider to keystone. If your Identity Provider does not use attributes known to `shibd`, you must configure them. For example, *samltest.id* uses a custom UID attribute. It is not discoverable in the Identity Provider metadata, but the attribute name and type is logged in the `mod_shib` logs when an authentication attempt is made. To allow the attribute, add it to `/etc/shibboleth/attribute-map.xml`:

```
<Attribute name="urn:oid:0.9.2342.19200300.100.1.1" id="uid" />
```

You may also want to use that attribute as a value for the `REMOTE_USER` variable, which will make the `REMOTE_USER` variable usable as a parameter to your mapping rules. To do so, add it to `/etc/shibboleth/shibboleth2.xml`:

```
<ApplicationDefaults entityID="https://sp.keystone.example.org/shibboleth"
    REMOTE_USER="uid">
```

Similarly, if using keystone as your Identity Provider, several custom attributes will be needed in `/etc/shibboleth/attribute-map.xml`:

```
<Attribute name="openstack_user" id="openstack_user"/>
<Attribute name="openstack_roles" id="openstack_roles"/>
<Attribute name="openstack_project" id="openstack_project"/>
<Attribute name="openstack_user_domain" id="openstack_user_domain"/>
<Attribute name="openstack_project_domain" id="openstack_project_domain"/>
<Attribute name="openstack_groups" id="openstack_groups"/>
```

And update the `REMOTE_USER` variable in `/etc/shibboleth/shibboleth2.xml` if desired:

```
<ApplicationDefaults entityID="https://sp.keystone.example.org/shibboleth"
    REMOTE_USER="openstack_user">
```

Restart the `shibd` daemon after making these changes:

```
# systemctl restart shibd
```

Exchange Metadata

Once configured, the Service Provider metadata is available to download:

```
# wget https://sp.keystone.example.org/Shibboleth.sso/Metadata
```

Upload your Service Providers metadata to your Identity Provider. This step depends on your Identity Provider choice and is not covered here. If keystone is your Identity Provider you do not need to upload this file.

Continue configuring keystone

Continue configuring keystone

Mapping Combinations

Description

During the authentication process an identity provider (IdP) will present keystone with a set of user attributes about the user that is authenticating. For example, in the SAML2 flow this comes to keystone in the form of a SAML document.

The attributes are typically processed by third-party software and are presented to keystone as environment variables. The original document from the IdP is generally not available to keystone. This is how the *Shibboleth* and *Mellon* implementations work.

The mapping format described in this document maps these environment variables to a local keystone user. The mapping may also define group membership for that user and projects the user can access.

An IdP has exactly one mapping specified per protocol. Mappings themselves can be used multiple times by different combinations of IdP and protocol.

Definitions

A mapping looks as follows:

```
{
    "rules": [
        {
            "local": [
                {
                    <user>
                    [<group>]
                    [<project>]
                }
            ],
            "remote": [
                {
                    <match>
                    [<condition>]
                }
            ]
        }
    ]
}
```

- *mapping*: a JSON object containing a list of rules.
- *rules*: a property in the mapping that contains the list of rules.
- *rule*: a JSON object containing *local* and *remote* properties to define the rule. There is no explicit *rule* property.

- *local*: a JSON object containing information on what local attributes will be mapped. The mapping engine processes this using the *context* (defined below) and the result is a representation of the user from keystones perspective.
  - <*user*>: the local user that will be mapped to the federated user.
  - <*group*>: (optional) the local groups the federated user will be placed in.
  - <*projects*>: (optional) the local projects mapped to the federated user.
  - <*domain*>: (optional) the local domain mapped to the federated user, projects, and groups. Projects and groups can also override this default domain by defining a domain of their own. Moreover, if no domain is defined in this configuration, the attribute mapping schema will use the identity provider OpenStack domain.
- *remote*: a JSON object containing information on what remote attributes will be mapped.
  - <*match*>: a JSON object that tells the mapping engine what federated attribute to make available for substitution in the local object. There can be one or more of these objects in the *remote* list.
  - <*condition*>: a JSON object containing conditions that allow a rule. There can be zero or more of these objects in the *remote* list.
- *direct mapping*: the mapping engine keeps track of each match and makes them available to the local rule for substitution.
- *assertion*: data provided to keystone by the IdP to assert facts (name, groups, etc) about the authenticating user. This is an XML document when using the SAML2 protocol.
- *mapping context*: the data, represented as key-value pairs, that is used by the mapping engine to turn the *local* object into a representation of the user from keystones perspective. The mapping context contains the environment of the keystone process and any *direct mapping* values calculated when processing the *remote* list.

How Mappings Are Processed

A mapping is selected by IdP and protocol. Then keystone takes the mapping and processes each rule sequentially stopping after the first matched rule. A rule is matched when all of its conditions are met.

First keystone evaluates each condition from the rules remote property to see if the rule is a match. If it is a match, keystone saves the data captured by each of the matches from the rules remote property in an ordered list. We call these matches *direct mappings* since they can be used in the next step.

After the rule is found using the rules conditions and a list of direct mappings is stored, keystone begins processing the rules *local* property. Each object in the *local* property is collapsed into a single JSON object. For example:

```
{
    "local": [
        {
            "user": {...}
        },
        {
            "projects": [...]
        },
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

becomes:

```
{
    "local": {
        "user": {...}
        "projects": [...]
    },
}
```

when the same property exists in the local multiple times the first occurrence wins:

```
{
    "local": [
        {
            "user": {#first#}
        },
        {
            "projects": [...]
        },
        {
            "user": {#second#}
        },
    ]
}
```

becomes:

```
{
    "local": {
        "user": {#first#}
        "projects": [...]
    },
}
```

We take this JSON object and then recursively process it in order to apply the direct mappings. This is simply looking for the pattern *{#}* and substituting it with values from the direct mappings list. The index of the direct mapping starts at zero.

Mapping Rules

Mapping Engine

The mapping engine can be tested before creating a federated setup. It can be tested with the `keystone-manage mapping_engine` command:

```
$ keystone-manage mapping_engine --rules <file> --input <file>
```

Note

Although the rules file is formatted as JSON, the input file of assertion data is formatted as individual lines of key: value pairs, see *keystone-manage mapping_engine help* for details.

Mapping Conditions

Mappings support 5 different types of conditions:

`empty`: The rule is matched to all claims containing the remote attribute type. This condition does not need to be specified.

`any_one_of`: The rule is matched only if any of the specified strings appear in the remote attribute type. Condition result is boolean, not the argument that is passed as input.

`not_any_of`: The rule is not matched if any of the specified strings appear in the remote attribute type. Condition result is boolean, not the argument that is passed as input.

`blacklist`: This rule removes all groups matched from the assertion. It is not intended to be used as a way to prevent users, or groups of users, from accessing the service provider. The output from filtering through a blacklist will be all groups from the assertion that were not listed in the blacklist.

`whitelist`: This rule explicitly states which groups should be carried over from the assertion. The result is the groups present in the assertion and in the whitelist.

Note

`empty`, `blacklist` and `whitelist` are the only conditions that can be used in direct mapping ({0}, {1}, etc.)

Multiple conditions can be combined to create a single rule.

Mappings Examples

The following are all examples of mapping rule types.

empty condition

```
{
    "rules": [
        {
            "local": [
                {
                    "user": {
                        "name": "{0} {1}",
                        "email": "{2}"
                    },
                    "group": {
                        "name": "{3}",
                        "domain": {
                            "id": "0cd5e9"
                        }
                    }
                }
            ],
            "remote": [
                {
                    "type": "FirstName"
                },
                {
                    "type": "LastName"
                },
                {
                    "type": "Email"
                },
                {
                    "type": "OIDC_GROUPS"
                }
            ]
        }
    ]
}
```

Note

The numbers in braces {} are indices, they map in order. For example:

```
- Mapping to user with the name matching the value in remote attribute␣
↪FirstName
- Mapping to user with the name matching the value in remote attribute␣
↪LastName
- Mapping to user with the email matching value in remote attribute Email
```

```
- Mapping to a group(s) with the name matching the value(s) in remote␣
↪attribute OIDC_GROUPS
```

Note

If the user id and name are not specified in the mapping, the server tries to directly map `REMOTE_USER` environment variable. If this variable is also unavailable the server returns an HTTP 401 Unauthorized error.

Groups can have multiple values. Each value must be separated by a `;` Example: OIDC_GROUPS=developers;testers

other conditions

In `<other_condition>` shown below, please supply one of the following: `any_one_of`, or `not_any_of`.

```
{
    "rules": [
        {
            "local": [
                {
                    "user": {
                        "name": "{0}"
                    },
                    "group": {
                        "id": "0cd5e9"
                    }
                }
            ],
            "remote": [
                {
                    "type": "UserName"
                },
                {
                    "type": "HTTP_OIDC_GROUPIDS",
                    "<other_condition>": [
                        "HTTP_OIDC_EMAIL"
                    ]
                }
            ]
        }
    ]
}
```

In `<other_condition>` shown below, please supply one of the following: `blacklist`, or `whitelist`.

```
{
    "rules": [
        {
            "local": [
                {
                    "user": {
                        "name": "{0}"
                    }
                },
                {
                    "groups": "{1}",
                    "domain": {
                        "id": "0cd5e9"
                    }
                }
            ],
            "remote": [
                {
                    "type": "UserName"
                },
                {
                    "type": "HTTP_OIDC_GROUPIDS",
                    "<other_condition>": [
                        "me@example.com"
                    ]
                }
            ]
        }
    ]
}
```

In the above example, a whitelist can be used to only map the user into a few of the groups in their `HTTP_OIDC_GROUPIDS` remote attribute:

```
{
    "type": "HTTP_OIDC_GROUPIDS",
    "whitelist": [
        "Developers",
        "OpsTeam"
    ]
}
```

A blacklist can map the user into all groups except those matched:

```
{
    "type": "HTTP_OIDC_GROUPIDS",
    "blacklist": [
        "Finance"
    ]
}
```

Regular expressions can be used in any condition for more flexible matches:

```
{
    "type": "HTTP_OIDC_GROUPIDS",
    "whitelist": [
        ".*Team$"
    ]
}
```

When mapping into groups, either ids or names can be provided in the local section:

```
{
    "local": [
        {
            "group": {
                "id":"0cd5e9"
            }
        }
    ]
}
```

```
{
    "local": [
        {
            "group": {
                "name": "developer_group",
                "domain": {
                    "id": "abc1234"
                }
            }
        }
    ]
}
```

```
{
    "local": [
        {
            "group": {
                "name": "developer_group",
                "domain": {
                    "name": "private_cloud"
                }
            }
        }
    ]
}
```

Users can be mapped to local users that already exist in keystones identity backend by setting the `type` attribute of the user to `local` and providing the domain to which the local user belongs:

```
{
    "local": [
        {
            "user": {
                "name": "local_user",
                "type": "local",
                "domain": {
                    "name": "local_domain"
                }
            }
        }
    ]
}
```

The user is then treated as existing in the local identity backend, and the server will attempt to fetch user details (id, name, roles, groups) from the identity backend. The local user and domain are not generated dynamically, so if they do not exist in the local identity backend, authentication attempts will result in a 401 Unauthorized error.

If you omit the `type` attribute or set it to `ephemeral` or do not provide a domain, the user is deemed ephemeral and becomes a member of the identity providers domain. It will not be looked up in the local keystone backend, so all of its attributes must come from the IdP and the mapping rules.

Note

Domain `Federated` is a service domain - it cannot be listed, displayed, added or deleted. There is no need to perform any operation on it prior to federation configuration.

Output

If a mapping is valid you will receive the following output:

```
{
    "group_ids": "[<group-ids>]",
    "user":
        {
            "domain":
                {
                    "id": "Federated" or "<local-domain-id>"
                },
            "type": "ephemeral" or "local",
            "name": "<local-user-name>",
            "id": "<local-user-id>"
        },
    "group_names":
        [
            {
                "domain":
                    {
```

(continues on next page)

(continued from previous page)

```
                        "name": "<domain-name>"
                    },
                "name":
                    {
                        "name": "[<groups-names>]"
                    }
            },
            {
                "domain":
                    {
                        "name": "<domain-name>"
                    },
                "name":
                    {
                        "name": "[<groups-names>]"
                    }
            }
        ]
}
```

If the mapped user is local, mapping engine will discard further group assigning and return set of roles configured for the user.

Regular Expressions

Regular expressions can be used in a mapping by specifying the `regex` key, and setting it to `true`.

```
{
    "rules": [
        {
            "local": [
                {
                    "user": {
                        "name": "{0}"
                    },
                    "group": {
                        "name": "{1}",
                        "domain": {
                            "id": "abc1234"
                        }
                    }
                },
            ],
            "remote": [
                {
                    "type": "UserName"
                },
                {
                    "type": "HTTP_OIDC_GROUPIDS",
```

(continues on next page)

(continued from previous page)

```
                    "any_one_of": [
                        ".*@yeah.com$"
                    ]
                    "regex": true
                },
                {
                    "type": "HTTP_OIDC_GROUPIDS",
                    "whitelist": [
                        "Project.*$"
                    ],
                    "regex": true
                }
            ]
        }
    ]
}
```

This allows any user with a claim containing a key with any value in `HTTP_OIDC_GROUPIDS` to be mapped to group with id `0cd5e9`. Additionally, for every value in the `HTTP_OIDC_GROUPIDS` claim matching the string `Project.*`, the user will be assigned to the project with that name.

Condition Combinations

Combinations of mappings conditions can also be done.

`empty`, `any_one_of`, and `not_any_of` can all be used in the same rule, but cannot be repeated within the same condition. `any_one_of` and `not_any_of` are mutually exclusive within a conditions scope. So are `whitelist` and `blacklist`.

```
{
    "rules": [
        {
            "local": [
                {
                    "user": {
                        "name": "{0}"
                    },
                    "group": {
                        "id": "0cd5e9"
                    }
                },
            ],
            "remote": [
                {
                    "type": "UserName"
                },
                {
                    "type": "cn=IBM_Canada_Lab",
                    "not_any_of": [
```

(continues on next page)

(continued from previous page)

```
                        ".*@naww.com$"
                    ],
                    "regex": true
                },
                {
                    "type": "cn=IBM_USA_Lab",
                    "any_one_of": [
                        ".*@yeah.com$"
                    ]
                    "regex": true
                }
            ]
        }
    ]
}
```

As before group names and users can also be provided in the local section.

This allows any user with the following claim information to be mapped to group with id 0cd5e9.

```
{"UserName":"<any_name>@yeah.com"}
{"cn=IBM_USA_Lab":"<any_name>@yeah.com"}
{"cn=IBM_Canada_Lab":"<any_name>@yeah.com"}
```

The following claims will be mapped:

- any claim containing the key UserName.
- any claim containing key cn=IBM_Canada_Lab that doesnt have the value <any_name>@naww.com.
- any claim containing key cn=IBM_USA_Lab that has value <any_name>@yeah.com.

Multiple Rules

Multiple rules can also be utilized in a mapping.

```
{
    "rules": [
        {
            "local": [
                {
                    "user": {
                        "name": "{0}"
                    },
                    "group": {
                        "name": "non-contractors",
                        "domain": {
                            "id": "abc1234"
                        }
                    }
```

(continues on next page)

(continued from previous page)

```
                }
            ],
            "remote": [
                {
                    "type": "UserName"
                },
                {
                    "type": "orgPersonType",
                    "not_any_of": [
                        "Contractor",
                        "SubContractor"
                    ]
                }
            ]
        },
        {
            "local": [
                {
                    "user": {
                        "name": "{0}"
                    },
                    "group": {
                        "name": "contractors",
                        "domain": {
                            "id": "abc1234"
                        }
                    }
                }
            ],
            "remote": [
                {
                    "type": "UserName"
                },
                {
                    "type": "orgPersonType",
                    "any_one_of": [
                        "Contractor",
                        "SubContractor"
                    ]
                }
            ]
        }
    ]
}
```

The above assigns groups membership basing on `orgPersonType` values:

- neither `Contractor` nor `SubContractor` will belong to the `non-contractors` group.
- either `Contractor or ``SubContractor` will belong to the `contractors` group.

Rules are additive, so permissions will only be granted for the rules that succeed. All the remote condi-

tions of a rule must be valid.

When using multiple rules you can specify more than one effective user identification, but only the first match will be considered and the others ignored ordered from top to bottom.

Since rules are additive one can specify one user identification and this will also work. The best practice for multiple rules is to create a rule for just user and another rule for just groups. Below is rules example repeated but with global username mapping.

```
{
    "rules": [{
        "local": [{
            "user": {
                "id": "{0}"
            }
        }],
        "remote": [{
            "type": "UserType"
        }]
    },
    {
        "local": [{
            "group": {
                "name": "non-contractors",
                "domain": {
                    "id": "abc1234"
                }
            }
        }],
        "remote": [{
            "type": "orgPersonType",
            "not_any_of": [
                "Contractor",
                "SubContractor"
            ]
        }]
    },
    {
        "local": [{
            "group": {
                "name": "contractors",
                "domain": {
                    "id": "abc1234"
                }
            }
        }],
        "remote": [{
            "type": "orgPersonType",
            "any_one_of": [
                "Contractor",
                "SubContractor"
            ]
```

(continues on next page)

(continued from previous page)

```
        }]
    }]
}
```

Auto-Provisioning

The mapping engine has the ability to aid in the auto-provisioning of resources when a federated user authenticates for the first time. This can be achieved using a specific mapping syntax that the mapping engine can parse and ultimately make decisions on.

For example, consider the following mapping:

```
{
    "rules": [
        {
            "local": [
                {
                    "user": {
                        "name": "{0}"
                    }
                },
                {
                    "projects": [
                        {
                            "name": "Production",
                            "roles": [
                                {
                                    "name": "reader"
                                }
                            ]
                        },
                        {
                            "name": "Staging",
                            "roles": [
                                {
                                    "name": "member"
                                }
                            ]
                        },
                        {
                            "name": "Project for {0}",
                            "roles": [
                                {
                                    "name": "admin"
                                }
                            ]
                        }
                    ]
                }
```

(continues on next page)

(continued from previous page)

```
            ],
            "remote": [
                {
                    "type": "UserName"
                }
            ]
        }
    ]
}
```

The semantics of the `remote` section have not changed. The difference between this mapping and the other examples is the addition of a `projects` section within the `local` rules. The `projects` list supplies a list of projects that the federated user will be given access to. The projects will be automatically created if they dont exist when the user authenticated and the mapping engine has applied values from the assertion and mapped them into the `local` rules.

In the above example, an authenticated federated user will be granted the `reader` role on the `Production` project, `member` role on the `Staging` project, and they will have `admin` role on the `Project for jsmith`.

It is important to note the following constraints apply when auto-provisioning:

- Projects are the only resource that will be created dynamically.
- Projects will be created within the domain associated with the Identity Provider or the domain mapped via the attribute mapping (*federation_attribute_mapping_schema_version >= 2.0*).
- The `projects` section of the mapping must also contain a `roles` section.
  - Roles within the project must already exist in the deployment or domain.
- Assignments are actually created for the user which is unlike the ephemeral group memberships.

Since the creation of roles typically requires policy changes across other services in the deployment, it is expected that roles are created ahead of time. Federated authentication should also be considered idempotent if the attributes from the SAML assertion have not changed. In the example from above, if the users name is still `jsmith`, then no new projects will be created as a result of authentication.

Mappings can be created that mix `groups` and `projects` within the `local` section. The mapping shown in the example above does not contain a `groups` section in the `local` rules. This will result in the federated user having direct role assignments on the projects in the `projects` list. The following example contains `local` rules comprised of both `projects` and `groups`, which allow for direct role assignments and group memberships.

```
{
    "rules": [
        {
            "local": [
                {
                    "user": {
                        "name": "{0}"
                    }
                },
                {
```

(continues on next page)

(continued from previous page)

```
                    "projects": [
                        {
                            "name": "Marketing",
                            "roles": [
                                {
                                    "name": "member"
                                }
                            ]
                        },
                        {
                            "name": "Development project for {0}",
                            "roles": [
                                {
                                    "name": "admin"
                                }
                            ]
                        }
                    ]
                },
                {
                    "group": {
                        "name": "Finance",
                        "domain": {
                            "id": "6fe767"
                        }
                    }
                }
            ],
            "remote": [
                {
                    "type": "UserName"
                }
            ]
        }
    ]
}
```

In the above example, a federated user will receive direct role assignments on the `Marketing` project, as well as a dedicated project specific to the federated users name. In addition to that, they will also be placed in the `Finance` group and receive all role assignments that group has on projects and domains.

keystone-to-keystone

keystone-to-keystone federation also utilizes mappings, but has some differences.

An attribute file (e.g. `/etc/shibboleth/attribute-map.xml` in a Shibboleth implementation) is used to add attributes to the mapping *context*. Attributes look as follows:

```
<!-- example 1 from a K2k Shibboleth implementation -->
<Attribute name="openstack_user" id="openstack_user"/>
<Attribute name="openstack_user_domain" id="openstack_user_domain"/>
```

The service provider must contain a mapping as shown below. `openstack_user`, and `openstack_user_domain` match to the attribute names we have in the Identity Provider. It will map any user with the name `user1` or `admin` in the `openstack_user` attribute and `openstack_domain` attribute `default` to a group with id `abc1234`.

```
{
    "rules": [
        {
            "local": [
                {
                    "group": {
                        "id": "abc1234"
                    }
                }
            ],
            "remote": [
                {
                    "type": "openstack_user",
                    "any_one_of": [
                        "user1",
                        "admin"
                    ]
                },
                {
                    "type":"openstack_user_domain",
                    "any_one_of": [
                        "Default"
                    ]
                }
            ]
        }
    ]
}
```

A keystone users groups can also be mapped to groups in the service provider. For example, with the following attributes declared in Shibboleths attributes file:

```
<!-- example 2 from a K2k Shibboleth implementation -->
<Attribute name="openstack_user" id="openstack_user"/>
<Attribute name="openstack_groups" id="openstack_groups"/>
```

Then the following mapping can be used to map the users group membership from the keystone IdP to groups in the keystone SP:

```
{
    "rules": [
        {
            "local":
            [
                {
                    "user":
                        {
                            "name": "{0}"
                        }
                },
                {
                    "groups": "{1}"
                }
            ],
            "remote":
            [
                {
                    "type": "openstack_user"
                },
                {
                    "type": "openstack_groups"
                }
            ]
        }
    ]
}
```

`openstack_user`, and `openstack_groups` will be matched by service provider to the attribute names we have in the Identity Provider. It will take the `openstack_user` attribute and finds in the assertion then inserts it directly in the mapping. The identity provider will set the value of `openstack_groups` by group name and domain name to which the user belongs in the Idp. Suppose the user belongs to group1 in domain Default in the IdP then it will map to a group with the same name and same domains name in the SP.

The possible attributes that can be used in a mapping are *openstack_user*, *openstack_user_domain*, *openstack_roles*, *openstack_project*, *openstack_project_domain* and *openstack_groups*.

8.7.4 Using external authentication with Keystone

When Keystone is executed in a web server like Apache HTTPD, it is possible to have the web server also handle authentication. This enables support for additional methods of authentication that are not provided by the identity store backend and the authentication plugins that Keystone supports.

Having the web server handle authentication is not exclusive, and both Keystone and the web server can provide different methods of authentication at the same time. For example, the web server can provide support for X.509 or Kerberos authentication, while Keystone provides support for password authentication (with SQL or an identity store as the backend).

When the web server authenticates a user, it sets environment variables, usually `REMOTE_USER`, which can be used in the underlying application. Keystone can be configured to use these environment variables to determine the identity of the user.

Configuration

In order to activate the external authentication mechanism for Identity API v3, the `external` method must be in the list of enabled authentication methods. By default it is enabled, so if you dont want to use external authentication, remove it from the `methods` option in the `auth` section.

To configure the plugin that should be used set the `external` option again in the `auth` section. There are two external authentication method plugins provided by Keystone:

- `DefaultDomain`: This plugin wont take into account the domain information that the external authentication method may pass down to Keystone and will always use the configured default domain. The `REMOTE_USER` variable is the username. This is the default if no plugin is given.
- `Domain`: This plugin expects that the `REMOTE_DOMAIN` variable contains the domain for the user. If this variable is not present, the configured default domain will be used. The `REMOTE_USER` variable is the username.

Caution

You should disable the external auth method if you are currently using federation. External auth and federation both use the `REMOTE_USER` variable. Since both the mapped and external plugin are being invoked to validate attributes in the request environment, it can cause conflicts.

For example, imagine there are two distinct users with the same username *foo*, one in the *Default* domain while the other is in the *BAR* domain. The external Federation modules (i.e. mod_shib) sets the `REMOTE_USER` attribute to *foo*. The external auth module also tries to set the `REMOTE_USER` attribute to *foo* for the *Default* domain. The federated mapping engine maps the incoming identity to *foo* in the *BAR* domain. This results in user_id conflict since both are using different user_ids to set *foo* in the *Default* domain and the *BAR* domain.

To disable this, simply remove *external* from the *methods* option in *keystone.conf*:

```
methods = external,password,token,oauth1
```

Using HTTPD authentication

Web servers like Apache HTTP support many methods of authentication. Keystone can profit from this feature and let the authentication be done in the web server, that will pass down the authenticated user to Keystone using the `REMOTE_USER` environment variable. This user must exist in advance in the identity backend to get a token from the controller.

To use this method, Keystone should be running on HTTPD.

X.509 example

The following snippet for the Apache conf will authenticate the user based on a valid X.509 certificate from a known CA:

```
<VirtualHost _default_:5000>
    SSLEngine on
    SSLCertificateFile    /etc/ssl/certs/ssl.cert
    SSLCertificateKeyFile /etc/ssl/private/ssl.key

    SSLCACertificatePath /etc/ssl/allowed_cas
    SSLCARevocationPath  /etc/ssl/allowed_cas
    SSLUserName          SSL_CLIENT_S_DN_CN
    SSLVerifyClient      require
    SSLVerifyDepth       10

    (...)
</VirtualHost>
```

8.7.5 Configuring Keystone for Tokenless Authorization

Definitions

- *X.509 Tokenless Authorization*: Provides a means to authorize client operations within Keystone by using an X.509 SSL client certificate without having to issue a token.

  This feature is designed to reduce the complexity of user token validation in Keystone `auth_token` middleware by eliminating the need for service user token for authentication and authorization. Therefore, theres no need to having to create and maintain a service user account for the sole purpose of user token validation. Furthermore, this feature improves efficiency by avoiding service user token handling (i.e. request, cache, and renewal). By not having to deal with service user credentials in the configuration files, deployers are relieved of the burden of having to protect the server user passwords throughout the deployment lifecycle. This feature also improve security by using X.509 certificate instead of password for authentication.

  For details, please refer to the specs Tokenless Authorization with X.509 Client SSL Certificate

- *Public Key Infrastructure or PKI*: a system which utilize public key cryptography to achieve authentication, authorization, confidentiality, integrity, non-repudiation. In this system, the identities are represented by public key certificates. Public key certificate handling is governed by the X.509 standard.

  See Public Key Infrastructure and X.509 for more information.

- *X.509 Certificate*: a time bound digital identity, which is certified or digitally signed by its issuer using cryptographic means as defined by the X.509 standard. It contains information which can be used to uniquely identify its owner. For example, the owner of the certificate is identified by the `Subject` attribute while the issuer is identified by `Issuer` attribute.

  In operation, certificates are usually stored in Privacy-Enhanced Mail (PEM) format.

  Heres an example of what a certificate typically contains:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4098 (0x1002)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: DC = com, DC = somedemo, O = openstack, OU = keystone, CN␣
↪= Intermediate CA
        Validity
            Not Before: Jul  5 18:42:01 2019 GMT
            Not After : Jul  2 18:42:01 2029 GMT
        Subject: DC = com, DC = somedemo, O = Default, OU = keystone, CN␣
↪= glance
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:cf:35:8b:cd:4f:17:28:38:25:f7:e2:ac:ce:4e:
                    d7:05:74:2f:99:04:f8:c2:13:14:50:18:70:d6:b0:
                    53:62:15:60:59:99:90:47:e2:7e:bf:ca:30:4a:18:
                    f5:b8:29:1e:cc:d4:b8:49:9c:4a:aa:d9:10:b9:d7:
                    9f:55:85:cf:e3:44:d2:3c:95:42:5a:b0:53:3e:49:
                    9d:6b:b2:a0:9f:72:9d:76:96:55:8b:ee:c4:71:46:
                    ab:bd:12:71:42:a0:60:29:7a:66:16:e1:fd:03:17:
                    af:a3:c7:26:c3:c3:8b:a7:f9:c0:22:08:2d:e4:5c:
                    07:e1:44:58:c1:b1:88:ae:45:5e:03:10:bb:b4:c2:
                    42:52:da:4e:b5:1b:d6:6f:49:db:a4:5f:8f:e5:79:
                    9f:73:c2:37:de:99:a7:4d:6f:cb:b5:f9:7e:97:e0:
                    77:c8:40:21:40:ef:ab:d3:55:72:37:6c:28:0f:bd:
                    37:8c:3a:9c:e9:a0:21:6b:63:3f:7a:dd:1b:2c:90:
                    07:37:66:86:66:36:ef:21:bb:43:df:d5:37:a9:fa:
                    4b:74:9a:7c:4b:cd:8b:9d:3b:af:6d:50:fe:c9:0a:
                    25:35:c5:1d:40:35:1d:1f:f9:10:fd:b6:5c:45:11:
                    bb:67:11:81:3f:ed:d6:27:04:98:8f:9e:99:a1:c8:
                    c1:2d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Client, S/MIME
            Netscape Comment:
                OpenSSL Generated Client Certificate
            X509v3 Subject Key Identifier:
                ␣
↪EE:38:FB:60:65:CD:81:CE:B2:01:E3:A5:99:1B:34:6C:1A:74:97:BB
            X509v3 Authority Key Identifier:
                ␣
↪keyid:64:17:77:31:00:F2:ED:90:9A:A8:1D:B5:7D:75:06:03:B5:FD:B9:C0

            X509v3 Key Usage: critical
```

(continues on next page)

(continued from previous page)

```
            Digital Signature, Non Repudiation, Key Encipherment
        X509v3 Extended Key Usage:
            TLS Web Client Authentication, E-mail Protection
    Signature Algorithm: sha256WithRSAEncryption
         82:8b:17:c6:f4:63:eb:8d:69:03:7a:bf:54:7f:37:02:eb:94:
         ef:57:fd:27:8f:f8:67:e9:0e:3b:0a:40:66:11:68:e6:04:1a:
         8a:da:47:ed:83:eb:54:34:3b:5b:70:18:cf:62:e2:6d:7c:74:
         4c:cf:14:b3:a9:70:b2:68:ed:19:19:71:6f:7d:87:22:38:8d:
         83:c6:59:15:74:19:5b:a2:64:6f:b9:9a:81:3d:0a:67:58:d1:
         e2:b2:9b:9b:8f:60:7a:8c:0e:61:d9:d7:04:63:cc:58:af:36:
         a4:61:86:44:1c:64:e2:9b:bd:f3:21:87:dd:18:81:80:af:0f:
         d6:4c:9f:ae:0f:01:e0:0e:38:4d:5d:71:da:0b:11:39:bd:c3:
         5d:0c:db:14:ca:bf:7f:07:37:c9:36:bd:22:a5:73:c6:e1:13:
         53:15:de:ac:4a:4b:dc:48:90:47:06:fa:d4:d2:5d:c6:d2:d4:
         3f:0f:49:0f:27:de:21:b0:bd:a3:92:c3:cb:69:b6:8d:94:e1:
         e3:40:b4:80:c7:e6:e2:df:0a:94:52:d1:16:41:0f:bc:29:a8:
         93:40:1b:77:28:a3:f2:cb:3c:7f:bb:ae:a6:0e:b3:01:78:09:
         d3:2b:cf:2f:47:83:91:36:37:43:34:6e:80:2b:81:10:27:95:
         95:ae:1e:93:42:94:a6:23:b8:07:c0:0f:38:23:70:b0:8e:79:
         14:cd:72:8a:90:bf:77:ad:74:3c:23:9e:67:5d:0e:26:15:6e:
         20:95:6d:d0:89:be:a3:6c:4a:13:1d:39:fb:21:e3:9c:9f:f3:
         ff:15:da:0a:28:29:4e:f4:7f:5e:0f:70:84:80:7c:09:5a:1c:
         f4:ac:c9:1b:9d:38:43:dd:27:00:95:ef:14:a0:57:3e:26:0b:
         d8:bb:40:d6:1f:91:92:f0:4e:5d:93:1c:b7:3d:bd:83:ef:79:
         ee:47:ca:61:04:00:e6:39:05:ab:f0:cd:47:e9:25:c8:3a:4c:
         e5:62:9f:aa:8a:ba:ea:46:10:ef:bd:1e:24:5f:0c:89:8a:21:
         bb:9d:c7:73:0f:b9:b5:72:1f:1f:1b:5b:ff:3a:cb:d8:51:bc:
         bb:9a:40:91:a9:d5:fe:95:ac:73:a5:12:6a:b2:e3:b1:b2:7d:
         bf:e7:db:cd:9f:24:63:6e:27:cf:d8:82:d9:ac:d8:c9:88:ea:
         4f:1c:ae:7d:b7:c7:81:b2:1c:f8:6b:6b:85:3b:f2:14:cb:c7:
         61:81:ad:64:e7:d9:90:a3:ea:69:7e:26:7a:0a:29:7b:1b:2a:
         e0:38:f7:58:d1:90:82:44:01:ab:05:fd:68:0c:ab:9e:c6:94:
         76:34:46:8b:66:bb:02:07
```

See public key certificate for more information.

- *Issuer*: the issuer of a X.509 certificate. It is also known as Certificate Authority (CA) or Certification Authority. Issuer is typically represented in RFC 2253 format. Throughout this document, `issuer`, `issuer DN`, `CA`, and `trusted issuer` are used interchangeably.

Prerequisites

This feature requires Keystone API proxy SSL terminator to validate the incoming X.509 SSL client certificate and pass the certificate information (i.e. subject DN, issuer DN, etc) to the Keystone application as part of the request environment. At the time of this writing the feature has been tested with either HAProxy or Apache as Keystone API proxy SSL terminator only.

The rest of this document required readers to familiar with:

- Public Key Infrastructure (PKI) and certificate management

- SSL with client authentication, or commonly known as two-way SSL
- Public Key Infrastructure (PKI) and certificate management
- Apache SSL configuration
- HAProxy SSL configuration

Configuring this feature requires OpenSSL Command Line Tool (CLI). Please refer to the respective OS installation guide on how to install it.

Keystone Configuration

This feature utilizes Keystone federation capability to determine the authorization associated with the incoming X.509 SSL client certificate by mapping the certificate attributes to a Keystone identity. Therefore, the direct issuer or trusted Certification Authority (CA) of the client certificate is the remote Identity Provider (IDP), and the hexadecimal output of the SHA256 hash of the issuer distinguished name (DN) is used as the IDP ID.

Note

Client certificate issuer DN may be formatted differently depending on the SSL terminator. For example, Apache mod_ssl may use RFC 2253 while HAProxy may use the old format. The old format is used by applications that linked with an older version of OpenSSL where the string representation of the distinguished name has not yet become a de facto standard. For more information on the old formation, please see the nameopt in the OpenSSL CLI manual. Therefore, it is critically important to keep the format consistent throughout the configuration as Keystone does exact string match when comparing certificate attributes.

How to obtain trusted issuer DN

If SSL terminates at either HAProxy or Apache, the client certificate issuer DN can be obtained by using the OpenSSL CLI.

Since version 2.3.11, Apache mod_ssl by default uses RFC 2253 when handling certificate distinguished names. However, deployer have the option to use the old format by configuring the LegacyDNStringFormat option.

HAProxy, on the other hand, only supports the old format.

To obtain issuer DN in RFC 2253 format:

```
$ openssl x509 -issuer -noout -in client_cert.pem -nameopt rfc2253 | sed 's/^\
↪s*issuer=//'
```

To obtain issuer DN in old format:

```
$ openssl x509 -issuer -noout -in client_cert.pem -nameopt compat | sed 's/^\
↪s*issuer=//'
```

How to calculate the IDP ID from trusted issuer DN

The hexadecimal output of the SHA256 hash of the trusted issuer DN is being used as the Identity Provider ID in Keystone. It can be obtained using OpenSSL CLI.

To calculate the IDP ID for issuer DN in RFC 2253 format:

```
$ openssl x509 -issuer -noout -in client_cert.pem -nameopt rfc2253 | tr -d '\n
↪' | sed 's/^\s*issuer=//' | openssl dgst -sha256 -hex | awk '{print $2}'
```

To calculate the IDP ID for issuer DN in old format:

```
$ openssl x509 -issuer -noout -in client_cert.pem -nameopt compat | tr -d '\n
↪' | sed 's/^\s*issuer=//' | openssl dgst -sha256 -hex | awk '{print $2}'
```

Keystone Configuration File Changes

The following options in the `tokenless_auth` section of the Keystone configuration file *keystone.conf* are used to enable the X.509 tokenless authorization feature:

- `trusted_issuer` - A list of trusted issuers for the X.509 SSL client certificates. More specifically the list of trusted issuer DNs mentioned in the *How to obtain trusted issuer DN* section above. The format of the trusted issuer DNs must match exactly with what the SSL terminator passed into the request environment. For example, if SSL terminates in Apache mod_ssl, then the issuer DN should be in RFC 2253 format. Whereas if SSL terminates in HAProxy, then the issuer DN is expected to be in the old format. This is a multi-string list option. The absence of any trusted issuers means the X.509 tokenless authorization feature is effectively disabled.
- `protocol` - The protocol name for the X.509 tokenless authorization along with the option *issuer_attribute* below can look up its corresponding mapping. It defaults to `x509`.
- `issuer_attribute` - The issuer attribute that is served as an IdP ID for the X.509 tokenless authorization along with the protocol to look up its corresponding mapping. It is the environment variable in the WSGI environment that references to the Issuer of the client certificate. It defaults to `SSL_CLIENT_I_DN`.

This is a sample configuration for two *trusted_issuer* and a *protocol* set to `x509`.

```
[tokenless_auth]
trusted_issuer = emailAddress=admin@foosigner.com,CN=Foo Signer,OU=eng,O=abc,
↪L=San Jose,ST=California,C=US
trusted_issuer = emailAddress=admin@openstack.com,CN=OpenStack Cert Signer,
↪OU=keystone,O=openstack,L=Sunnyvale,ST=California,C=US
protocol = x509
```

Setup Mapping

Like federation, X.509 tokenless authorization also utilizes the mapping mechanism to formulate an identity. The identity provider must correspond to the issuer of the X.509 SSL client certificate. The protocol for the given identity is `x509` by default, but can be configurable.

Create an Identity Provider (IDP)

As mentioned, the Identity Provider ID is the hexadecimal output of the SHA256 hash of the issuer distinguished name (DN).

Note

If there are multiple trusted issuers, there must be multiple IDP created, one for each trusted issuer.

To create an IDP for a given trusted issuer, follow the instructions in the *How to calculate the IDP ID from trusted issuer DN* section to calculate the IDP ID. Then use OpenStack CLI to create the IDP. i.e.

```
$ openstack identity provider create --description 'IDP foo' <IDP ID>
```

Create a Map

A mapping needs to be created to map the `Subject DN` in the client certificate as a user to yield a valid local user if the users `type` defined as `local` in the mapping. For example, the client certificate has `Subject DN` as `CN=alex,OU=eng,O=nice-network,L=Sunnyvale, ST=California,C=US`, in the following examples, `user_name` will be mapped to"alex" and `domain_name` will be mapped to `nice-network`. And it has users `type` set to `local`. If users `type` is not defined, it defaults to `ephemeral`.

Please refer to mod_ssl for the detailed mapping attributes.

```
[
    {
        "local": [
            {
                "user": {
                    "name": "{0}",
                    "domain": {
                        "name": "{1}"
                    },
                    "type": "local"
                }
            }
        ],
        "remote": [
            {
                "type": "SSL_CLIENT_S_DN_CN",
                "whitelist": ["glance", "nova", "swift", "neutron"]
```

(continues on next page)

(continued from previous page)

```
            },
            {
                 "type": "SSL_CLIENT_S_DN_O",
                 "whitelist": ["Default"]
            }
        ]
    }
]
```

When users `type` is not defined or set to `ephemeral`, the mapped user does not have to be a valid local user but the mapping must yield at least one valid local group. For example:

```
[
    {
        "local": [
            {
                "user": {
                    "name": "{0}",
                    "type": "ephemeral"
                },
                "group": {
                    "domain": {
                        "name": "{1}"
                    },
                    "name": "openstack_services"
                }
            }
        ],
        "remote": [
            {
                "type": "SSL_CLIENT_S_DN_CN",
                "whitelist": ["glance", "nova", "swift", "neutron"]
            },
            {
                 "type": "SSL_CLIENT_S_DN_O",
                 "whitelist": ["Default"]
            }
        ]
    }
]
```

Note

The above mapping assume openstack_services group already exist and have the proper role assignments (i.e. allow token validation) If not, it will need to be created.

To create a mapping using OpenStack CLI, assuming the mapping is saved into a file `x509_tokenless_mapping.json`:

```
$ openstack mapping create --rules x509_tokenless_mapping.json x509_tokenless
```

Note

The mapping ID is arbitrary and it can be any string as opposed to IDP ID.

Create a Protocol

The name of the protocol must be the same as the one specified by the `protocol` option in `tokenless_auth` section of the Keystone configuration file. The protocol name is user designed and it can be any name as opposed to IDP ID.

A protocol name and an IDP ID will uniquely identify a mapping.

To create a protocol using OpenStack CLI:

```
$ openstack federation protocol create --identity-provider <IDP ID>
  --mapping x509_tokenless x509
```

Note

If there are multiple trusted issuers, there must be multiple protocol created, one for each IDP. All IDP can share a same mapping but the combination of IDP ID and protocol must be unique.

SSL Terminator Configuration

Apache Configuration

If SSL terminates at Apache mod_ssl, Apache must be configured to handle two-way SSL and pass the SSL certificate information to the Keystone application as part of the request environment.

The Client authentication attribute `SSLVerifyClient` should be set as `optional` to allow other token authentication methods and attribute `SSLOptions` needs to set as `+StdEnvVars` to allow certificate attributes to be passed. For example,

```
<VirtualHost *:443>
    WSGIScriptAlias / /var/www/cgi-bin/keystone/main
    ErrorLog /var/log/apache2/keystone.log
    CustomLog /var/log/apache2/access.log combined
    SSLEngine on
    SSLCertificateFile    /etc/apache2/ssl/apache.cer
    SSLCertificateKeyFile /etc/apache2/ssl/apache.key
    SSLCACertificatePath /etc/apache2/capath
    SSLOptions +StdEnvVars
    SSLVerifyClient optional
</VirtualHost>
```

HAProxy and Apache Configuration

If SSL terminates at HAProxy and Apache is the API proxy for the Keystone application, HAProxy must configured to handle two-way SSL and convey the SSL certificate information via the request headers. Apache in turn will need to bring those request headers into the request environment.

Heres an example on how to configure HAProxy to handle two-way SSL and pass the SSL certificate information via the request headers.

```
frontend http-frontend
    mode http
    option forwardfor
    bind 10.1.1.1:5000 ssl crt /etc/keystone/ssl/keystone.pem ca-file /etc/
↪keystone/ssl/ca.pem verify optional

    reqadd X-Forwarded-Proto:\ https if { ssl_fc }
    http-request set-header X-SSL                   %[ssl_fc]
    http-request set-header X-SSL-Client-Verify     %[ssl_c_verify]
    http-request set-header X-SSL-Client-SHA1       %{+Q}[ssl_c_sha1]
    http-request set-header X-SSL-Client-DN         %{+Q}[ssl_c_s_dn]
    http-request set-header X-SSL-Client-CN         %{+Q}[ssl_c_s_dn(cn)]
    http-request set-header X-SSL-Client-O          %{+Q}[ssl_c_s_dn(o)]
    http-request set-header X-SSL-Issuer            %{+Q}[ssl_c_i_dn]
    http-request set-header X-SSL-Issuer-CN         %{+Q}[ssl_c_i_dn(cn)]
```

When the request gets to the Apache Keystone API Proxy, Apache will need to bring those SSL headers into the request environment. Heres an example on how to configure Apache to achieve that.

```
<VirtualHost 192.168.0.10:5000>
    WSGIScriptAlias / /var/www/cgi-bin/keystone/main

    # Bring the needed SSL certificate attributes from HAProxy into the
    # request environment
    SetEnvIf X-SSL-Issuer "^(.*)$" SSL_CLIENT_I_DN=$0
    SetEnvIf X-SSL-Issuer-CN "^(.*)$" SSL_CLIENT_I_DN_CN=$0
    SetEnvIf X-SSL-Client-CN "^(.*)$" SSL_CLIENT_S_DN_CN=$0
    SetEnvIf X-SSL-Client-O "^(.*)$" SSL_CLIENT_S_DN_O=$0
</VirtualHost>
```

Setup `auth_token` middleware

In order to use `auth_token` middleware as the service client for X.509 tokenless authorization, both configurable options and scope information will need to be setup.

Configurable Options

The following configurable options in `auth_token` middleware should set to the correct values:

- `auth_type` - Must set to `v3tokenlessauth`.
- `certfile` - Set to the full path of the certificate file.
- `keyfile` - Set to the full path of the private key file.
- `cafile` - Set to the full path of the trusted CA certificate file.
- `project_name` or `project_id` - set to the scoped project.
- `project_domain_name` or `project_domain_id` - if `project_name` is specified.

Heres an example of `auth_token` middleware configuration using X.509 tokenless authorization for user token validation.

```
[keystone_authtoken]
memcached_servers = localhost:11211
cafile = /etc/keystone/ca.pem
project_domain_name = Default
project_name = service
auth_url = https://192.168.0.10/identity/v3
auth_type = v3tokenlessauth
certfile = /etc/glance/certs/glance.pem
keyfile = /etc/glance/private/glance_private_key.pem
```

8.7.6 OAuth1 1.0a

The OAuth 1.0a feature provides the ability for Identity users to delegate roles to third party consumers via the OAuth 1.0a specification.

To enable OAuth1:

1. Add the oauth1 driver to the `[oauth1]` section in `keystone.conf`. For example:

```
[oauth1]
driver = sql
```

2. Add the `oauth1` authentication method to the `[auth]` section in `keystone.conf`:

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

3. If deploying under Apache httpd with `mod_wsgi`, set the *WSGIPassAuthorization* to allow the OAuth Authorization headers to pass through *mod_wsgi*. For example, add the following to the keystone virtual host file:

```
WSGIPassAuthorization On
```

See API Specification for OAuth 1.0a for the details of API definition.

8.8 OAuth2.0 Client Credentials Grant Flow

8.8.1 Overview

OAuth2.0 Client Credentials Grant based on RFC6749 is implemented as an extension of Keystone. This extension uses the application credentials as its back-end because they have some similar features. Users can use `application_credentials_id` and `application_credentials_secret` as client credentials to obtain the OAuth2.0 access token. The access token can then be used to access the protected resources of the OpenStack API using Keystonemiddleware that supports receiving access tokens in the Authorization header. See the Identity API reference for more information on generating OAuth2.0 access token.

8.8.2 Guide

Enable Keystone identity server to support OAuth2.0 Client Credentials Grant by the following steps in this guide. In this example, `keystone.host` is the domain name used by the Keystone identity server.

Warning

It is strongly recommended that HTTPS be enabled in Keystone when using OAuth2.0 Client Credentials. See *Configure HTTPS in Identity Service* for details. According to RFC6749 , HTTPS **must** be enabled in the authorization server since requests include sensitive information, e.g., a client secret, in plain text. Note that you might have to enable both HTTP and HTTPS as some other OpenStack services or third-party applications dont use OAuth2.0 and need HTTP for the authentication with the Keystone identity server.

Enable application credentials authentication

Due to the design of the current implementation, the application credentials must be enabled in Keystone as it is used for the management of OAuth2.0 client credentials.

1. Modify `keystone.conf` to support application credentials authentication.

```
stack@oauth2-0-server:/$ vi /etc/keystone/keystone.conf
[auth]
methods = external,password,token,application_credential
```

2. Restart Keystone service so that the modified configuration information takes effect.

```
stack@oauth2-0-server:/$ sudo systemctl restart devstack@keystone.service
```

Try to access the Keystone APIs

At last, try to access the Keystone APIs to confirm that the server is working properly.

1. Create OAuth2.0 client credentials through the application credentials API.

```
stack@oauth2-0-server:/$ openstack application credential create sample_001
+--------------+--------------------------------------------------------------------
↪--------------------------+
| Field        | Value                                                              ␣
↪                          |
+--------------+--------------------------------------------------------------------
↪--------------------------+
| description  | None                                                               ␣
↪                          |
| expires_at   | None                                                               ␣
↪                          |
| id           | a7850381222a4e2cb595664dfd57d083                                   ␣
↪                          |
| name         | sample_001                                                         ␣
↪                          |
| project_id   | 2b90a96668694041a640a2ef84be6de7                                   ␣
↪                          |
| roles        | admin reader member                                                ␣
↪                          |
| secret       | GVm33KC6AqpDZj_
↪ZzKhZClDqnCpNDMNh66Mvait8Dxw7Kc8kwVj7ImkwnRWvovs437f2aftbW46wEMtH0cyBQA |
| system       | None                                                               ␣
↪                          |
| unrestricted | False                                                              ␣
↪                          |
| user_id      | 0b8426bb83d944bc8d0fe4c3b9a3f635                                   ␣
↪                          |
+--------------+--------------------------------------------------------------------
↪--------------------------+
```

2. Obtain oauth2.0 access tokens through the Basic HTTP authentication with OAuth2.0 client credentials.

```
stack@oauth2-0-server:/$ curl -sik -u "
↪$a7850381222a4e2cb595664dfd57d083:GVm33KC6AqpDZj_
↪ZzKhZClDqnCpNDMNh66Mvait8Dxw7Kc8kwVj7ImkwnRWvovs437f2aftbW46wEMtH0cyBQA" \
-X POST https://keystone.host/identity/v3/OS-OAUTH2/token
-H "application/x-www-form-urlencoded" -d "grant_type=client_credentials"
HTTP/1.1 200 OK
Date: Tue, 01 Mar 2022 00:56:59 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Type: application/json
Content-Length: 264
Vary: X-Auth-Token
x-openstack-request-id: req-a8358f51-2e0f-45a7-bb1e-7d29c6a793f4
```

(continues on next page)

(continued from previous page)

```
Connection: close

{"access_token":
↪"gAAAAABhi1cMynG89h8t6TJrxNiZuNzjcIUIxNctoVfuqTw7BpUedLKxjPymClVEnj9GhIT5u2mpjaJATlEAtaa3I
↪t8jk_fV-mqo2IUlsmTPTnMwkcjh5FSHQVRdqvDxgY3nSqLA_Hfv-
↪zPmjS5KWX3hmyDE5YWO1ztX6QNVQb4wTPyNL1-7I","expires_in":3600,"token_type":
↪"Bearer"}
```

8.9 Configure HTTPS in Identity Service

The following part describes steps to enable both HTTP and HTTPS with a self-signed certificate.

1. Generate an RSA private key.

```
stack@oauth2-0-server:/$ openssl genrsa -out keystone.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................+++++
.......................+++++
e is 65537 (0x010001)
```

2. Create a certificate signing request.

```
stack@oauth2-0-server:/$ openssl req -new -key keystone.key -out keystone.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:keystone.host
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3. Generate a self-signed certificate.

```
stack@oauth2-0-server:/$ openssl x509 -req -days 365 -in keystone.csr \
-signkey keystone.key -out keystone.host.crt
Signature ok
```

(continues on next page)

(continued from previous page)

```
subject=C = , ST = , L = , O = , OU = , CN = keystone.host, emailAddress =
Getting Private key
```

4. Append the configuration file for setting the HTTPS port service under the directory `/etc/apache2/sites-enabled/`.

```
stack@oauth2-0-server:/$ sudo ln -s \
/etc/apache2/sites-available/000-default.conf \
/etc/apache2/sites-enabled/000-default.conf
```

5. Modify the apache configuration file and add proxy rules to implement HTTPS support for the Keystone service.

```
stack@oauth2-0-server:/$ vi 000-default.conf
<VirtualHost *:443>
DocumentRoot /var/www/html
SSLCertificateFile /etc/ssl/certs/keystone.host.crt
SSLCertificateKeyFile /etc/ssl/certs/keystone.key
SSLEngine on
SSLProtocol  all -SSLv2 -SSLv3
SSLCipherSuite ECDH:AESGCM:HIGH:!RC4:!DH:!MD5:!aNULL:!eNULL
SSLHonorCipherOrder on
ProxyPass "/identity" "unix:/var/run/uwsgi/keystone-wsgi-public.socket|uwsgi:/
↪/uwsgi-uds-keystone-wsgi-public" retry=0
</VirtualHost>
```

6. Restart apache service so that the modified configuration information takes effect.

```
stack@oauth2-0-server:/$ systemctl restart apache2.service
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ===
Authentication is required to restart 'apache2.service'.
Authenticating as: Ubuntu (ubuntu)
Password:
==== AUTHENTICATION COMPLETE ===
```

CHAPTER

NINE

KEYSTONE CONFIGURATION OPTIONS

This section provides a list of all possible options and sample files for keystone configuration.

9.1 API Configuration options

9.1.1 Configuration

The Identity service is configured in the `/etc/keystone/keystone.conf` file.

The following tables provide a comprehensive list of the Identity service options.

DEFAULT

admin_token

Type
string

Default
<None>

Using this feature is *NOT* recommended. Instead, use the *keystone-manage bootstrap* command. The value of this option is treated as a shared secret that can be used to bootstrap Keystone through the API. This token does not represent a user (it has no identity), and carries no explicit authorization (it effectively bypasses most authorization checks). If set to *None*, the value is ignored and the *admin_token* middleware is effectively disabled.

public_endpoint

Type
URI

Default
<None>

The base public endpoint URL for Keystone that is advertised to clients (NOTE: this does NOT affect how Keystone listens for connections). Defaults to the base host URL of the request. For example, if keystone receives a request to *http://server:5000/v3/users*, then this will option will be automatically treated as *http://server:5000*. You should only need to set option if either the value of the base URL contains a path that keystone does not automatically infer (*/prefix/v3*), or if the endpoint should be found on a different host.

`max_project_tree_depth`

Type
integer

Default
5

Maximum depth of the project hierarchy, excluding the project acting as a domain at the top of the hierarchy. WARNING: Setting it to a large value may adversely impact performance.

`max_param_size`

Type
integer

Default
64

Limit the sizes of user & project ID/names.

`max_token_size`

Type
integer

Default
255

Similar to *[DEFAULT] max_param_size*, but provides an exception for token values. With Fernet tokens, this can be set as low as 255.

`list_limit`

Type
integer

Default
<None>

The maximum number of entities that will be returned in a collection. This global limit may be then overridden for a specific driver, by specifying a list_limit in the appropriate section (for example, *[assignment]*). No limit is set by default. In larger deployments, it is recommended that you set this to a reasonable number to prevent operations like listing all users and projects from placing an unnecessary load on the system.

`strict_password_check`

Type
boolean

Default
False

If set to true, strict password length checking is performed for password manipulation. If a password exceeds the maximum length, the operation will fail with an HTTP 403 Forbidden error. If set to false, passwords are automatically truncated to the maximum length.

`insecure_debug`

Type
boolean

Default
`False`

If set to true, then the server will return information in HTTP responses that may allow an unauthenticated or authenticated user to get more information than normal, such as additional details about why authentication failed. This may be useful for debugging but is insecure.

`default_publisher_id`

Type
string

Default
<None>

Default *publisher_id* for outgoing notifications. If left undefined, Keystone will default to using the servers host name.

`notification_format`

Type
string

Default
`cadf`

Valid Values
basic, cadf

Define the notification format for identity service events. A *basic* notification only has information about the resource being operated on. A *cadf* notification has the same information, as well as information about the initiator of the event. The *cadf* option is entirely backwards compatible with the *basic* option, but is fully CADF-compliant, and is recommended for auditing use cases.

`notification_opt_out`

Type
multi-valued

Default
`identity.authenticate.success`

Default
`identity.authenticate.pending`

You can reduce the number of notifications keystone emits by explicitly opting out. Keystone will not emit notifications that match the patterns expressed in this list. Values are expected to be in the form of *identity.<resource_type>.<operation>*. By default, all notifications related to authentication are automatically suppressed. This field can be set multiple times in order to opt-out of multiple notification topics. For example, the following suppresses notifications describing user creation or successful authentication events: notification_opt_out=identity.user.create notification_opt_out=identity.authenticate.success

`rpc_conn_pool_size`

Type
integer

Default
`30`

Minimum Value
1

Size of RPC connection pool.

Table 1: Deprecated Variations

Group	Name
DEFAULT	rpc_conn_pool_size

`conn_pool_min_size`

Type
integer

Default
`2`

The pool size limit for connections expiration policy

`conn_pool_ttl`

Type
integer

Default
`1200`

The time-to-live in sec of idle connections in the pool

`executor_thread_pool_size`

Type
integer

Default
`64`

Size of executor thread pool when executor is threading or eventlet.

Table 2: Deprecated Variations

Group	Name
DEFAULT	rpc_thread_pool_size

`rpc_response_timeout`

Type
integer

Default
`60`

Seconds to wait for a response from a call.

`transport_url`

Type
string

Default
`rabbit://`

The network address and optional user credentials for connecting to the messaging backend, in URL format. The expected format is:

driver://[user:pass@]host:port[,[userN:passN@]hostN:portN]/virtual_host?query

Example: rabbit://rabbitmq:password@127.0.0.1:5672//

For full details on the fields in the URL see the documentation of oslo_messaging.TransportURL at https://docs.openstack.org/oslo.messaging/latest/reference/transport.html

`control_exchange`

Type
string

Default
`keystone`

The default exchange under which topics are scoped. May be overridden by an exchange name specified in the transport_url option.

`rpc_ping_enabled`

Type
boolean

Default
`False`

Add an endpoint to answer to ping calls. Endpoint is named oslo_rpc_server_ping

`debug`

Type
boolean

Default
`False`

Mutable
This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

`log_config_append`

Type
string

Default
<None>

Mutable
This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 3: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

`log_date_format`

Type
string

Default
`%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

`log_file`

Type
string

Default
<None>

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 4: Deprecated Variations

Group	Name
DEFAULT	logfile

`log_dir`

Type
string

Default
<None>

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 5: Deprecated Variations

Group	Name
DEFAULT	logdir

`watch_log_file`

Type
boolean

Default
`False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
This function is known to have bene broken for long time, and depends on the unmaintained library

`use_syslog`

Type
boolean

Default
`False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

`use_journal`

Type
boolean

Default
`False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

`syslog_log_facility`

Type
string

Default
`LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

`use_json`

Type
boolean

Default
`False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

`use_stderr`

Type
boolean

Default
`False`

Log output to standard error. This option is ignored if log_config_append is set.

`use_eventlog`

Type
boolean

Default
`False`

Log output to Windows Event Log.

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
Windows support is no longer maintained.

`log_color`

Type
boolean

Default
`False`

(Optional) Set the color key according to log levels. This option takes effect only when logging to stderr or stdout is used. This option is ignored if log_config_append is set.

`log_rotate_interval`

Type
integer

Default
1

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is set to interval.

`log_rotate_interval_type`

Type
string

Default
`days`

Valid Values
Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

`max_logfile_count`

Type
integer

Default
`30`

Maximum number of rotated log files.

`max_logfile_size_mb`

Type
integer

Default
`200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

`log_rotation_type`

Type
string

Default
none

Valid Values
interval, size, none

Log rotation type.

Possible values

interval
Rotate logs at predefined time intervals.

size
Rotate logs once they reach a predefined size.

none
Do not rotate log files.

`logging_context_format_string`

Type
string

Default
```
%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s
[%(global_request_id)s %(request_id)s %(user_identity)s]
%(instance)s%(message)s
```

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

`logging_default_format_string`

Type
string

Default
```
%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-]
%(instance)s%(message)s
```

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

`logging_debug_format_suffix`

Type
string

Default
```
%(funcName)s %(pathname)s:%(lineno)d
```

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

`logging_exception_prefix`

Type
string

Default
```
%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s
%(instance)s
```

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

`logging_user_identity_format`

Type
string

Default
```
%(user)s %(project)s %(domain)s %(system_scope)s
%(user_domain)s %(project_domain)s
```

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

`default_log_levels`

Type
list

Default
`INFO`

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

`publish_errors`

Type
boolean

Default
`False`

Enables or disables publication of error events.

`instance_format`

Type
string

Default
`"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

`instance_uuid_format`

Type
string

Default
`"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

`rate_limit_interval`

Type
integer

Default
`0`

Interval, number of seconds, of log rate limiting.

`rate_limit_burst`

Type
integer

Default
`0`

Maximum number of logged messages per rate_limit_interval.

`rate_limit_except_level`

Type
string

Default
`CRITICAL`

Valid Values
CRITICAL, ERROR, INFO, WARNING, DEBUG,

Log level name used by rate limiting. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

`fatal_deprecations`

Type
boolean

Default
`False`

Enables or disables fatal status of deprecations.

application_credential

`driver`

Type
string

Default
`sql`

Entry point for the application credential backend driver in the *keystone.application_credential* namespace. Keystone only provides a *sql* driver, so there is no reason to change this unless you are providing a custom entry point.

`caching`

Type
boolean

Default
`True`

Toggle for application credential caching. This has no effect unless global caching is enabled.

`cache_time`

Type
integer

Default
<None>

Time to cache application credential data in seconds. This has no effect unless global caching is enabled.

`user_limit`

Type
integer

Default
`-1`

Maximum number of application credentials a user is permitted to create. A value of -1 means unlimited. If a limit is not set, users are permitted to create application credentials at will, which could lead to bloat in the keystone database or open keystone to a DoS attack.

assignment

`driver`

Type
string

Default
`sql`

Entry point for the assignment backend driver (where role assignments are stored) in the *keystone.assignment* namespace. Only a SQL driver is supplied by keystone itself. Unless you are writing proprietary drivers for keystone, you do not need to set this option.

`prohibited_implied_role`

Type
list

Default
`['admin']`

A list of role names which are prohibited from being an implied role.

auth

`methods`

Type
list

Default
```
['external', 'password', 'token', 'oauth1', 'mapped',
'application_credential']
```

Allowed authentication methods. Note: You should disable the *external* auth method if you are currently using federation. External auth and federation both use the REMOTE_USER variable. Since both the mapped and external plugin are being invoked to validate attributes in the request environment, it can cause conflicts.

`password`

Type
string

Default
<None>

Entry point for the password auth plugin module in the *keystone.auth.password* namespace. You do not need to set this unless you are overriding keystones own password authentication plugin.

`token`

Type
string

Default
<None>

Entry point for the token auth plugin module in the *keystone.auth.token* namespace. You do not need to set this unless you are overriding keystones own token authentication plugin.

`external`

Type
string

Default
<None>

Entry point for the external (*REMOTE_USER*) auth plugin module in the *keystone.auth.external* namespace. Supplied drivers are *DefaultDomain* and *Domain*. The default driver is *DefaultDomain*, which assumes that all users identified by the username specified to keystone in the *REMOTE_USER* variable exist within the context of the default domain. The *Domain* option expects an additional environment variable be presented to keystone, *REMOTE_DOMAIN*, containing the domain name of the *REMOTE_USER* (if *REMOTE_DOMAIN* is not set, then the default domain will be used instead). You do not need to set this unless you are taking advantage of external authentication, where the application server (such as Apache) is handling authentication instead of keystone.

`oauth1`

Type
string

Default
<None>

Entry point for the OAuth 1.0a auth plugin module in the *keystone.auth.oauth1* namespace. You do not need to set this unless you are overriding keystones own *oauth1* authentication plugin.

`mapped`

Type
string

Default
<None>

Entry point for the mapped auth plugin module in the *keystone.auth.mapped* namespace. You do not need to set this unless you are overriding keystones own *mapped* authentication plugin.

`application_credential`

Type
string

Default
<None>

Entry point for the application_credential auth plugin module in the *keystone.auth.application_credential* namespace. You do not need to set this unless you are overriding keystones own *application_credential* authentication plugin.

cache

config_prefix

Type
string

Default
`cache.oslo`

Prefix for building the configuration dictionary for the cache region. This should not need to be changed unless there is another dogpile.cache region with the same configuration name.

expiration_time

Type
integer

Default
`600`

Default TTL, in seconds, for any cached item in the dogpile.cache region. This applies to any cached method that doesnt have an explicit cache expiration time defined for it.

backend

Type
string

Default
`dogpile.cache.null`

Valid Values
oslo_cache.memcache_pool, oslo_cache.dict, oslo_cache.mongo, oslo_cache.etcd3gw, dogpile.cache.pymemcache, dogpile.cache.memcached, dogpile.cache.pylibmc, dogpile.cache.bmemcached, dogpile.cache.dbm, dogpile.cache.redis, dogpile.cache.redis_sentinel, dogpile.cache.memory, dogpile.cache.memory_pickle, dogpile.cache.null

Cache backend module. For eventlet-based or environments with hundreds of threaded servers, Memcache with pooling (oslo_cache.memcache_pool) is recommended. For environments with less than 100 threaded servers, Memcached (dogpile.cache.memcached) or Redis (dogpile.cache.redis) is recommended. Test environments with a single instance of the server can use the dogpile.cache.memory backend.

backend_argument

Type
multi-valued

Default
''

Arguments supplied to the backend module. Specify this option once per argument to be passed to the dogpile.cache backend. Example format: <argname>:<value>.

`proxies`

Type
list

Default
`[]`

Proxy classes to import that will affect the way the dogpile.cache backend functions. See the dogpile.cache documentation on changing-backend-behavior.

`enabled`

Type
boolean

Default
`True`

Global toggle for caching.

`debug_cache_backend`

Type
boolean

Default
`False`

Extra debugging from the cache backend (cache keys, get/set/delete/etc calls). This is only really useful if you need to see the specific cache-backend get/set/delete calls with the keys/values. Typically this should be left set to false.

`memcache_servers`

Type
list

Default
`['localhost:11211']`

Memcache servers in the format of host:port. This is used by backends dependent on Memcached.If `dogpile.cache.memcached` or `oslo_cache.memcache_pool` is used and a given host refer to an IPv6 or a given domain refer to IPv6 then you should prefix the given address with the address family (`inet6`) (e.g `inet6[::1]:11211`, `inet6:[fd12:3456:789a:1::1]:11211`, `inet6:[controller-0.internalapi]:11211`). If the address family is not given then these backends will use the default `inet` address family which corresponds to IPv4

`memcache_dead_retry`

Type
integer

Default
`300`

Number of seconds memcached server is considered dead before it is tried again. (dogpile.cache.memcache and oslo_cache.memcache_pool backends only).

`memcache_socket_timeout`

Type
floating point

Default
`1.0`

Timeout in seconds for every call to a server. (dogpile.cache.memcache and oslo_cache.memcache_pool backends only).

`memcache_pool_maxsize`

Type
integer

Default
`10`

Max total number of open connections to every memcached server. (oslo_cache.memcache_pool backend only).

`memcache_pool_unused_timeout`

Type
integer

Default
`60`

Number of seconds a connection to memcached is held unused in the pool before it is closed. (oslo_cache.memcache_pool backend only).

`memcache_pool_connection_get_timeout`

Type
integer

Default
`10`

Number of seconds that an operation will wait to get a memcache client connection.

`memcache_pool_flush_on_reconnect`

Type
boolean

Default
`False`

Global toggle if memcache will be flushed on reconnect. (oslo_cache.memcache_pool backend only).

memcache_sasl_enabled

Type
boolean

Default
`False`

Enable the SASL(Simple Authentication and SecurityLayer) if the SASL_enable is true, else disable.

memcache_username

Type
string

Default
<None>

the user name for the memcached which SASL enabled

memcache_password

Type
string

Default
<None>

the password for the memcached which SASL enabled

redis_server

Type
string

Default
`localhost:6379`

Redis server in the format of host:port

redis_db

Type
integer

Default
`0`

Minimum Value
0

Database id in Redis server

redis_username

Type
string

Default
<None>

the user name for redis

`redis_password`

Type
string

Default
<None>

the password for redis

`redis_sentinels`

Type
list

Default
`['localhost:26379']`

Redis sentinel servers in the format of host:port

`redis_socket_timeout`

Type
floating point

Default
`1.0`

Timeout in seconds for every call to a server. (dogpile.cache.redis and dogpile.cache.redis_sentinel backends only).

`redis_sentinel_service_name`

Type
string

Default
`mymaster`

Service name of the redis sentinel cluster.

`tls_enabled`

Type
boolean

Default
`False`

Global toggle for TLS usage when communicating with the caching servers. Currently supported by `dogpile.cache.bmemcache`, `dogpile.cache.pymemcache`, `oslo_cache.memcache_pool`, `dogpile.cache.redis` and `dogpile.cache.redis_sentinel`.

`tls_cafile`

Type
string

Default
<None>

Path to a file of concatenated CA certificates in PEM format necessary to establish the caching servers authenticity. If tls_enabled is False, this option is ignored.

`tls_certfile`

Type
string

Default
<None>

Path to a single file in PEM format containing the clients certificate as well as any number of CA certificates needed to establish the certificates authenticity. This file is only required when client side authentication is necessary. If tls_enabled is False, this option is ignored.

`tls_keyfile`

Type
string

Default
<None>

Path to a single file containing the clients private key in. Otherwise the private key will be taken from the file specified in tls_certfile. If tls_enabled is False, this option is ignored.

`tls_allowed_ciphers`

Type
string

Default
<None>

Set the available ciphers for sockets created with the TLS context. It should be a string in the OpenSSL cipher list format. If not specified, all OpenSSL enabled ciphers will be available. Currently supported by `dogpile.cache.bmemcache`, `dogpile.cache.pymemcache` and `oslo_cache.memcache_pool`.

`enable_socket_keepalive`

Type
boolean

Default
`False`

Global toggle for the socket keepalive of dogpiles pymemcache backend

`socket_keepalive_idle`

Type
integer

Default
1

Minimum Value
0

The time (in seconds) the connection needs to remain idle before TCP starts sending keepalive probes. Should be a positive integer most greater than zero.

`socket_keepalive_interval`

Type
integer

Default
1

Minimum Value
0

The time (in seconds) between individual keepalive probes. Should be a positive integer greater than zero.

`socket_keepalive_count`

Type
integer

Default
1

Minimum Value
0

The maximum number of keepalive probes TCP should send before dropping the connection. Should be a positive integer greater than zero.

`enable_retry_client`

Type
boolean

Default
`False`

Enable retry client mechanisms to handle failure. Those mechanisms can be used to wrap all kind of pymemcache clients. The wrapper allows you to define how many attempts to make and how long to wait between attemots.

`retry_attempts`

Type
integer

Default
2

Minimum Value
1

Number of times to attempt an action before failing.

`retry_delay`

Type
floating point

Default
`0`

Number of seconds to sleep between each attempt.

`hashclient_retry_attempts`

Type
integer

Default
`2`

Minimum Value
1

Amount of times a client should be tried before it is marked dead and removed from the pool in the HashClients internal mechanisms.

`hashclient_retry_delay`

Type
floating point

Default
`1`

Time in seconds that should pass between retry attempts in the HashClients internal mechanisms.

`dead_timeout`

Type
floating point

Default
`60`

Time in seconds before attempting to add a node back in the pool in the HashClients internal mechanisms.

`enforce_fips_mode`

Type
boolean

Default
`False`

Global toggle for enforcing the OpenSSL FIPS mode. This feature requires Python support. This is available in Python 3.9 in all environments and may have been backported to older Python versions on select environments. If the Python executable used does not support OpenSSL FIPS mode, an exception will be raised. Currently supported by `dogpile.cache.bmemcache`, `dogpile.cache.pymemcache` and `oslo_cache.memcache_pool`.

catalog

template_file

Type
string

Default
`default_catalog.templates`

Absolute path to the file used for the templated catalog backend. This option is only used if the *[catalog] driver* is set to *templated*.

driver

Type
string

Default
`sql`

Entry point for the catalog driver in the *keystone.catalog* namespace. Keystone provides a *sql* option (which supports basic CRUD operations through SQL), a *templated* option (which loads the catalog from a templated catalog file on disk), and a *endpoint_filter.sql* option (which supports arbitrary service catalogs per project).

caching

Type
boolean

Default
`True`

Toggle for catalog caching. This has no effect unless global caching is enabled. In a typical deployment, there is no reason to disable this.

cache_time

Type
integer

Default
<None>

Time to cache catalog data (in seconds). This has no effect unless global and catalog caching are both enabled. Catalog data (services, endpoints, etc.) typically does not change frequently, and so a longer duration than the global default may be desirable.

list_limit

Type
integer

Default
<None>

Maximum number of entities that will be returned in a catalog collection. There is typically no reason to set this, as it would be unusual for a deployment to have enough services or endpoints to exceed a reasonable limit.

cors

allowed_origin

Type
list

Default
<None>

Indicate whether this resource may be shared with the domain received in the requests origin header. Format: <protocol>://<host>[:<port>], no trailing slash. Example: https://horizon.example.com

allow_credentials

Type
boolean

Default
True

Indicate that the actual request can include user credentials

expose_headers

Type
list

Default
['X-Auth-Token', 'X-Openstack-Request-Id', 'X-Subject-Token', 'Openstack-Auth-Receipt']

Indicate which headers are safe to expose to the API. Defaults to HTTP Simple Headers.

max_age

Type
integer

Default
3600

Maximum cache age of CORS preflight requests.

allow_methods

Type
list

Default
['GET', 'PUT', 'POST', 'DELETE', 'PATCH']

Indicate which methods can be used during the actual request.

allow_headers

Type
list

Default

```
['X-Auth-Token', 'X-Openstack-Request-Id', 'X-Subject-Token',
'X-Project-Id', 'X-Project-Name', 'X-Project-Domain-Id',
'X-Project-Domain-Name', 'X-Domain-Id', 'X-Domain-Name',
'Openstack-Auth-Receipt']
```

Indicate which header field names may be used during the actual request.

credential

driver

Type
string

Default
`sql`

Entry point for the credential backend driver in the *keystone.credential* namespace. Keystone only provides a *sql* driver, so theres no reason to change this unless you are providing a custom entry point.

provider

Type
string

Default
`fernet`

Entry point for credential encryption and decryption operations in the *keystone.credential.provider* namespace. Keystone only provides a *fernet* driver, so theres no reason to change this unless you are providing a custom entry point to encrypt and decrypt credentials.

key_repository

Type
string

Default
`/etc/keystone/credential-keys/`

Directory containing Fernet keys used to encrypt and decrypt credentials stored in the credential backend. Fernet keys used to encrypt credentials have no relationship to Fernet keys used to encrypt Fernet tokens. Both sets of keys should be managed separately and require different rotation policies. Do not share this repository with the repository used to manage keys for Fernet tokens.

caching

Type
boolean

Default
`True`

Toggle for caching only on retrieval of user credentials. This has no effect unless global caching is enabled.

`cache_time`

Type
integer

Default
<None>

Time to cache credential data in seconds. This has no effect unless global caching is enabled.

`auth_ttl`

Type
integer

Default
15

The length of time in minutes for which a signed EC2 or S3 token request is valid from the timestamp contained in the token request.

`user_limit`

Type
integer

Default
-1

Maximum number of credentials a user is permitted to create. A value of -1 means unlimited. If a limit is not set, users are permitted to create credentials at will, which could lead to bloat in the keystone database or open keystone to a DoS attack.

database

`sqlite_synchronous`

Type
boolean

Default
True

If True, SQLite uses synchronous mode.

`backend`

Type
string

Default
sqlalchemy

The back end to use for the database.

`connection`

Type
string

Default
<None>

The SQLAlchemy connection string to use to connect to the database.

`slave_connection`

Type
string

Default
<None>

The SQLAlchemy connection string to use to connect to the slave database.

`mysql_sql_mode`

Type
string

Default
TRADITIONAL

The SQL mode to be used for MySQL sessions. This option, including the default, overrides any server-set SQL mode. To use whatever SQL mode is set by the server configuration, set this to no value. Example: mysql_sql_mode=

`mysql_wsrep_sync_wait`

Type
integer

Default
<None>

For Galera only, configure wsrep_sync_wait causality checks on new connections. Default is None, meaning dont configure any setting.

`connection_recycle_time`

Type
integer

Default
3600

Connections which have been present in the connection pool longer than this number of seconds will be replaced with a new one the next time they are checked out from the pool.

`max_pool_size`

Type
integer

Default
5

Maximum number of SQL connections to keep open in a pool. Setting a value of 0 indicates no limit.

`max_retries`

Type
integer

Default
`10`

Maximum number of database connection retries during startup. Set to -1 to specify an infinite retry count.

`retry_interval`

Type
integer

Default
`10`

Interval between retries of opening a SQL connection.

`max_overflow`

Type
integer

Default
`50`

If set, use this value for max_overflow with SQLAlchemy.

`connection_debug`

Type
integer

Default
`0`

Minimum Value
0

Maximum Value
100

Verbosity of SQL debugging information: 0=None, 100=Everything.

`connection_trace`

Type
boolean

Default
`False`

Add Python stack traces to SQL as comment strings.

`pool_timeout`

Type
integer

Default
<None>

If set, use this value for pool_timeout with SQLAlchemy.

`use_db_reconnect`

Type
boolean

Default
`False`

Enable the experimental use of database reconnect on connection lost.

`db_retry_interval`

Type
integer

Default
`1`

Seconds between retries of a database transaction.

`db_inc_retry_interval`

Type
boolean

Default
`True`

If True, increases the interval between retries of a database operation up to db_max_retry_interval.

`db_max_retry_interval`

Type
integer

Default
`10`

If db_inc_retry_interval is set, the maximum seconds between retries of a database operation.

`db_max_retries`

Type
integer

Default
`20`

Maximum retries in case of connection error or deadlock error before error is raised. Set to -1 to specify an infinite retry count.

`connection_parameters`

Type
string

Default
''

Optional URL parameters to append onto the connection URL at connect time; specify as param1=value1&param2=value2&

domain_config

driver

Type
string

Default
`sql`

Entry point for the domain-specific configuration driver in the *keystone.resource.domain_config* namespace. Only a *sql* option is provided by keystone, so there is no reason to set this unless you are providing a custom entry point.

caching

Type
boolean

Default
`True`

Toggle for caching of the domain-specific configuration backend. This has no effect unless global caching is enabled. There is normally no reason to disable this.

cache_time

Type
integer

Default
`300`

Time-to-live (TTL, in seconds) to cache domain-specific configuration data. This has no effect unless *[domain_config] caching* is enabled.

additional_whitelisted_options

Type
unknown type

Default
<None>

Additional whitelisted domain-specific options for out-of-tree drivers. This is a dictonary of lists with the key being the group name and value a list of group options.

additional_sensitive_options

Type
unknown type

Default
<None>

Additional sensitive domain-specific options for out-of-tree drivers. This is a dictonary of lists with the key being the group name and value a list of group options.

endpoint_filter

driver

Type
string

Default
sql

Entry point for the endpoint filter driver in the *keystone.endpoint_filter* namespace. Only a *sql* option is provided by keystone, so there is no reason to set this unless you are providing a custom entry point.

return_all_endpoints_if_no_filter

Type
boolean

Default
True

This controls keystones behavior if the configured endpoint filters do not result in any endpoints for a user + project pair (and therefore a potentially empty service catalog). If set to true, keystone will return the entire service catalog. If set to false, keystone will return an empty service catalog.

endpoint_policy

driver

Type
string

Default
sql

Entry point for the endpoint policy driver in the *keystone.endpoint_policy* namespace. Only a *sql* driver is provided by keystone, so there is no reason to set this unless you are providing a custom entry point.

federation

driver

Type
string

Default
`sql`

Entry point for the federation backend driver in the *keystone.federation* namespace. Keystone only provides a *sql* driver, so there is no reason to set this option unless you are providing a custom entry point.

assertion_prefix

Type
string

Default
`''`

Prefix to use when filtering environment variable names for federated assertions. Matched variables are passed into the federated mapping engine.

remote_id_attribute

Type
string

Default
`<None>`

Default value for all protocols to be used to obtain the entity ID of the Identity Provider from the environment. For *mod_shib*, this would be *Shib-Identity-Provider*. For *mod_auth_openidc*, this could be *HTTP_OIDC_ISS*. For *mod_auth_mellon*, this could be *MELLON_IDP*. This can be overridden on a per-protocol basis by providing a *remote_id_attribute* to the federation protocol using the API.

federated_domain_name

Type
string

Default
`Federated`

An arbitrary domain name that is reserved to allow federated ephemeral users to have a domain concept. Note that an admin will not be able to create a domain with this name or update an existing domain to this name. You are not advised to change this value unless you really have to.

Warning

This option is deprecated for removal since T. Its value may be silently ignored in the future.

Reason
This option has been superseded by ephemeral users existing in the domain of their identity provider.

`trusted_dashboard`

Type
multi-valued

Default
''

A list of trusted dashboard hosts. Before accepting a Single Sign-On request to return a token, the origin host must be a member of this list. This configuration option may be repeated for multiple values. You must set this in order to use web-based SSO flows. For example: trusted_dashboard=https://acme.example.com/auth/websso trusted_dashboard=https://beta.example.com/auth/websso

`sso_callback_template`

Type
string

Default
`/etc/keystone/sso_callback_template.html`

Absolute path to an HTML file used as a Single Sign-On callback handler. This page is expected to redirect the user from keystone back to a trusted dashboard host, by form encoding a token in a POST request. Keystones default value should be sufficient for most deployments.

`caching`

Type
boolean

Default
`True`

Toggle for federation caching. This has no effect unless global caching is enabled. There is typically no reason to disable this.

`default_authorization_ttl`

Type
integer

Default
`0`

Default time in minutes for the validity of group memberships carried over from a mapping. Default is 0, which means disabled.

`attribute_mapping_default_schema_version`

Type
string

Default
`1.0`

The attribute mapping default schema version to be used, if the attribute mapping being registered does not have a schema version. One must bear in mind that changing this value will have no effect on attribute mappings that were previously registered when another default value was applied.

Once registered, one needs to update the attribute mapping schema via the update API to be able to change an attribute mapping schema version.

fernet_receipts

key_repository

Type
string

Default
`/etc/keystone/fernet-keys/`

Directory containing Fernet receipt keys. This directory must exist before using *keystone-manage fernet_setup* for the first time, must be writable by the user running *keystone-manage fernet_setup* or *keystone-manage fernet_rotate*, and of course must be readable by keystones server process. The repository may contain keys in one of three states: a single staged key (always index 0) used for receipt validation, a single primary key (always the highest index) used for receipt creation and validation, and any number of secondary keys (all other index values) used for receipt validation. With multiple keystone nodes, each node must share the same key repository contents, with the exception of the staged key (index 0). It is safe to run *keystone-manage fernet_rotate* once on any one node to promote a staged key (index 0) to be the new primary (incremented from the previous highest index), and produce a new staged key (a new key with index 0); the resulting repository can then be atomically replicated to other nodes without any risk of race conditions (for example, it is safe to run *keystone-manage fernet_rotate* on host A, wait any amount of time, create a tarball of the directory on host A, unpack it on host B to a temporary location, and atomically move (*mv*) the directory into place on host B). Running *keystone-manage fernet_rotate twice* on a key repository without syncing other nodes will result in receipts that can not be validated by all nodes.

max_active_keys

Type
integer

Default
3

Minimum Value
1

This controls how many keys are held in rotation by *keystone-manage fernet_rotate* before they are discarded. The default value of 3 means that keystone will maintain one staged key (always index 0), one primary key (the highest numerical index), and one secondary key (every other index). Increasing this value means that additional secondary keys will be kept in the rotation.

fernet_tokens

key_repository

Type
string

Default
`/etc/keystone/fernet-keys/`

Directory containing Fernet token keys. This directory must exist before using *keystone-manage fernet_setup* for the first time, must be writable by the user running *keystone-manage fernet_setup* or *keystone-manage fernet_rotate*, and of course must be readable by keystones server process. The repository may contain keys in one of three states: a single staged key (always index 0) used for token validation, a single primary key (always the highest index) used for token creation and validation, and any number of secondary keys (all other index values) used for token validation. With multiple keystone nodes, each node must share the same key repository contents, with the exception of the staged key (index 0). It is safe to run *keystone-manage fernet_rotate* once on any one node to promote a staged key (index 0) to be the new primary (incremented from the previous highest index), and produce a new staged key (a new key with index 0); the resulting repository can then be atomically replicated to other nodes without any risk of race conditions (for example, it is safe to run *keystone-manage fernet_rotate* on host A, wait any amount of time, create a tarball of the directory on host A, unpack it on host B to a temporary location, and atomically move (*mv*) the directory into place on host B). Running *keystone-manage fernet_rotate twice* on a key repository without syncing other nodes will result in tokens that can not be validated by all nodes.

max_active_keys

Type
integer

Default
3

Minimum Value
1

This controls how many keys are held in rotation by *keystone-manage fernet_rotate* before they are discarded. The default value of 3 means that keystone will maintain one staged key (always index 0), one primary key (the highest numerical index), and one secondary key (every other index). Increasing this value means that additional secondary keys will be kept in the rotation.

healthcheck

path

Type
string

Default
`/healthcheck`

The path to respond to healtcheck requests on.

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

`detailed`

Type
boolean

Default
`False`

Show more detailed information as part of the response. Security note: Enabling this option may expose sensitive details about the service being monitored. Be sure to verify that it will not violate your security policies.

`backends`

Type
list

Default
`[]`

Additional backends that can perform health checks and report that information back as part of a request.

`allowed_source_ranges`

Type
list

Default
`[]`

A list of network addresses to limit source ip allowed to access healthcheck information. Any request from ip outside of these network addresses are ignored.

`ignore_proxied_requests`

Type
boolean

Default
`False`

Ignore requests with proxy headers.

`disable_by_file_path`

Type
string

Default
<None>

Check the presence of a file to determine if an application is running on a port. Used by DisableByFileHealthcheck plugin.

`disable_by_file_paths`

Type
list

Default
`[]`

Check the presence of a file based on a port to determine if an application is running on a port. Expects a port:path list of strings. Used by DisableByFilesPortsHealthcheck plugin.

`enable_by_file_paths`

Type
list

Default
`[]`

Check the presence of files. Used by EnableByFilesHealthcheck plugin.

identity

`default_domain_id`

Type
string

Default
`default`

This references the domain to use for all Identity API v2 requests (which are not aware of domains). A domain with this ID can optionally be created for you by *keystone-manage bootstrap*. The domain referenced by this ID cannot be deleted on the v3 API, to prevent accidentally breaking the v2 API. There is nothing special about this domain, other than the fact that it must exist to order to maintain support for your v2 clients. There is typically no reason to change this value.

`domain_specific_drivers_enabled`

Type
boolean

Default
`False`

A subset (or all) of domains can have their own identity driver, each with their own partial configuration options, stored in either the resource backend or in a file in a domain configuration directory (depending on the setting of *[identity] domain_configurations_from_database*). Only values specific to the domain need to be specified in this manner. This feature is disabled by default, but may be enabled by default in a future release; set to true to enable.

`domain_configurations_from_database`

Type
boolean

Default
`False`

By default, domain-specific configuration data is read from files in the directory identified by *[identity] domain_config_dir*. Enabling this configuration option allows you to instead manage domain-specific configurations through the API, which are then persisted in the backend (typically, a SQL database), rather than using configuration files on disk.

`domain_config_dir`

Type
string

Default
`/etc/keystone/domains`

Absolute path where keystone should locate domain-specific *[identity]* configuration files. This option has no effect unless *[identity] domain_specific_drivers_enabled* is set to true. There is typically no reason to change this value.

`driver`

Type
string

Default
`sql`

Entry point for the identity backend driver in the *keystone.identity* namespace. Keystone provides a *sql* and *ldap* driver. This option is also used as the default driver selection (along with the other configuration variables in this section) in the event that *[identity] domain_specific_drivers_enabled* is enabled, but no applicable domain-specific configuration is defined for the domain in question. Unless your deployment primarily relies on *ldap* AND is not using domain-specific configuration, you should typically leave this set to *sql*.

`caching`

Type
boolean

Default
`True`

Toggle for identity caching. This has no effect unless global caching is enabled. There is typically no reason to disable this.

`cache_time`

Type
integer

Default
`600`

Time to cache identity data (in seconds). This has no effect unless global and identity caching are enabled.

`max_password_length`

Type
integer

Default
4096

Maximum Value
4096

Maximum allowed length for user passwords. Decrease this value to improve performance. Changing this value does not effect existing passwords. This value can also be overridden by certain hashing algorithms maximum allowed length which takes precedence over the configured value. The bcrypt max_password_length is 72 bytes.

`list_limit`

Type
integer

Default
<None>

Maximum number of entities that will be returned in an identity collection.

`password_hash_algorithm`

Type
string

Default
bcrypt

Valid Values
bcrypt, bcrypt_sha256, scrypt, pbkdf2_sha512

The password hashing algorithm to use for passwords stored within keystone.

`password_hash_rounds`

Type
integer

Default
<None>

This option represents a trade off between security and performance. Higher values lead to slower performance, but higher security. Changing this option will only affect newly created passwords as existing password hashes already have a fixed number of rounds applied, so it is safe to tune this option in a running cluster. The default for bcrypt is 12, must be between 4 and 31, inclusive. The default for scrypt is 16, must be within *range(1,32)*. The default for pbkdf_sha512 is 60000, must be within *range(1,1«32)* WARNING: If using scrypt, increasing this value increases BOTH time AND memory requirements to hash a password.

`scrypt_block_size`

Type
integer

Default
<None>

Optional block size to pass to scrypt hash function (the *r* parameter). Useful for tuning scrypt to optimal performance for your CPU architecture. This option is only used when the *password_hash_algorithm* option is set to *scrypt*. Defaults to 8.

`scrypt_parallelism`

Type
integer

Default
<None>

Optional parallelism to pass to scrypt hash function (the *p* parameter). This option is only used when the *password_hash_algorithm* option is set to *scrypt*. Defaults to 1.

`salt_bytesize`

Type
integer

Default
<None>

Minimum Value
0

Maximum Value
96

Number of bytes to use in scrypt and pbkfd2_sha512 hashing salt. Default for scrypt is 16 bytes. Default for pbkfd2_sha512 is 16 bytes. Limited to a maximum of 96 bytes due to the size of the column used to store password hashes.

identity_mapping

`driver`

Type
string

Default
`sql`

Entry point for the identity mapping backend driver in the *keystone.identity.id_mapping* namespace. Keystone only provides a *sql* driver, so there is no reason to change this unless you are providing a custom entry point.

`generator`

Type
string

Default
`sha256`

Entry point for the public ID generator for user and group entities in the *keystone.identity.id_generator* namespace. The Keystone identity mapper only supports generators

that produce 64 bytes or less. Keystone only provides a *sha256* entry point, so there is no reason to change this value unless youre providing a custom entry point.

`backward_compatible_ids`

Type
boolean

Default
`True`

The format of user and group IDs changed in Juno for backends that do not generate UUIDs (for example, LDAP), with keystone providing a hash mapping to the underlying attribute in LDAP. By default this mapping is disabled, which ensures that existing IDs will not change. Even when the mapping is enabled by using domain-specific drivers (*[identity] domain_specific_drivers_enabled*), any users and groups from the default domain being handled by LDAP will still not be mapped to ensure their IDs remain backward compatible. Setting this value to false will enable the new mapping for all backends, including the default LDAP driver. It is only guaranteed to be safe to enable this option if you do not already have assignments for users and groups from the default LDAP domain, and you consider it to be acceptable for Keystone to provide the different IDs to clients than it did previously (existing IDs in the API will suddenly change). Typically this means that the only time you can set this value to false is when configuring a fresh installation, although that is the recommended value.

jwt_tokens

`jws_public_key_repository`

Type
string

Default
`/etc/keystone/jws-keys/public`

Directory containing public keys for validating JWS token signatures. This directory must exist in order for keystones server process to start. It must also be readable by keystones server process. It must contain at least one public key that corresponds to a private key in *keystone.conf [jwt_tokens] jws_private_key_repository*. This option is only applicable in deployments issuing JWS tokens and setting *keystone.conf [token] provider = jws*.

`jws_private_key_repository`

Type
string

Default
`/etc/keystone/jws-keys/private`

Directory containing private keys for signing JWS tokens. This directory must exist in order for keystones server process to start. It must also be readable by keystones server process. It must contain at least one private key that corresponds to a public key in *keystone.conf [jwt_tokens] jws_public_key_repository*. In the event there are multiple private keys in this directory, keystone will use a key named *private.pem* to sign tokens. In the future, keystone may support the ability to sign tokens with multiple private keys. For now, only a key named *private.pem* within this directory is required to issue JWS tokens. This option is only applicable in deployments issuing JWS tokens and setting *keystone.conf [token] provider = jws*.

ldap

url

Type
string

Default
`ldap://localhost`

URL(s) for connecting to the LDAP server. Multiple LDAP URLs may be specified as a comma separated string. The first URL to successfully bind is used for the connection.

randomize_urls

Type
boolean

Default
`False`

Randomize the order of URLs in each keystone process. This makes the failure behavior more gradual, since if the first server is down, a process/thread will wait for the specified timeout before attempting a connection to a server further down the list. This defaults to False, for backward compatibility.

user

Type
string

Default
<None>

The user name of the administrator bind DN to use when querying the LDAP server, if your LDAP server requires it.

password

Type
string

Default
<None>

The password of the administrator bind DN to use when querying the LDAP server, if your LDAP server requires it.

suffix

Type
string

Default
`cn=example,cn=com`

The default LDAP server suffix to use, if a DN is not defined via either *[ldap] user_tree_dn* or *[ldap] group_tree_dn*.

`query_scope`

Type
string

Default
`one`

Valid Values
one, sub

The search scope which defines how deep to search within the search base. A value of *one* (representing *oneLevel* or *singleLevel*) indicates a search of objects immediately below to the base object, but does not include the base object itself. A value of *sub* (representing *subtree* or *wholeSubtree*) indicates a search of both the base object itself and the entire subtree below it.

`page_size`

Type
integer

Default
`0`

Minimum Value
0

Defines the maximum number of results per page that keystone should request from the LDAP server when listing objects. A value of zero (*0*) disables paging.

`alias_dereferencing`

Type
string

Default
`default`

Valid Values
never, searching, always, finding, default

The LDAP dereferencing option to use for queries involving aliases. A value of *default* falls back to using default dereferencing behavior configured by your *ldap.conf*. A value of *never* prevents aliases from being dereferenced at all. A value of *searching* dereferences aliases only after name resolution. A value of *finding* dereferences aliases only during name resolution. A value of *always* dereferences aliases in all cases.

`debug_level`

Type
integer

Default
<None>

Minimum Value
-1

Sets the LDAP debugging level for LDAP calls. A value of 0 means that debugging is not enabled. This value is a bitmask, consult your LDAP documentation for possible values.

`chase_referrals`

Type
boolean

Default
<None>

Sets keystones referral chasing behavior across directory partitions. If left unset, the systems default behavior will be used.

`user_tree_dn`

Type
string

Default
<None>

The search base to use for users. Defaults to *ou=Users* with the *[ldap] suffix* appended to it.

`user_filter`

Type
string

Default
<None>

The LDAP search filter to use for users.

`user_objectclass`

Type
string

Default
`inetOrgPerson`

The LDAP object class to use for users.

`user_id_attribute`

Type
string

Default
`cn`

The LDAP attribute mapped to user IDs in keystone. This must NOT be a multivalued attribute. User IDs are expected to be globally unique across keystone domains and URL-safe.

`user_name_attribute`

Type
string

Default
`sn`

The LDAP attribute mapped to user names in keystone. User names are expected to be unique only within a keystone domain and are not expected to be URL-safe.

`user_description_attribute`

Type
string

Default
`description`

The LDAP attribute mapped to user descriptions in keystone.

`user_mail_attribute`

Type
string

Default
`mail`

The LDAP attribute mapped to user emails in keystone.

`user_pass_attribute`

Type
string

Default
`userPassword`

The LDAP attribute mapped to user passwords in keystone.

`user_enabled_attribute`

Type
string

Default
`enabled`

The LDAP attribute mapped to the user enabled attribute in keystone. If setting this option to *userAccountControl*, then you may be interested in setting *[ldap] user_enabled_mask* and *[ldap] user_enabled_default* as well.

`user_enabled_invert`

Type
boolean

Default
`False`

Logically negate the boolean value of the enabled attribute obtained from the LDAP server. Some LDAP servers use a boolean lock attribute where true means an account is disabled. Setting *[ldap] user_enabled_invert = true* will allow these lock attributes to be used. This option will have no effect if either the *[ldap] user_enabled_mask* or *[ldap] user_enabled_emulation* options are in use.

`user_enabled_mask`

Type
integer

Default
`0`

Minimum Value
0

Bitmask integer to select which bit indicates the enabled value if the LDAP server represents enabled as a bit on an integer rather than as a discrete boolean. A value of *0* indicates that the mask is not used. If this is not set to *0* the typical value is *2*. This is typically used when *[ldap] user_enabled_attribute = userAccountControl*. Setting this option causes keystone to ignore the value of *[ldap] user_enabled_invert*.

`user_enabled_default`

Type
string

Default
`True`

The default value to enable users. This should match an appropriate integer value if the LDAP server uses non-boolean (bitmask) values to indicate if a user is enabled or disabled. If this is not set to *True*, then the typical value is *512*. This is typically used when *[ldap] user_enabled_attribute = userAccountControl*.

`user_attribute_ignore`

Type
list

Default
`['default_project_id']`

List of user attributes to ignore on create and update, or whether a specific user attribute should be filtered for list or show user.

`user_default_project_id_attribute`

Type
string

Default
<None>

The LDAP attribute mapped to a users default_project_id in keystone. This is most commonly used when keystone has write access to LDAP.

`user_enabled_emulation`

Type
boolean

Default
`False`

If enabled, keystone uses an alternative method to determine if a user is enabled or not by checking if they are a member of the group defined by the *[ldap] user_enabled_emulation_dn* option. Enabling this option causes keystone to ignore the value of *[ldap] user_enabled_invert*.

`user_enabled_emulation_dn`

Type
string

Default
<None>

DN of the group entry to hold enabled users when using enabled emulation. Setting this option has no effect unless *[ldap] user_enabled_emulation* is also enabled.

`user_enabled_emulation_use_group_config`

Type
boolean

Default
`False`

Use the *[ldap] group_member_attribute* and *[ldap] group_objectclass* settings to determine membership in the emulated enabled group. Enabling this option has no effect unless *[ldap] user_enabled_emulation* is also enabled.

`user_additional_attribute_mapping`

Type
list

Default
`[]`

A list of LDAP attribute to keystone user attribute pairs used for mapping additional attributes to users in keystone. The expected format is *<ldap_attr>:<user_attr>*, where *ldap_attr* is the attribute in the LDAP object and *user_attr* is the attribute which should appear in the identity API.

`group_tree_dn`

Type
string

Default
<None>

The search base to use for groups. Defaults to *ou=UserGroups* with the *[ldap] suffix* appended to it.

`group_filter`

Type
string

Default
<None>

The LDAP search filter to use for groups.

`group_objectclass`

Type
string

Default
`groupOfNames`

The LDAP object class to use for groups. If setting this option to *posixGroup*, you may also be interested in enabling the *[ldap] group_members_are_ids* option.

`group_id_attribute`

Type
string

Default
`cn`

The LDAP attribute mapped to group IDs in keystone. This must NOT be a multivalued attribute. Group IDs are expected to be globally unique across keystone domains and URL-safe.

`group_name_attribute`

Type
string

Default
`ou`

The LDAP attribute mapped to group names in keystone. Group names are expected to be unique only within a keystone domain and are not expected to be URL-safe.

`group_member_attribute`

Type
string

Default
`member`

The LDAP attribute used to indicate that a user is a member of the group.

`group_members_are_ids`

Type
boolean

Default
`False`

Enable this option if the members of the group object class are keystone user IDs rather than LDAP DNs. This is the case when using *posixGroup* as the group object class in Open Directory.

`group_desc_attribute`

Type
string

Default
`description`

The LDAP attribute mapped to group descriptions in keystone.

`group_attribute_ignore`

Type
list

Default
`[]`

List of group attributes to ignore on create and update. or whether a specific group attribute should be filtered for list or show group.

`group_additional_attribute_mapping`

Type
list

Default
`[]`

A list of LDAP attribute to keystone group attribute pairs used for mapping additional attributes to groups in keystone. The expected format is <*ldap_attr*>:<*group_attr*>, where *ldap_attr* is the attribute in the LDAP object and *group_attr* is the attribute which should appear in the identity API.

`group_ad_nesting`

Type
boolean

Default
`False`

If enabled, group queries will use Active Directory specific filters for nested groups.

`tls_cacertfile`

Type
string

Default
<None>

An absolute path to a CA certificate file to use when communicating with LDAP servers. This option will take precedence over *[ldap] tls_cacertdir*, so there is no reason to set both.

`tls_cacertdir`

Type
string

Default
<None>

An absolute path to a CA certificate directory to use when communicating with LDAP servers. There is no reason to set this option if youve also set *[ldap] tls_cacertfile*.

`use_tls`

Type
boolean

Default
`False`

Enable TLS when communicating with LDAP servers. You should also set the *[ldap] tls_cacertfile* and *[ldap] tls_cacertdir* options when using this option. Do not set this option if you are using LDAP over SSL (LDAPS) instead of TLS.

`tls_req_cert`

Type
string

Default
`demand`

Valid Values
demand, never, allow

Specifies which checks to perform against client certificates on incoming TLS sessions. If set to *demand*, then a certificate will always be requested and required from the LDAP server. If set to *allow*, then a certificate will always be requested but not required from the LDAP server. If set to *never*, then a certificate will never be requested.

`connection_timeout`

Type
integer

Default
`-1`

Minimum Value
-1

The connection timeout to use with the LDAP server. A value of *-1* means that connections will never timeout.

`use_pool`

Type
boolean

Default
`True`

Enable LDAP connection pooling for queries to the LDAP server. There is typically no reason to disable this.

`pool_size`

Type
integer

Default
`10`

Minimum Value
1

The size of the LDAP connection pool. This option has no effect unless *[ldap] use_pool* is also enabled.

`pool_retry_max`

Type
integer

Default
3

Minimum Value
1

The maximum number of times to attempt connecting to the LDAP server before aborting. A value of one makes only one connection attempt. This option has no effect unless *[ldap] use_pool* is also enabled.

`pool_retry_delay`

Type
floating point

Default
0.1

The number of seconds to wait before attempting to reconnect to the LDAP server. This option has no effect unless *[ldap] use_pool* is also enabled.

`pool_connection_timeout`

Type
integer

Default
-1

Minimum Value
-1

The connection timeout to use when pooling LDAP connections. A value of *-1* means that connections will never timeout. This option has no effect unless *[ldap] use_pool* is also enabled.

`pool_connection_lifetime`

Type
integer

Default
600

Minimum Value
1

The maximum connection lifetime to the LDAP server in seconds. When this lifetime is exceeded, the connection will be unbound and removed from the connection pool. This option has no effect unless *[ldap] use_pool* is also enabled.

`use_auth_pool`

Type
boolean

Default
`True`

Enable LDAP connection pooling for end user authentication. There is typically no reason to disable this.

`auth_pool_size`

Type
integer

Default
`100`

Minimum Value
1

The size of the connection pool to use for end user authentication. This option has no effect unless *[ldap] use_auth_pool* is also enabled.

`auth_pool_connection_lifetime`

Type
integer

Default
`60`

Minimum Value
1

The maximum end user authentication connection lifetime to the LDAP server in seconds. When this lifetime is exceeded, the connection will be unbound and removed from the connection pool. This option has no effect unless *[ldap] use_auth_pool* is also enabled.

oauth1

`driver`

Type
string

Default
`sql`

Entry point for the OAuth backend driver in the *keystone.oauth1* namespace. Typically, there is no reason to set this option unless you are providing a custom entry point.

`request_token_duration`

Type
integer

Default
`28800`

Minimum Value
0

Number of seconds for the OAuth Request Token to remain valid after being created. This is the amount of time the user has to authorize the token. Setting this option to zero means that request tokens will last forever.

`access_token_duration`

Type
integer

Default
`86400`

Minimum Value
0

Number of seconds for the OAuth Access Token to remain valid after being created. This is the amount of time the consumer has to interact with the service provider (which is typically keystone). Setting this option to zero means that access tokens will last forever.

oauth2

`oauth2_authn_methods`

Type
list

Default
`['tls_client_auth', 'client_secret_basic']`

The OAuth2.0 authentication method supported by the system when user obtains an access token through the OAuth2.0 token endpoint. This option can be set to certificate or secret. If the option is not set, the default value is certificate. When the option is set to secret, the OAuth2.0 token endpoint uses client_secret_basic method for authentication, otherwise tls_client_auth method is used for authentication.

`oauth2_cert_dn_mapping_id`

Type
string

Default
`oauth2_mapping`

Used to define the mapping rule id. When not set, the mapping rule id is oauth2_mapping.

oslo_messaging_amqp

`container_name`

Type
string

Default
<None>

Name for the AMQP container. must be globally unique. Defaults to a generated UUID

Table 6: Deprecated Variations

Group	Name
amqp1	container_name

`idle_timeout`

Type
integer

Default
`0`

Timeout for inactive connections (in seconds)

Table 7: Deprecated Variations

Group	Name
amqp1	idle_timeout

`trace`

Type
boolean

Default
`False`

Debug: dump AMQP frames to stdout

Table 8: Deprecated Variations

Group	Name
amqp1	trace

`ssl`

Type
boolean

Default
`False`

Attempt to connect via SSL. If no other ssl-related parameters are given, it will use the systems CA-bundle to verify the servers certificate.

`ssl_ca_file`

Type
string

Default
`''`

CA certificate PEM file used to verify the servers certificate

Table 9: Deprecated Variations

Group	Name
amqp1	ssl_ca_file

ssl_cert_file

Type
string

Default
''

Self-identifying certificate PEM file for client authentication

Table 10: Deprecated Variations

Group	Name
amqp1	ssl_cert_file

ssl_key_file

Type
string

Default
''

Private key PEM file used to sign ssl_cert_file certificate (optional)

Table 11: Deprecated Variations

Group	Name
amqp1	ssl_key_file

ssl_key_password

Type
string

Default
<None>

Password for decrypting ssl_key_file (if encrypted)

Table 12: Deprecated Variations

Group	Name
amqp1	ssl_key_password

`ssl_verify_vhost`

Type
boolean

Default
`False`

By default SSL checks that the name in the servers certificate matches the hostname in the transport_url. In some configurations it may be preferable to use the virtual hostname instead, for example if the server uses the Server Name Indication TLS extension (rfc6066) to provide a certificate per virtual host. Set ssl_verify_vhost to True if the servers SSL certificate uses the virtual host name instead of the DNS name.

`sasl_mechanisms`

Type
string

Default
`''`

Space separated list of acceptable SASL mechanisms

Table 13: Deprecated Variations

Group	Name
amqp1	sasl_mechanisms

`sasl_config_dir`

Type
string

Default
`''`

Path to directory that contains the SASL configuration

Table 14: Deprecated Variations

Group	Name
amqp1	sasl_config_dir

`sasl_config_name`

Type
string

Default
`''`

Name of configuration file (without .conf suffix)

Table 15: Deprecated Variations

Group	Name
amqp1	sasl_config_name

`sasl_default_realm`

Type
string

Default
''

SASL realm to use if no realm present in username

`connection_retry_interval`

Type
integer

Default
1

Minimum Value
1

Seconds to pause before attempting to re-connect.

`connection_retry_backoff`

Type
integer

Default
2

Minimum Value
0

Increase the connection_retry_interval by this many seconds after each unsuccessful failover attempt.

`connection_retry_interval_max`

Type
integer

Default
30

Minimum Value
1

Maximum limit for connection_retry_interval + connection_retry_backoff

`link_retry_delay`

Type
integer

Default
10

Minimum Value
1

Time to pause between re-connecting an AMQP 1.0 link that failed due to a recoverable error.

`default_reply_retry`

Type
integer

Default
0

Minimum Value
-1

The maximum number of attempts to re-send a reply message which failed due to a recoverable error.

`default_reply_timeout`

Type
integer

Default
30

Minimum Value
5

The deadline for an rpc reply message delivery.

`default_send_timeout`

Type
integer

Default
30

Minimum Value
5

The deadline for an rpc cast or call message delivery. Only used when caller does not provide a timeout expiry.

`default_notify_timeout`

Type
integer

Default
30

Minimum Value
5

The deadline for a sent notification message delivery. Only used when caller does not provide a timeout expiry.

`default_sender_link_timeout`

Type
integer

Default
`600`

Minimum Value
1

The duration to schedule a purge of idle sender links. Detach link after expiry.

`addressing_mode`

Type
string

Default
`dynamic`

Indicates the addressing mode used by the driver. Permitted values: legacy - use legacy non-routable addressing routable - use routable addresses dynamic - use legacy addresses if the message bus does not support routing otherwise use routable addressing

`pseudo_vhost`

Type
boolean

Default
`True`

Enable virtual host support for those message buses that do not natively support virtual hosting (such as qpidd). When set to true the virtual host name will be added to all message bus addresses, effectively creating a private subnet per virtual host. Set to False if the message bus supports virtual hosting using the hostname field in the AMQP 1.0 Open performative as the name of the virtual host.

`server_request_prefix`

Type
string

Default
`exclusive`

address prefix used when sending to a specific server

Table 16: Deprecated Variations

Group	Name
amqp1	server_request_prefix

`broadcast_prefix`

Type
string

Default
`broadcast`

address prefix used when broadcasting to all servers

Table 17: Deprecated Variations

Group	Name
amqp1	broadcast_prefix

`group_request_prefix`

Type
string

Default
`unicast`

address prefix when sending to any server in group

Table 18: Deprecated Variations

Group	Name
amqp1	group_request_prefix

`rpc_address_prefix`

Type
string

Default
`openstack.org/om/rpc`

Address prefix for all generated RPC addresses

`notify_address_prefix`

Type
string

Default
`openstack.org/om/notify`

Address prefix for all generated Notification addresses

`multicast_address`

Type
string

Default
`multicast`

Appended to the address prefix when sending a fanout message. Used by the message bus to identify fanout messages.

`unicast_address`

Type
string

Default
`unicast`

Appended to the address prefix when sending to a particular RPC/Notification server. Used by the message bus to identify messages sent to a single destination.

`anycast_address`

Type
string

Default
`anycast`

Appended to the address prefix when sending to a group of consumers. Used by the message bus to identify messages that should be delivered in a round-robin fashion across consumers.

`default_notification_exchange`

Type
string

Default
`<None>`

Exchange name used in notification addresses. Exchange name resolution precedence: Target.exchange if set else default_notification_exchange if set else control_exchange if set else notify

`default_rpc_exchange`

Type
string

Default
`<None>`

Exchange name used in RPC addresses. Exchange name resolution precedence: Target.exchange if set else default_rpc_exchange if set else control_exchange if set else rpc

`reply_link_credit`

Type
integer

Default
`200`

Minimum Value
1

Window size for incoming RPC Reply messages.

`rpc_server_credit`

Type
integer

Default
100

Minimum Value
1

Window size for incoming RPC Request messages

`notify_server_credit`

Type
integer

Default
100

Minimum Value
1

Window size for incoming Notification messages

`pre_settled`

Type
multi-valued

Default
rpc-cast

Default
rpc-reply

Send messages of this type pre-settled. Pre-settled messages will not receive acknowledgement from the peer. Note well: pre-settled messages may be silently discarded if the delivery fails. Permitted values: rpc-call - send RPC Calls pre-settled rpc-reply- send RPC Replies pre-settled rpc-cast - Send RPC Casts pre-settled notify - Send Notifications pre-settled

oslo_messaging_kafka

`kafka_max_fetch_bytes`

Type
integer

Default
1048576

Max fetch bytes of Kafka consumer

`kafka_consumer_timeout`

Type
floating point

Default
1.0

Default timeout(s) for Kafka consumers

`pool_size`

Type
integer

Default
`10`

Pool Size for Kafka Consumers

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
Driver no longer uses connection pool.

`conn_pool_min_size`

Type
integer

Default
`2`

The pool size limit for connections expiration policy

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
Driver no longer uses connection pool.

`conn_pool_ttl`

Type
integer

Default
`1200`

The time-to-live in sec of idle connections in the pool

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
Driver no longer uses connection pool.

`consumer_group`

Type
string

Default
`oslo_messaging_consumer`

Group id for Kafka consumer. Consumers in one group will coordinate message consumption

`producer_batch_timeout`

Type
floating point

Default
`0.0`

Upper bound on the delay for KafkaProducer batching in seconds

`producer_batch_size`

Type
integer

Default
`16384`

Size of batch for the producer async send

`compression_codec`

Type
string

Default
`none`

Valid Values
none, gzip, snappy, lz4, zstd

The compression codec for all data generated by the producer. If not set, compression will not be used. Note that the allowed values of this depend on the kafka version

`enable_auto_commit`

Type
boolean

Default
`False`

Enable asynchronous consumer commits

`max_poll_records`

Type
integer

Default
`500`

The maximum number of records returned in a poll call

`security_protocol`

Type
string

Default
`PLAINTEXT`

Valid Values
PLAINTEXT, SASL_PLAINTEXT, SSL, SASL_SSL

Protocol used to communicate with brokers

`sasl_mechanism`

Type
string

Default
`PLAIN`

Mechanism when security protocol is SASL

`ssl_cafile`

Type
string

Default
`''`

CA certificate PEM file used to verify the server certificate

`ssl_client_cert_file`

Type
string

Default
`''`

Client certificate PEM file used for authentication.

`ssl_client_key_file`

Type
string

Default
`''`

Client key PEM file used for authentication.

`ssl_client_key_password`

Type
string

Default
`''`

Client key password file used for authentication.

oslo_messaging_notifications

driver

Type
multi-valued

Default
''

The Drivers(s) to handle sending notifications. Possible values are messaging, messagingv2, routing, log, test, noop

Table 19: Deprecated Variations

Group	Name
DEFAULT	notification_driver

transport_url

Type
string

Default
<None>

A URL representing the messaging driver to use for notifications. If not set, we fall back to the same configuration used for RPC.

Table 20: Deprecated Variations

Group	Name
DEFAULT	notification_transport_url

topics

Type
list

Default
['notifications']

AMQP topic used for OpenStack notifications.

Table 21: Deprecated Variations

Group	Name
rpc_notifier2	topics
DEFAULT	notification_topics

retry

Type
integer

Default
`-1`

The maximum number of attempts to re-send a notification message which failed to be delivered due to a recoverable error. 0 - No retry, -1 - indefinite

oslo_messaging_rabbit

amqp_durable_queues

Type
boolean

Default
`False`

Use durable queues in AMQP. If rabbit_quorum_queue is enabled, queues will be durable and this value will be ignored.

amqp_auto_delete

Type
boolean

Default
`False`

Auto-delete queues in AMQP.

Table 22: Deprecated Variations

Group	Name
DEFAULT	amqp_auto_delete

ssl

Type
boolean

Default
`False`

Connect over SSL.

Table 23: Deprecated Variations

Group	Name
oslo_messaging_rabbit	rabbit_use_ssl

ssl_version

Type
string

Default
''

SSL version to use (valid only if SSL enabled). Valid values are TLSv1 and SSLv23. SSLv2, SSLv3, TLSv1_1, and TLSv1_2 may be available on some distributions.

Table 24: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_version

`ssl_key_file`

Type
string

Default
''

SSL key file (valid only if SSL enabled).

Table 25: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_keyfile

`ssl_cert_file`

Type
string

Default
''

SSL cert file (valid only if SSL enabled).

Table 26: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_certfile

`ssl_ca_file`

Type
string

Default
''

SSL certification authority file (valid only if SSL enabled).

Table 27: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_ca_certs

ssl_enforce_fips_mode

Type
boolean

Default
`False`

Global toggle for enforcing the OpenSSL FIPS mode. This feature requires Python support. This is available in Python 3.9 in all environments and may have been backported to older Python versions on select environments. If the Python executable used does not support OpenSSL FIPS mode, an exception will be raised.

heartbeat_in_pthread

Type
boolean

Default
`False`

(DEPRECATED) It is recommend not to use this option anymore. Run the health check heartbeat thread through a native python thread by default. If this option is equal to False then the health check heartbeat will inherit the execution model from the parent process. For example if the parent process has monkey patched the stdlib by using eventlet/greenlet then the heartbeat will be run through a green thread. This option should be set to True only for the wsgi services.

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
The option is related to Eventlet which will be removed. In addition this has never worked as expected with services using eventlet for core service framework.

kombu_reconnect_delay

Type
floating point

Default
`1.0`

Minimum Value
0.0

Maximum Value
4.5

How long to wait (in seconds) before reconnecting in response to an AMQP consumer cancel notification.

Table 28: Deprecated Variations

Group	Name
DEFAULT	kombu_reconnect_delay

kombu_compression

Type
string

Default
<None>

EXPERIMENTAL: Possible values are: gzip, bz2. If not set compression will not be used. This option may not be available in future versions.

kombu_missing_consumer_retry_timeout

Type
integer

Default
60

How long to wait a missing client before abandoning to send it its replies. This value should not be longer than rpc_response_timeout.

Table 29: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_reconnect_timeout

kombu_failover_strategy

Type
string

Default
round-robin

Valid Values
round-robin, shuffle

Determines how the next RabbitMQ node is chosen in case the one we are currently connected to becomes unavailable. Takes effect only if more than one RabbitMQ node is provided in config.

rabbit_login_method

Type
string

Default
AMQPLAIN

Valid Values
PLAIN, AMQPLAIN, EXTERNAL, RABBIT-CR-DEMO

The RabbitMQ login method.

Table 30: Deprecated Variations

Group	Name
DEFAULT	rabbit_login_method

`rabbit_retry_interval`

Type
integer

Default
`1`

How frequently to retry connecting with RabbitMQ.

`rabbit_retry_backoff`

Type
integer

Default
`2`

How long to backoff for between retries when connecting to RabbitMQ.

Table 31: Deprecated Variations

Group	Name
DEFAULT	rabbit_retry_backoff

`rabbit_interval_max`

Type
integer

Default
`30`

Maximum interval of RabbitMQ connection retries. Default is 30 seconds.

`rabbit_ha_queues`

Type
boolean

Default
`False`

Try to use HA queues in RabbitMQ (x-ha-policy: all). If you change this option, you must wipe the RabbitMQ database. In RabbitMQ 3.0, queue mirroring is no longer controlled by the x-ha-policy argument when declaring a queue. If you just want to make sure that all queues (except those with

auto-generated names) are mirrored across all nodes, run: rabbitmqctl set_policy HA ^(?!amq.).* {ha-mode: all}

Table 32: Deprecated Variations

Group	Name
DEFAULT	rabbit_ha_queues

`rabbit_quorum_queue`

Type
boolean

Default
`False`

Use quorum queues in RabbitMQ (x-queue-type: quorum). The quorum queue is a modern queue type for RabbitMQ implementing a durable, replicated FIFO queue based on the Raft consensus algorithm. It is available as of RabbitMQ 3.8.0. If set this option will conflict with the HA queues (`rabbit_ha_queues`) aka mirrored queues, in other words the HA queues should be disabled. Quorum queues are also durable by default so the amqp_durable_queues option is ignored when this option is enabled.

`rabbit_transient_quorum_queue`

Type
boolean

Default
`False`

Use quorum queues for transients queues in RabbitMQ. Enabling this option will then make sure those queues are also using quorum kind of rabbit queues, which are HA by default.

`rabbit_quorum_delivery_limit`

Type
integer

Default
`0`

Each time a message is redelivered to a consumer, a counter is incremented. Once the redelivery count exceeds the delivery limit the message gets dropped or dead-lettered (if a DLX exchange has been configured) Used only when rabbit_quorum_queue is enabled, Default 0 which means dont set a limit.

`rabbit_quorum_max_memory_length`

Type
integer

Default
`0`

By default all messages are maintained in memory if a quorum queue grows in length it can put memory pressure on a cluster. This option can limit the number of messages in the quorum queue. Used only when rabbit_quorum_queue is enabled, Default 0 which means dont set a limit.

Table 33: Deprecated Variations

Group	Name
oslo_messaging_rabbit	rabbit_quroum_max_memory_length

rabbit_quorum_max_memory_bytes

Type
integer

Default
0

By default all messages are maintained in memory if a quorum queue grows in length it can put memory pressure on a cluster. This option can limit the number of memory bytes used by the quorum queue. Used only when rabbit_quorum_queue is enabled, Default 0 which means dont set a limit.

Table 34: Deprecated Variations

Group	Name
oslo_messaging_rabbit	rabbit_quroum_max_memory_bytes

rabbit_transient_queues_ttl

Type
integer

Default
1800

Minimum Value
0

Positive integer representing duration in seconds for queue TTL (x-expires). Queues which are unused for the duration of the TTL are automatically deleted. The parameter affects only reply and fanout queues. Setting 0 as value will disable the x-expires. If doing so, make sure you have a rabbitmq policy to delete the queues or you deployment will create an infinite number of queue over time.

rabbit_qos_prefetch_count

Type
integer

Default
0

Specifies the number of messages to prefetch. Setting to zero allows unlimited messages.

heartbeat_timeout_threshold

Type
integer

Default
`60`

Number of seconds after which the Rabbit broker is considered down if heartbeats keep-alive fails (0 disables heartbeat).

`heartbeat_rate`

Type
integer

Default
`3`

How often times during the heartbeat_timeout_threshold we check the heartbeat.

`direct_mandatory_flag`

Type
boolean

Default
`True`

(DEPRECATED) Enable/Disable the RabbitMQ mandatory flag for direct send. The direct send is used as reply, so the MessageUndeliverable exception is raised in case the client queue does not exist.MessageUndeliverable exception will be used to loop for a timeout to lets a chance to sender to recover.This flag is deprecated and it will not be possible to deactivate this functionality anymore

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
Mandatory flag no longer deactivable.

`enable_cancel_on_failover`

Type
boolean

Default
`False`

Enable x-cancel-on-ha-failover flag so that rabbitmq server will cancel and notify consumerswhen queue is down

`use_queue_manager`

Type
boolean

Default
`False`

Should we use consistant queue names or random ones

hostname

Type
string

Default
`node1.example.com`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Hostname used by queue manager. Defaults to the value returned by socket.gethostname().

processname

Type
string

Default
`sphinx-build`

Process name used by queue manager

rabbit_stream_fanout

Type
boolean

Default
`False`

Use stream queues in RabbitMQ (x-queue-type: stream). Streams are a new persistent and replicated data structure (queue type) in RabbitMQ which models an append-only log with non-destructive consumer semantics. It is available as of RabbitMQ 3.9.0. If set this option will replace all fanout queues with only one stream queue.

oslo_middleware

max_request_body_size

Type
integer

Default
`114688`

The maximum body size for each request, in bytes.

Table 35: Deprecated Variations

Group	Name
DEFAULT	osapi_max_request_body_size
DEFAULT	max_request_body_size

enable_proxy_headers_parsing

Type
boolean

Default
`False`

Whether the application is behind a proxy or not. This determines if the middleware should parse the headers or not.

`http_basic_auth_user_file`

Type
string

Default
`/etc/htpasswd`

HTTP basic auth password file.

oslo_policy

`enforce_scope`

Type
boolean

Default
`True`

This option controls whether or not to enforce scope when evaluating policies. If `True`, the scope of the token used in the request is compared to the `scope_types` of the policy being enforced. If the scopes do not match, an `InvalidScope` exception will be raised. If `False`, a message will be logged informing operators that policies are being invoked with mismatching scope.

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
This configuration was added temporarily to facilitate a smooth transition to the new RBAC. OpenStack will always enforce scope checks. This configuration option is deprecated and will be removed in the 2025.2 cycle.

`enforce_new_defaults`

Type
boolean

Default
`True`

This option controls whether or not to use old deprecated defaults when evaluating policies. If `True`, the old deprecated defaults are not going to be evaluated. This means if any existing token is allowed for old defaults but is disallowed for new defaults, it will be disallowed. It is encouraged to enable this flag along with the `enforce_scope` flag so that you can get the benefits of new defaults and `scope_type` together. If `False`, the deprecated policy check string is logically ORd with the

new policy check string, allowing for a graceful upgrade experience between releases with new policies, which is the default behavior.

`policy_file`

Type
string

Default
`policy.yaml`

The relative or absolute path of a file that maps roles to permissions for a given service. Relative paths must be specified in relation to the configuration file setting this option.

Table 36: Deprecated Variations

Group	Name
DEFAULT	policy_file

`policy_default_rule`

Type
string

Default
`default`

Default rule. Enforced when a requested rule is not found.

Table 37: Deprecated Variations

Group	Name
DEFAULT	policy_default_rule

`policy_dirs`

Type
multi-valued

Default
`policy.d`

Directories where policy configuration files are stored. They can be relative to any directory in the search path defined by the config_dir option, or absolute paths. The file defined by policy_file must exist for these directories to be searched. Missing or empty directories are ignored.

Table 38: Deprecated Variations

Group	Name
DEFAULT	policy_dirs

`remote_content_type`

Type
string

Default
`application/x-www-form-urlencoded`

Valid Values
application/x-www-form-urlencoded, application/json

Content Type to send and receive data for REST based policy check

`remote_ssl_verify_server_crt`

Type
boolean

Default
`False`

server identity verification for REST based policy check

`remote_ssl_ca_crt_file`

Type
string

Default
<None>

Absolute path to ca cert file for REST based policy check

`remote_ssl_client_crt_file`

Type
string

Default
<None>

Absolute path to client cert for REST based policy check

`remote_ssl_client_key_file`

Type
string

Default
<None>

Absolute path client key file REST based policy check

policy

`driver`

Type
string

Default
`sql`

Entry point for the policy backend driver in the *keystone.policy* namespace. Supplied drivers are *rules* (which does not support any CRUD operations for the v3 policy API) and *sql*. Typically, there is no reason to set this option unless you are providing a custom entry point.

`list_limit`

Type
integer

Default
<None>

Maximum number of entities that will be returned in a policy collection.

profiler

`enabled`

Type
boolean

Default
`False`

Enable the profiling for all services on this node.

Default value is False (fully disable the profiling feature).

Possible values:

- True: Enables the feature
- False: Disables the feature. The profiling cannot be started via this project operations. If the profiling is triggered by another project, this project part will be empty.

Table 39: Deprecated Variations

Group	Name
profiler	profiler_enabled

`trace_sqlalchemy`

Type
boolean

Default
`False`

Enable SQL requests profiling in services.

Default value is False (SQL requests wont be traced).

Possible values:

- True: Enables SQL requests profiling. Each SQL query will be part of the trace and can the be analyzed by how much time was spent for that.
- False: Disables SQL requests profiling. The spent time is only shown on a higher level of operations. Single SQL queries cannot be analyzed this way.

`trace_requests`

Type
boolean

Default
`False`

Enable python requests package profiling.

Supported drivers: jaeger+otlp

Default value is False.

Possible values:

- True: Enables requests profiling.
- False: Disables requests profiling.

`hmac_keys`

Type
string

Default
`SECRET_KEY`

Secret key(s) to use for encrypting context data for performance profiling.

This string value should have the following format: <key1>[,<key2>,<keyn>], where each key is some random string. A user who triggers the profiling via the REST API has to set one of these keys in the headers of the REST API call to include profiling results of this node for this particular project.

Both enabled flag and hmac_keys config options should be set to enable profiling. Also, to generate correct profiling information across all services at least one key needs to be consistent between OpenStack projects. This ensures it can be used from client side to generate the trace, containing information from all possible resources.

`connection_string`

Type
string

Default
`messaging://`

Connection string for a notifier backend.

Default value is `messaging://` which sets the notifier to oslo_messaging.

Examples of possible values:

- `messaging://` - use oslo_messaging driver for sending spans.
- `redis://127.0.0.1:6379` - use redis driver for sending spans.
- `mongodb://127.0.0.1:27017` - use mongodb driver for sending spans.
- `elasticsearch://127.0.0.1:9200` - use elasticsearch driver for sending spans.
- `jaeger://127.0.0.1:6831` - use jaeger tracing as driver for sending spans.

`es_doc_type`

Type
string

Default
`notification`

Document type for notification indexing in elasticsearch.

`es_scroll_time`

Type
string

Default
`2m`

This parameter is a time value parameter (for example: es_scroll_time=2m), indicating for how long the nodes that participate in the search will maintain relevant resources in order to continue and support it.

`es_scroll_size`

Type
integer

Default
`10000`

Elasticsearch splits large requests in batches. This parameter defines maximum size of each batch (for example: es_scroll_size=10000).

`socket_timeout`

Type
floating point

Default
`0.1`

Redissentinel provides a timeout option on the connections. This parameter defines that timeout (for example: socket_timeout=0.1).

`sentinel_service_name`

Type
string

Default
`mymaster`

Redissentinel uses a service name to identify a master redis service. This parameter defines the name (for example: `sentinal_service_name=mymaster`).

`filter_error_trace`

Type
boolean

Default
`False`

Enable filter traces that contain error/exception to a separated place.

Default value is set to False.

Possible values:

- True: Enable filter traces that contain error/exception.
- False: Disable the filter.

profiler_jaeger

service_name_prefix

Type
string

Default
<None>

Set service name prefix to Jaeger service name.

process_tags

Type
dict

Default
`{}`

Set process tracer tags.

profiler_otlp

service_name_prefix

Type
string

Default
<None>

Set service name prefix to OTLP exporters.

receipt

expiration

Type
integer

Default
`300`

Minimum Value
0

Maximum Value
86400

The amount of time that a receipt should remain valid (in seconds). This value should always be very short, as it represents how long a user has to reattempt auth with the missing auth methods.

`provider`

Type
string

Default
`fernet`

Entry point for the receipt provider in the *keystone.receipt.provider* namespace. The receipt provider controls the receipt construction and validation operations. Keystone includes just the *fernet* receipt provider for now. *fernet* receipts do not need to be persisted at all, but require that you run *keystone-manage fernet_setup* (also see the *keystone-manage fernet_rotate* command).

`caching`

Type
boolean

Default
`True`

Toggle for caching receipt creation and validation data. This has no effect unless global caching is enabled, or if cache_on_issue is disabled as we only cache receipts on issue.

`cache_time`

Type
integer

Default
`300`

Minimum Value
0

The number of seconds to cache receipt creation and validation data. This has no effect unless both global and *[receipt] caching* are enabled.

`cache_on_issue`

Type
boolean

Default
`True`

Enable storing issued receipt data to receipt validation cache so that first receipt validation doesnt actually cause full validation cycle. This option has no effect unless global caching and receipt caching are enabled.

resource

driver

Type
string

Default
`sql`

Entry point for the resource driver in the *keystone.resource* namespace. Only a *sql* driver is supplied by keystone. Unless you are writing proprietary drivers for keystone, you do not need to set this option.

caching

Type
boolean

Default
`True`

Toggle for resource caching. This has no effect unless global caching is enabled.

Table 40: Deprecated Variations

Group	Name
assignment	caching

cache_time

Type
integer

Default
<None>

Time to cache resource data in seconds. This has no effect unless global caching is enabled.

Table 41: Deprecated Variations

Group	Name
assignment	cache_time

list_limit

Type
integer

Default
<None>

Maximum number of entities that will be returned in a resource collection.

Table 42: Deprecated Variations

Group	Name
assignment	list_limit

admin_project_domain_name

Type
string

Default
<None>

Name of the domain that owns the *admin_project_name*. If left unset, then there is no admin project. *[resource] admin_project_name* must also be set to use this option.

admin_project_name

Type
string

Default
<None>

This is a special project which represents cloud-level administrator privileges across services. Tokens scoped to this project will contain a true *is_admin_project* attribute to indicate to policy systems that the role assignments on that specific project should apply equally across every project. If left unset, then there is no admin project, and thus no explicit means of cross-project role assignments. *[resource] admin_project_domain_name* must also be set to use this option.

project_name_url_safe

Type
string

Default
off

Valid Values
off, new, strict

This controls whether the names of projects are restricted from containing URL-reserved characters. If set to *new*, attempts to create or update a project with a URL-unsafe name will fail. If set to *strict*, attempts to scope a token with a URL-unsafe project name will fail, thereby forcing all project names to be updated to be URL-safe.

domain_name_url_safe

Type
string

Default
off

Valid Values
off, new, strict

This controls whether the names of domains are restricted from containing URL-reserved characters. If set to *new*, attempts to create or update a domain with a URL-unsafe name will fail. If set to *strict*, attempts to scope a token with a URL-unsafe domain name will fail, thereby forcing all domain names to be updated to be URL-safe.

revoke

driver

Type
string

Default
`sql`

Entry point for the token revocation backend driver in the *keystone.revoke* namespace. Keystone only provides a *sql* driver, so there is no reason to set this option unless you are providing a custom entry point.

expiration_buffer

Type
integer

Default
`1800`

Minimum Value
0

The number of seconds after a token has expired before a corresponding revocation event may be purged from the backend.

caching

Type
boolean

Default
`True`

Toggle for revocation event caching. This has no effect unless global caching is enabled.

cache_time

Type
integer

Default
`3600`

Time to cache the revocation list and the revocation events (in seconds). This has no effect unless global and *[revoke] caching* are both enabled.

Table 43: Deprecated Variations

Group	Name
token	revocation_cache_time

role

driver

Type
string

Default
<None>

Entry point for the role backend driver in the *keystone.role* namespace. Keystone only provides a *sql* driver, so theres no reason to change this unless you are providing a custom entry point.

caching

Type
boolean

Default
True

Toggle for role caching. This has no effect unless global caching is enabled. In a typical deployment, there is no reason to disable this.

cache_time

Type
integer

Default
<None>

Time to cache role data, in seconds. This has no effect unless both global caching and *[role] caching* are enabled.

list_limit

Type
integer

Default
<None>

Maximum number of entities that will be returned in a role collection. This may be useful to tune if you have a large number of discrete roles in your deployment.

saml

assertion_expiration_time

Type
integer

Default
3600

Determines the lifetime for any SAML assertions generated by keystone, using *NotOnOrAfter* attributes.

`xmlsec1_binary`

Type
string

Default
`xmlsec1`

Name of, or absolute path to, the binary to be used for XML signing. Although only the XML Security Library (*xmlsec1*) is supported, it may have a non-standard name or path on your system. If keystone cannot find the binary itself, you may need to install the appropriate package, use this option to specify an absolute path, or adjust keystones PATH environment variable.

`certfile`

Type
string

Default
`/etc/keystone/ssl/certs/signing_cert.pem`

Absolute path to the public certificate file to use for SAML signing. The value cannot contain a comma (,).

`keyfile`

Type
string

Default
`/etc/keystone/ssl/private/signing_key.pem`

Absolute path to the private key file to use for SAML signing. The value cannot contain a comma (,).

`idp_entity_id`

Type
URI

Default
<None>

This is the unique entity identifier of the identity provider (keystone) to use when generating SAML assertions. This value is required to generate identity provider metadata and must be a URI (a URL is recommended). For example: *https://keystone.example.com/v3/OS-FEDERATION/saml2/idp*.

`idp_sso_endpoint`

Type
URI

Default
<None>

This is the single sign-on (SSO) service location of the identity provider which accepts HTTP POST requests. A value is required to generate identity provider metadata. For example: *https://keystone.example.com/v3/OS-FEDERATION/saml2/sso*.

`idp_lang`

Type
string

Default
`en`

This is the language used by the identity providers organization.

`idp_organization_name`

Type
string

Default
`SAML Identity Provider`

This is the name of the identity providers organization.

`idp_organization_display_name`

Type
string

Default
`OpenStack SAML Identity Provider`

This is the name of the identity providers organization to be displayed.

`idp_organization_url`

Type
URI

Default
`https://example.com/`

This is the URL of the identity providers organization. The URL referenced here should be useful to humans.

`idp_contact_company`

Type
string

Default
`Example, Inc.`

This is the company name of the identity providers contact person.

`idp_contact_name`

Type
string

Default
`SAML Identity Provider Support`

This is the given name of the identity providers contact person.

`idp_contact_surname`

Type
string

Default
`Support`

This is the surname of the identity providers contact person.

`idp_contact_email`

Type
string

Default
`support@example.com`

This is the email address of the identity providers contact person.

`idp_contact_telephone`

Type
string

Default
`+1 800 555 0100`

This is the telephone number of the identity providers contact person.

`idp_contact_type`

Type
string

Default
`other`

Valid Values
technical, support, administrative, billing, other

This is the type of contact that best describes the identity providers contact person.

`idp_metadata_path`

Type
string

Default
`/etc/keystone/saml2_idp_metadata.xml`

Absolute path to the identity provider metadata file. This file should be generated with the *keystone-manage saml_idp_metadata* command. There is typically no reason to change this value.

`relay_state_prefix`

Type
string

Default
`ss:mem:`

The prefix of the RelayState SAML attribute to use when generating enhanced client and proxy (ECP) assertions. In a typical deployment, there is no reason to change this value.

security_compliance

disable_user_account_days_inactive

Type
integer

Default
<None>

Minimum Value
1

The maximum number of days a user can go without authenticating before being considered inactive and automatically disabled (locked). This feature is disabled by default; set any value to enable it. This feature depends on the *sql* backend for the *[identity] driver*. When a user exceeds this threshold and is considered inactive, the users *enabled* attribute in the HTTP API may not match the value of the users *enabled* column in the user table.

lockout_failure_attempts

Type
integer

Default
<None>

Minimum Value
1

The maximum number of times that a user can fail to authenticate before the user account is locked for the number of seconds specified by *[security_compliance] lockout_duration*. This feature is disabled by default. If this feature is enabled and *[security_compliance] lockout_duration* is not set, then users may be locked out indefinitely until the user is explicitly enabled via the API. This feature depends on the *sql* backend for the *[identity] driver*.

lockout_duration

Type
integer

Default
1800

Minimum Value
1

The number of seconds a user account will be locked when the maximum number of failed authentication attempts (as specified by *[security_compliance] lockout_failure_attempts*) is exceeded. Setting this option will have no effect unless you also set *[security_compliance] lockout_failure_attempts* to a non-zero value. This feature depends on the *sql* backend for the *[identity] driver*.

`password_expires_days`

Type
integer

Default
<None>

Minimum Value
1

The number of days for which a password will be considered valid before requiring it to be changed. This feature is disabled by default. If enabled, new password changes will have an expiration date, however existing passwords would not be impacted. This feature depends on the *sql* backend for the *[identity] driver*.

`unique_last_password_count`

Type
integer

Default
`0`

Minimum Value
0

This controls the number of previous user password iterations to keep in history, in order to enforce that newly created passwords are unique. The total number which includes the new password should not be greater or equal to this value. Setting the value to zero (the default) disables this feature. Thus, to enable this feature, values must be greater than 0. This feature depends on the *sql* backend for the *[identity] driver*.

`minimum_password_age`

Type
integer

Default
`0`

Minimum Value
0

The number of days that a password must be used before the user can change it. This prevents users from changing their passwords immediately in order to wipe out their password history and reuse an old password. This feature does not prevent administrators from manually resetting passwords. It is disabled by default and allows for immediate password changes. This feature depends on the *sql* backend for the *[identity] driver*. Note: If *[security_compliance] password_expires_days* is set, then the value for this option should be less than the *password_expires_days*.

`password_regex`

Type
string

Default
<None>

The regular expression used to validate password strength requirements. By default, the regular expression will match any password. The following is an example of a pattern which requires at least 1 letter, 1 digit, and have a minimum length of 7 characters: ^(?=.*\d)(?=.*[a-zA-Z]).{7,}$ This feature depends on the *sql* backend for the *[identity] driver*.

`password_regex_description`

Type
string

Default
<None>

Describe your password regular expression here in language for humans. If a password fails to match the regular expression, the contents of this configuration variable will be returned to users to explain why their requested password was insufficient.

`change_password_upon_first_use`

Type
boolean

Default
`False`

Enabling this option requires users to change their password when the user is created, or upon administrative reset. Before accessing any services, affected users will have to change their password. To ignore this requirement for specific users, such as service users, set the *options* attribute *ignore_change_password_upon_first_use* to *True* for the desired user via the update user API. This feature is disabled by default. This feature is only applicable with the *sql* backend for the *[identity] driver*.

shadow_users

`driver`

Type
string

Default
`sql`

Entry point for the shadow users backend driver in the *keystone.identity.shadow_users* namespace. This driver is used for persisting local user references to externally-managed identities (via federation, LDAP, etc). Keystone only provides a *sql* driver, so there is no reason to change this option unless you are providing a custom entry point.

token

expiration

Type
integer

Default
`3600`

Minimum Value
0

Maximum Value
9223372036854775807

The amount of time that a token should remain valid (in seconds). Drastically reducing this value may break long-running operations that involve multiple services to coordinate together, and will force users to authenticate with keystone more frequently. Drastically increasing this value will increase the number of tokens that will be simultaneously valid. Keystone tokens are also bearer tokens, so a shorter duration will also reduce the potential security impact of a compromised token.

provider

Type
string

Default
`fernet`

Entry point for the token provider in the *keystone.token.provider* namespace. The token provider controls the token construction, validation, and revocation operations. Supported upstream providers are *fernet* and *jws*. Neither *fernet* or *jws* tokens require persistence and both require additional setup. If using *fernet*, youre required to run *keystone-manage fernet_setup*, which creates symmetric keys used to encrypt tokens. If using *jws*, youre required to generate an ECDSA keypair using a SHA-256 hash algorithm for signing and validating token, which can be done with *keystone-manage create_jws_keypair*. Note that *fernet* tokens are encrypted and *jws* tokens are only signed. Please be sure to consider this if your deployment has security requirements regarding payload contents used to generate token IDs.

caching

Type
boolean

Default
`True`

Toggle for caching token creation and validation data. This has no effect unless global caching is enabled.

cache_time

Type
integer

Default
<None>

Minimum Value
0

Maximum Value
9223372036854775807

The number of seconds to cache token creation and validation data. This has no effect unless both global and *[token] caching* are enabled.

`revoke_by_id`

Type
boolean

Default
`True`

This toggles support for revoking individual tokens by the token identifier and thus various token enumeration operations (such as listing all tokens issued to a specific user). These operations are used to determine the list of tokens to consider revoked. Do not disable this option if youre using the *kvs [revoke] driver*.

`allow_rescope_scoped_token`

Type
boolean

Default
`True`

This toggles whether scoped tokens may be re-scoped to a new project or domain, thereby preventing users from exchanging a scoped token (including those with a default project scope) for any other token. This forces users to either authenticate for unscoped tokens (and later exchange that unscoped token for tokens with a more specific scope) or to provide their credentials in every request for a scoped token to avoid re-scoping altogether.

`cache_on_issue`

Type
boolean

Default
`True`

Enable storing issued token data to token validation cache so that first token validation doesnt actually cause full validation cycle. This option has no effect unless global caching is enabled and will still cache tokens even if *[token] caching = False*.

Warning

This option is deprecated for removal since S. Its value may be silently ignored in the future.

Reason
Keystone already exposes a configuration option for caching tokens. Having a separate configuration option to cache tokens when they are issued is redundant, unnecessarily complicated, and is misleading if token caching is

disabled because tokens will still be pre-cached by default when they are issued. The ability to pre-cache tokens when they are issued is going to rely exclusively on the `keystone.conf [token] caching` option in the future.

`allow_expired_window`

Type
integer

Default
`172800`

This controls the number of seconds that a token can be retrieved for beyond the built-in expiry time. This allows long running operations to succeed. Defaults to two days.

tokenless_auth

`trusted_issuer`

Type
multi-valued

Default
`''`

The list of distinguished names which identify trusted issuers of client certificates allowed to use X.509 tokenless authorization. If the option is absent then no certificates will be allowed. The format for the values of a distinguished name (DN) must be separated by a comma and contain no spaces. Furthermore, because an individual DN may contain commas, this configuration option may be repeated multiple times to represent multiple values. For example, keystone.conf would include two consecutive lines in order to trust two different DNs, such as *trusted_issuer = CN=john,OU=keystone,O=openstack* and *trusted_issuer = CN=mary,OU=eng,O=abc*.

`protocol`

Type
string

Default
`x509`

The federated protocol ID used to represent X.509 tokenless authorization. This is used in combination with the value of *[tokenless_auth] issuer_attribute* to find a corresponding federated mapping. In a typical deployment, there is no reason to change this value.

`issuer_attribute`

Type
string

Default
`SSL_CLIENT_I_DN`

The name of the WSGI environment variable used to pass the issuer of the client certificate to keystone. This attribute is used as an identity provider ID for the X.509 tokenless authorization

along with the protocol to look up its corresponding mapping. In a typical deployment, there is no reason to change this value.

totp

included_previous_windows

Type
integer

Default
1

Minimum Value
0

Maximum Value
10

The number of previous windows to check when processing TOTP passcodes.

trust

allow_redelegation

Type
boolean

Default
`False`

Allows authorization to be redelegated from one user to another, effectively chaining trusts together. When disabled, the *remaining_uses* attribute of a trust is constrained to be zero.

max_redelegation_count

Type
integer

Default
3

Maximum number of times that authorization can be redelegated from one user to another in a chain of trusts. This number may be reduced further for a specific trust.

driver

Type
string

Default
`sql`

Entry point for the trust backend driver in the *keystone.trust* namespace. Keystone only provides a *sql* driver, so there is no reason to change this unless you are providing a custom entry point.

unified_limit

driver

Type
string

Default
`sql`

Entry point for the unified limit backend driver in the *keystone.unified_limit* namespace. Keystone only provides a *sql* driver, so theres no reason to change this unless you are providing a custom entry point.

caching

Type
boolean

Default
`True`

Toggle for unified limit caching. This has no effect unless global caching is enabled. In a typical deployment, there is no reason to disable this.

cache_time

Type
integer

Default
<None>

Time to cache unified limit data, in seconds. This has no effect unless both global caching and *[unified_limit] caching* are enabled.

list_limit

Type
integer

Default
<None>

Maximum number of entities that will be returned in a unified limit collection. This may be useful to tune if you have a large number of unified limits in your deployment.

enforcement_model

Type
string

Default
`flat`

Valid Values
flat, strict_two_level

The enforcement model to use when validating limits associated to projects. Enforcement models will behave differently depending on the existing limits, which may result in backwards incompatible changes if a model is switched in a running deployment.

wsgi

debug_middleware

Type
boolean

Default
`False`

If set to true, this enables the oslo debug middleware in Keystone. This Middleware prints a lot of information about the request and the response. It is useful for getting information about the data on the wire (decoded) and passed to the WSGI application pipeline. This middleware has no effect on the debug setting in the [DEFAULT] section of the config file or setting Keystones log-level to DEBUG; it is specific to debugging the WSGI data as it enters and leaves Keystone (specific request-related data). This option is used for introspection on the request and response data between the web server (apache, nginx, etc) and Keystone. This middleware is inserted as the first element in the middleware chain and will show the data closest to the wire. WARNING: NOT INTENDED FOR USE IN PRODUCTION. THIS MIDDLEWARE CAN AND WILL EMIT SENSITIVE/PRIVILEGED DATA.

9.1.2 Domain-specific Identity drivers

The Identity service supports domain-specific Identity drivers installed on an SQL or LDAP back end, and supports domain-specific Identity configuration options, which are stored in domain-specific configuration files. See *Domain-specific configuration* for more information.

9.2 Policy configuration

Warning

JSON formatted policy file is deprecated since Keystone 19.0.0 (Wallaby). This oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

9.2.1 Configuration

The following is an overview of all available policies in Keystone.

keystone

admin_required

Default
`role:admin or is_admin:1`

(no description provided)

service_role

Default
`role:service`

(no description provided)

service_or_admin

Default
`rule:admin_required or rule:service_role`

(no description provided)

owner

Default
`user_id:%(user_id)s`

(no description provided)

admin_or_owner

Default
`rule:admin_required or rule:owner`

(no description provided)

token_subject

Default
`user_id:%(target.token.user_id)s`

(no description provided)

admin_or_token_subject

Default
`rule:admin_required or rule:token_subject`

(no description provided)

service_admin_or_token_subject

Default
`rule:service_or_admin or rule:token_subject`

(no description provided)

domain_managed_target_role

Default
`'manager':%(target.role.name)s or 'member':%(target.role.name)s or 'reader':%(target.role.name)s`

(no description provided)

`identity:get_access_rule`

Default

```
(role:reader and system_scope:all) or user_id:%(target.user.
id)s
```

Operations

- **GET** /v3/users/{user_id}/access_rules/{access_rule_id}
- **HEAD** /v3/users/{user_id}/access_rules/{access_rule_id}

Scope Types

- **system**
- **project**

Show access rule details.

`identity:list_access_rules`

Default

```
(role:reader and system_scope:all) or user_id:%(target.user.
id)s
```

Operations

- **GET** /v3/users/{user_id}/access_rules
- **HEAD** /v3/users/{user_id}/access_rules

Scope Types

- **system**
- **project**

List access rules for a user.

`identity:delete_access_rule`

Default

```
(role:admin and system_scope:all) or user_id:%(target.user.
id)s
```

Operations

- **DELETE** /v3/users/{user_id}/access_rules/{access_rule_id}

Scope Types

- **system**
- **project**

Delete an access_rule.

`identity:authorize_request_token`

Default

```
rule:admin_required
```

Operations

- **PUT** /v3/OS-OAUTH1/authorize/{request_token_id}

Scope Types

- **project**

Authorize OAUTH1 request token.

`identity:get_access_token`

Default
rule:admin_required

Operations

- **GET** /v3/users/{user_id}/OS-OAUTH1/access_tokens/{access_token_id}

Scope Types

- **project**

Get OAUTH1 access token for user by access token ID.

`identity:get_access_token_role`

Default
rule:admin_required

Operations

- **GET** /v3/users/{user_id}/OS-OAUTH1/access_tokens/{access_token_id}/roles/{role_id}

Scope Types

- **project**

Get role for user OAUTH1 access token.

`identity:list_access_tokens`

Default
rule:admin_required

Operations

- **GET** /v3/users/{user_id}/OS-OAUTH1/access_tokens

Scope Types

- **project**

List OAUTH1 access tokens for user.

`identity:list_access_token_roles`

Default
rule:admin_required

Operations

- **GET** /v3/users/{user_id}/OS-OAUTH1/access_tokens/{access_token_id}/roles

Scope Types

- **project**

List OAUTH1 access token roles.

`identity:delete_access_token`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/users/{user_id}/OS-OAUTH1/access_tokens/{access_token_id}`

Scope Types

- **project**

Delete OAUTH1 access token.

`identity:get_application_credential`

Default
`(rule:admin_required) or (role:reader and system_scope:all) or rule:owner`

Operations

- **GET** `/v3/users/{user_id}/application_credentials/{application_credential_id}`
- **HEAD** `/v3/users/{user_id}/application_credentials/{application_credential_id}`

Scope Types

- **system**
- **project**

Show application credential details.

`identity:list_application_credentials`

Default
`(rule:admin_required) or (role:reader and system_scope:all) or rule:owner`

Operations

- **GET** `/v3/users/{user_id}/application_credentials`
- **HEAD** `/v3/users/{user_id}/application_credentials`

Scope Types

- **system**
- **project**

List application credentials for a user.

`identity:create_application_credential`

Default
`user_id:%(user_id)s`

Operations

- **POST** `/v3/users/{user_id}/application_credentials`

Scope Types

- **project**

Create an application credential.

`identity:delete_application_credential`

Default
`rule:admin_or_owner`

Operations

- **DELETE** `/v3/users/{user_id}/application_credentials/{application_credential_id}`

Scope Types

- **system**
- **project**

Delete an application credential.

`identity:get_auth_catalog`

Default
<empty string>

Operations

- **GET** `/v3/auth/catalog`
- **HEAD** `/v3/auth/catalog`

Get service catalog.

`identity:get_auth_projects`

Default
<empty string>

Operations

- **GET** `/v3/auth/projects`
- **HEAD** `/v3/auth/projects`

List all projects a user has access to via role assignments.

`identity:get_auth_domains`

Default
<empty string>

Operations

- **GET** `/v3/auth/domains`

- **HEAD** /v3/auth/domains

List all domains a user has access to via role assignments.

`identity:get_auth_system`

Default
<empty string>

Operations

- **GET** /v3/auth/system
- **HEAD** /v3/auth/system

List systems a user has access to via role assignments.

`identity:get_consumer`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** /v3/OS-OAUTH1/consumers/{consumer_id}

Scope Types

- **system**
- **project**

Show OAUTH1 consumer details.

`identity:list_consumers`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** /v3/OS-OAUTH1/consumers

Scope Types

- **system**
- **project**

List OAUTH1 consumers.

`identity:create_consumer`

Default
`rule:admin_required`

Operations

- **POST** /v3/OS-OAUTH1/consumers

Scope Types

- **system**
- **project**

Create OAUTH1 consumer.

`identity:update_consumer`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/OS-OAUTH1/consumers/{consumer_id}`

Scope Types

- **system**
- **project**

Update OAUTH1 consumer.

`identity:delete_consumer`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/OS-OAUTH1/consumers/{consumer_id}`

Scope Types

- **system**
- **project**

Delete OAUTH1 consumer.

`identity:get_credential`

Default
`(rule:admin_required) or (role:reader and system_scope:all) or user_id:%(target.credential.user_id)s`

Operations

- **GET** `/v3/credentials/{credential_id}`

Scope Types

- **system**
- **domain**
- **project**

Show credentials details.

`identity:list_credentials`

Default
`(rule:admin_required) or (role:reader and system_scope:all) or user_id:%(target.credential.user_id)s`

Operations

- **GET** `/v3/credentials`

Scope Types

- **system**
- **domain**
- **project**

List credentials.

`identity:create_credential`

Default
`(rule:admin_required) or user_id:%(target.credential.user_id)s`

Operations

- **POST** `/v3/credentials`

Scope Types

- **system**
- **domain**
- **project**

Create credential.

`identity:update_credential`

Default
`(rule:admin_required) or user_id:%(target.credential.user_id)s`

Operations

- **PATCH** `/v3/credentials/{credential_id}`

Scope Types

- **system**
- **domain**
- **project**

Update credential.

`identity:delete_credential`

Default
`(rule:admin_required) or user_id:%(target.credential.user_id)s`

Operations

- **DELETE** `/v3/credentials/{credential_id}`

Scope Types

- **system**
- **domain**
- **project**

Delete credential.

`identity:get_domain`

Default

```
rule:admin_required or (role:reader and system_scope:all) or
token.domain.id:%(target.domain.id)s or token.project.domain.
id:%(target.domain.id)s
```

Operations

- **GET** /v3/domains/{domain_id}

Scope Types

- **system**
- **domain**
- **project**

Show domain details.

`identity:list_domains`

Default

```
rule:admin_required or (role:reader and system_scope:all) or
(role:reader and domain_id:%(target.domain.id)s)
```

Operations

- **GET** /v3/domains

Scope Types

- **system**
- **domain**
- **project**

List domains.

`identity:create_domain`

Default

```
rule:admin_required
```

Operations

- **POST** /v3/domains

Scope Types

- **system**
- **project**

Create domain.

`identity:update_domain`

Default

```
rule:admin_required
```

Operations

- **PATCH** /v3/domains/{domain_id}

Scope Types

- **system**
- **project**

Update domain.

`identity:delete_domain`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/domains/{domain_id}`

Scope Types

- **system**
- **project**

Delete domain.

`identity:create_domain_config`

Default
`rule:admin_required`

Operations

- **PUT** `/v3/domains/{domain_id}/config`

Scope Types

- **system**
- **project**

Create domain configuration.

`identity:get_domain_config`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/domains/{domain_id}/config`
- **HEAD** `/v3/domains/{domain_id}/config`
- **GET** `/v3/domains/{domain_id}/config/{group}`
- **HEAD** `/v3/domains/{domain_id}/config/{group}`
- **GET** `/v3/domains/{domain_id}/config/{group}/{option}`
- **HEAD** `/v3/domains/{domain_id}/config/{group}/{option}`

Scope Types

- **system**
- **project**

Get the entire domain configuration for a domain, an option group within a domain, or a specific configuration option within a group for a domain.

`identity:get_security_compliance_domain_config`

Default
<empty string>

Operations

- **GET** `/v3/domains/{domain_id}/config/security_compliance`
- **HEAD** `/v3/domains/{domain_id}/config/security_compliance`
- **GET** `/v3/domains/{domain_id}/config/security_compliance/{option}`
- **HEAD** `/v3/domains/{domain_id}/config/security_compliance/{option}`

Scope Types

- **system**
- **domain**
- **project**

Get security compliance domain configuration for either a domain or a specific option in a domain.

`identity:update_domain_config`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/domains/{domain_id}/config`
- **PATCH** `/v3/domains/{domain_id}/config/{group}`
- **PATCH** `/v3/domains/{domain_id}/config/{group}/{option}`

Scope Types

- **system**
- **project**

Update domain configuration for either a domain, specific group or a specific option in a group.

`identity:delete_domain_config`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/domains/{domain_id}/config`
- **DELETE** `/v3/domains/{domain_id}/config/{group}`
- **DELETE** `/v3/domains/{domain_id}/config/{group}/{option}`

Scope Types

- **system**
- **project**

Delete domain configuration for either a domain, specific group or a specific option in a group.

`identity:get_domain_config_default`

Default

`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/domains/config/default`
- **HEAD** `/v3/domains/config/default`
- **GET** `/v3/domains/config/{group}/default`
- **HEAD** `/v3/domains/config/{group}/default`
- **GET** `/v3/domains/config/{group}/{option}/default`
- **HEAD** `/v3/domains/config/{group}/{option}/default`

Scope Types

- **system**
- **project**

Get domain configuration default for either a domain, specific group or a specific option in a group.

`identity:ec2_get_credential`

Default

`(rule:admin_required) or (role:reader and system_scope:all) or user_id:%(target.credential.user_id)s`

Operations

- **GET** `/v3/users/{user_id}/credentials/OS-EC2/{credential_id}`

Scope Types

- **system**
- **project**

Show ec2 credential details.

`identity:ec2_list_credentials`

Default

`(rule:admin_required) or (role:reader and system_scope:all) or rule:owner`

Operations

- **GET** `/v3/users/{user_id}/credentials/OS-EC2`

Scope Types

- **system**
- **project**

List ec2 credentials.

`identity:ec2_create_credential`

Default
`rule:admin_or_owner`

Operations

- **POST** `/v3/users/{user_id}/credentials/OS-EC2`

Scope Types

- **system**
- **project**

Create ec2 credential.

`identity:ec2_delete_credential`

Default
`(rule:admin_required) or user_id:%(target.credential.user_id)s`

Operations

- **DELETE** `/v3/users/{user_id}/credentials/OS-EC2/{credential_id}`

Scope Types

- **system**
- **project**

Delete ec2 credential.

`identity:get_endpoint`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/endpoints/{endpoint_id}`

Scope Types

- **system**
- **project**

Show endpoint details.

`identity:list_endpoints`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/endpoints`

Scope Types

- **system**
- **project**

List endpoints.

`identity:create_endpoint`

Default
`rule:admin_required`

Operations

- **POST** `/v3/endpoints`

Scope Types

- **system**
- **project**

Create endpoint.

`identity:update_endpoint`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/endpoints/{endpoint_id}`

Scope Types

- **system**
- **project**

Update endpoint.

`identity:delete_endpoint`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/endpoints/{endpoint_id}`

Scope Types

- **system**
- **project**

Delete endpoint.

`identity:create_endpoint_group`

Default
`rule:admin_required`

Operations

- **POST** `/v3/OS-EP-FILTER/endpoint_groups`

Scope Types

- **system**
- **project**

Create endpoint group.

`identity:list_endpoint_groups`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-EP-FILTER/endpoint_groups`

Scope Types

- **system**
- **project**

List endpoint groups.

`identity:get_endpoint_group`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}`
- **HEAD** `/v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}`

Scope Types

- **system**
- **project**

Get endpoint group.

`identity:update_endpoint_group`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}`

Scope Types

- **system**
- **project**

Update endpoint group.

`identity:delete_endpoint_group`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}`

Scope Types

- **system**

- **project**

Delete endpoint group.

`identity:list_projects_associated_with_endpoint_group`

Default

`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}/projects`

Scope Types

- **system**
- **project**

List all projects associated with a specific endpoint group.

`identity:list_endpoints_associated_with_endpoint_group`

Default

`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}/endpoints`

Scope Types

- **system**
- **project**

List all endpoints associated with an endpoint group.

`identity:get_endpoint_group_in_project`

Default

`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}/projects/{project_id}`
- **HEAD** `/v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}/projects/{project_id}`

Scope Types

- **system**
- **project**

Check if an endpoint group is associated with a project.

`identity:list_endpoint_groups_for_project`

Default

`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** /v3/OS-EP-FILTER/projects/{project_id}/endpoint_groups

Scope Types

- **system**
- **project**

List endpoint groups associated with a specific project.

`identity:add_endpoint_group_to_project`

Default

rule:admin_required

Operations

- **PUT** /v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}/projects/{project_id}

Scope Types

- **system**
- **project**

Allow a project to access an endpoint group.

`identity:remove_endpoint_group_from_project`

Default

rule:admin_required

Operations

- **DELETE** /v3/OS-EP-FILTER/endpoint_groups/{endpoint_group_id}/projects/{project_id}

Scope Types

- **system**
- **project**

Remove endpoint group from project.

`identity:check_grant`

Default

(rule:admin_required) or ((role:reader and system_scope:all) or ((role:reader and domain_id:%(target.user.domain_id)s and domain_id:%(target.project.domain_id)s) or (role:reader and domain_id:%(target.user.domain_id)s and domain_id:%(target.domain.id)s) or (role:reader and domain_id:%(target.group.domain_id)s and domain_id:%(target.project.domain_id)s) or (role:reader and domain_id:%(target.group.domain_id)s and domain_id:%(target.domain.id)s)) and (domain_id:%(target.role.domain_id)s or None:%(target.role.domain_id)s))

Operations

- **HEAD** /v3/projects/{project_id}/users/{user_id}/roles/{role_id}
- **GET** /v3/projects/{project_id}/users/{user_id}/roles/{role_id}
- **HEAD** /v3/projects/{project_id}/groups/{group_id}/roles/{role_id}
- **GET** /v3/projects/{project_id}/groups/{group_id}/roles/{role_id}
- **HEAD** /v3/domains/{domain_id}/users/{user_id}/roles/{role_id}
- **GET** /v3/domains/{domain_id}/users/{user_id}/roles/{role_id}
- **HEAD** /v3/domains/{domain_id}/groups/{group_id}/roles/{role_id}
- **GET** /v3/domains/{domain_id}/groups/{group_id}/roles/{role_id}
- **HEAD** /v3/OS-INHERIT/projects/{project_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- **GET** /v3/OS-INHERIT/projects/{project_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- **HEAD** /v3/OS-INHERIT/projects/{project_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects
- **GET** /v3/OS-INHERIT/projects/{project_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects
- **HEAD** /v3/OS-INHERIT/domains/{domain_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- **GET** /v3/OS-INHERIT/domains/{domain_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- **HEAD** /v3/OS-INHERIT/domains/{domain_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects
- **GET** /v3/OS-INHERIT/domains/{domain_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects

Scope Types

- **system**
- **domain**
- **project**

Check a role grant between a target and an actor. A target can be either a domain or a project. An actor can be either a user or a group. These terms also apply to the OS-INHERIT APIs, where grants on the target are inherited to all projects in the subtree, if applicable.

`identity:list_grants`

Default

```
(rule:admin_required) or ((role:reader and system_scope:all)
or (role:reader and domain_id:%(target.user.domain_id)s and
domain_id:%(target.project.domain_id)s) or (role:reader and
domain_id:%(target.user.domain_id)s and domain_id:%(target.
domain.id)s) or (role:reader and domain_id:%(target.group.
domain_id)s and domain_id:%(target.project.domain_id)s) or
(role:reader and domain_id:%(target.group.domain_id)s and
domain_id:%(target.domain.id)s))
```

Operations

- **GET** /v3/projects/{project_id}/users/{user_id}/roles
- **HEAD** /v3/projects/{project_id}/users/{user_id}/roles
- **GET** /v3/projects/{project_id}/groups/{group_id}/roles
- **HEAD** /v3/projects/{project_id}/groups/{group_id}/roles
- **GET** /v3/domains/{domain_id}/users/{user_id}/roles
- **HEAD** /v3/domains/{domain_id}/users/{user_id}/roles
- **GET** /v3/domains/{domain_id}/groups/{group_id}/roles
- **HEAD** /v3/domains/{domain_id}/groups/{group_id}/roles
- **GET** /v3/OS-INHERIT/domains/{domain_id}/groups/{group_id}/roles/inherited_to_projects
- **GET** /v3/OS-INHERIT/domains/{domain_id}/users/{user_id}/roles/inherited_to_projects

Scope Types

- **system**
- **domain**
- **project**

List roles granted to an actor on a target. A target can be either a domain or a project. An actor can be either a user or a group. For the OS-INHERIT APIs, it is possible to list inherited role grants for actors on domains, where grants are inherited to all projects in the specified domain.

`identity:create_grant`

Default

```
(rule:admin_required) or ((role:admin and domain_id:%(target.
user.domain_id)s and domain_id:%(target.project.domain_id)s)
or (role:admin and domain_id:%(target.user.domain_id)s
and domain_id:%(target.domain.id)s) or (role:admin and
domain_id:%(target.group.domain_id)s and domain_id:%(target.
project.domain_id)s) or (role:admin and domain_id:%(target.
group.domain_id)s and domain_id:%(target.domain.id)s)) and
(domain_id:%(target.role.domain_id)s or None:%(target.role.
domain_id)s) or ((role:manager and domain_id:%(target.user.
domain_id)s and domain_id:%(target.project.domain_id)s)
or (role:manager and domain_id:%(target.user.domain_id)s
```

```
and domain_id:%(target.domain.id)s) or (role:manager and
domain_id:%(target.group.domain_id)s and domain_id:%(target.
project.domain_id)s) or (role:manager and domain_id:%(target.
group.domain_id)s and domain_id:%(target.domain.id)s)) and
rule:domain_managed_target_role
```

Operations

- **PUT** /v3/projects/{project_id}/users/{user_id}/roles/{role_id}
- **PUT** /v3/projects/{project_id}/groups/{group_id}/roles/{role_id}
- **PUT** /v3/domains/{domain_id}/users/{user_id}/roles/{role_id}
- **PUT** /v3/domains/{domain_id}/groups/{group_id}/roles/{role_id}
- **PUT** /v3/OS-INHERIT/projects/{project_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- **PUT** /v3/OS-INHERIT/projects/{project_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects
- **PUT** /v3/OS-INHERIT/domains/{domain_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- **PUT** /v3/OS-INHERIT/domains/{domain_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects

Scope Types

- **system**
- **domain**
- **project**

Create a role grant between a target and an actor. A target can be either a domain or a project. An actor can be either a user or a group. These terms also apply to the OS-INHERIT APIs, where grants on the target are inherited to all projects in the subtree, if applicable.

`identity:revoke_grant`

Default

```
(rule:admin_required) or ((role:admin and domain_id:%(target.
user.domain_id)s and domain_id:%(target.project.domain_id)s)
or (role:admin and domain_id:%(target.user.domain_id)s
and domain_id:%(target.domain.id)s) or (role:admin and
domain_id:%(target.group.domain_id)s and domain_id:%(target.
project.domain_id)s) or (role:admin and domain_id:%(target.
group.domain_id)s and domain_id:%(target.domain.id)s)) and
(domain_id:%(target.role.domain_id)s or None:%(target.role.
domain_id)s) or ((role:manager and domain_id:%(target.user.
domain_id)s and domain_id:%(target.project.domain_id)s)
or (role:manager and domain_id:%(target.user.domain_id)s
and domain_id:%(target.domain.id)s) or (role:manager and
```

domain_id:%(target.group.domain_id)s and domain_id:%(target.project.domain_id)s) or (role:manager and domain_id:%(target.group.domain_id)s and domain_id:%(target.domain.id)s)) and rule:domain_managed_target_role

Operations

- **DELETE** /v3/projects/{project_id}/users/{user_id}/roles/{role_id}
- **DELETE** /v3/projects/{project_id}/groups/{group_id}/roles/{role_id}
- **DELETE** /v3/domains/{domain_id}/users/{user_id}/roles/{role_id}
- **DELETE** /v3/domains/{domain_id}/groups/{group_id}/roles/{role_id}
- **DELETE** /v3/OS-INHERIT/projects/{project_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- **DELETE** /v3/OS-INHERIT/projects/{project_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects
- **DELETE** /v3/OS-INHERIT/domains/{domain_id}/users/{user_id}/roles/{role_id}/inherited_to_projects
- **DELETE** /v3/OS-INHERIT/domains/{domain_id}/groups/{group_id}/roles/{role_id}/inherited_to_projects

Scope Types

- **system**
- **domain**
- **project**

Revoke a role grant between a target and an actor. A target can be either a domain or a project. An actor can be either a user or a group. These terms also apply to the OS-INHERIT APIs, where grants on the target are inherited to all projects in the subtree, if applicable. In that case, revoking the role grant in the target would remove the logical effect of inheriting it to the targets projects subtree.

`identity:list_system_grants_for_user`

Default

rule:admin_required or (role:reader and system_scope:all)

Operations

- **[HEAD, GET]** /v3/system/users/{user_id}/roles

Scope Types

- **system**
- **project**

List all grants a specific user has on the system.

`identity:check_system_grant_for_user`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **[HEAD, GET]** `/v3/system/users/{user_id}/roles/{role_id}`

Scope Types

- **system**
- **project**

Check if a user has a role on the system.

`identity:create_system_grant_for_user`

Default
`rule:admin_required`

Operations

- **[PUT]** `/v3/system/users/{user_id}/roles/{role_id}`

Scope Types

- **system**
- **project**

Grant a user a role on the system.

`identity:revoke_system_grant_for_user`

Default
`rule:admin_required`

Operations

- **[DELETE]** `/v3/system/users/{user_id}/roles/{role_id}`

Scope Types

- **system**
- **project**

Remove a role from a user on the system.

`identity:list_system_grants_for_group`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **[HEAD, GET]** `/v3/system/groups/{group_id}/roles`

Scope Types

- **system**
- **project**

List all grants a specific group has on the system.

`identity:check_system_grant_for_group`

Default

`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **[HEAD, GET]** `/v3/system/groups/{group_id}/roles/{role_id}`

Scope Types

- **system**
- **project**

Check if a group has a role on the system.

`identity:create_system_grant_for_group`

Default

`rule:admin_required`

Operations

- **[PUT]** `/v3/system/groups/{group_id}/roles/{role_id}`

Scope Types

- **system**
- **project**

Grant a group a role on the system.

`identity:revoke_system_grant_for_group`

Default

`rule:admin_required`

Operations

- **[DELETE]** `/v3/system/groups/{group_id}/roles/{role_id}`

Scope Types

- **system**
- **project**

Remove a role from a group on the system.

`identity:get_group`

Default

```
(rule:admin_required) or (role:reader and system_scope:all) or
(role:reader and domain_id:%(target.group.domain_id)s)
```

Operations

- **GET** `/v3/groups/{group_id}`
- **HEAD** `/v3/groups/{group_id}`

Scope Types

- **system**
- **domain**

- **project**

Show group details.

`identity:list_groups`

Default

```
(rule:admin_required) or (role:reader and system_scope:all) or
(role:reader and domain_id:%(target.group.domain_id)s)
```

Operations

- **GET** `/v3/groups`
- **HEAD** `/v3/groups`

Scope Types

- **system**
- **domain**
- **project**

List groups.

`identity:list_groups_for_user`

Default

```
(rule:admin_required) or (role:reader and system_scope:all)
or (role:reader and domain_id:%(target.user.domain_id)s) or
user_id:%(user_id)s
```

Operations

- **GET** `/v3/users/{user_id}/groups`
- **HEAD** `/v3/users/{user_id}/groups`

Scope Types

- **system**
- **domain**
- **project**

List groups to which a user belongs.

`identity:create_group`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
group.domain_id)s)
```

Operations

- **POST** `/v3/groups`

Scope Types

- **system**
- **domain**
- **project**

Create group.

`identity:update_group`

Default
`(rule:admin_required) or (role:manager and domain_id:%(target.group.domain_id)s)`

Operations

- **PATCH** `/v3/groups/{group_id}`

Scope Types

- **system**
- **domain**
- **project**

Update group.

`identity:delete_group`

Default
`(rule:admin_required) or (role:manager and domain_id:%(target.group.domain_id)s)`

Operations

- **DELETE** `/v3/groups/{group_id}`

Scope Types

- **system**
- **domain**
- **project**

Delete group.

`identity:list_users_in_group`

Default
`(rule:admin_required) or (role:reader and system_scope:all) or (role:reader and domain_id:%(target.group.domain_id)s)`

Operations

- **GET** `/v3/groups/{group_id}/users`
- **HEAD** `/v3/groups/{group_id}/users`

Scope Types

- **system**
- **domain**
- **project**

List members of a specific group.

`identity:remove_user_from_group`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
group.domain_id)s and domain_id:%(target.user.domain_id)s)
```

Operations

- **DELETE** /v3/groups/{group_id}/users/{user_id}

Scope Types

- **system**
- **domain**
- **project**

Remove user from group.

`identity:check_user_in_group`

Default

```
(rule:admin_required) or (role:reader and system_scope:all)
or (role:reader and domain_id:%(target.group.domain_id)s and
domain_id:%(target.user.domain_id)s)
```

Operations

- **HEAD** /v3/groups/{group_id}/users/{user_id}
- **GET** /v3/groups/{group_id}/users/{user_id}

Scope Types

- **system**
- **domain**
- **project**

Check whether a user is a member of a group.

`identity:add_user_to_group`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
group.domain_id)s and domain_id:%(target.user.domain_id)s)
```

Operations

- **PUT** /v3/groups/{group_id}/users/{user_id}

Scope Types

- **system**
- **domain**
- **project**

Add user to group.

`identity:create_identity_provider`

Default

```
rule:admin_required
```

Operations

- **PUT** /v3/OS-FEDERATION/identity_providers/{idp_id}

Scope Types

- **system**
- **project**

Create identity provider.

`identity:list_identity_providers`

Default
rule:admin_required or (role:reader and system_scope:all)

Operations

- **GET** /v3/OS-FEDERATION/identity_providers
- **HEAD** /v3/OS-FEDERATION/identity_providers

Scope Types

- **system**
- **project**

List identity providers.

`identity:get_identity_provider`

Default
rule:admin_required or (role:reader and system_scope:all)

Operations

- **GET** /v3/OS-FEDERATION/identity_providers/{idp_id}
- **HEAD** /v3/OS-FEDERATION/identity_providers/{idp_id}

Scope Types

- **system**
- **project**

Get identity provider.

`identity:update_identity_provider`

Default
rule:admin_required

Operations

- **PATCH** /v3/OS-FEDERATION/identity_providers/{idp_id}

Scope Types

- **system**
- **project**

Update identity provider.

`identity:delete_identity_provider`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/OS-FEDERATION/identity_providers/{idp_id}`

Scope Types

- **system**
- **project**

Delete identity provider.

`identity:get_implied_role`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/roles/{prior_role_id}/implies/{implied_role_id}`

Scope Types

- **system**
- **project**

Get information about an association between two roles. When a relationship exists between a prior role and an implied role and the prior role is assigned to a user, the user also assumes the implied role.

`identity:list_implied_roles`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/roles/{prior_role_id}/implies`
- **HEAD** `/v3/roles/{prior_role_id}/implies`

Scope Types

- **system**
- **project**

List associations between two roles. When a relationship exists between a prior role and an implied role and the prior role is assigned to a user, the user also assumes the implied role. This will return all the implied roles that would be assumed by the user who gets the specified prior role.

`identity:create_implied_role`

Default
`rule:admin_required`

Operations

- **PUT** `/v3/roles/{prior_role_id}/implies/{implied_role_id}`

Scope Types

- **system**
- **project**

Create an association between two roles. When a relationship exists between a prior role and an implied role and the prior role is assigned to a user, the user also assumes the implied role.

`identity:delete_implied_role`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/roles/{prior_role_id}/implies/{implied_role_id}`

Scope Types

- **system**
- **project**

Delete the association between two roles. When a relationship exists between a prior role and an implied role and the prior role is assigned to a user, the user also assumes the implied role. Removing the association will cause that effect to be eliminated.

`identity:list_role_inference_rules`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/role_inferences`
- **HEAD** `/v3/role_inferences`

Scope Types

- **system**
- **project**

List all associations between two roles in the system. When a relationship exists between a prior role and an implied role and the prior role is assigned to a user, the user also assumes the implied role.

`identity:check_implied_role`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **HEAD** `/v3/roles/{prior_role_id}/implies/{implied_role_id}`

Scope Types

- **system**
- **project**

Check an association between two roles. When a relationship exists between a prior role and an implied role and the prior role is assigned to a user, the user also assumes the implied role.

`identity:get_limit_model`

Default
<empty string>

Operations

- **GET** `/v3/limits/model`
- **HEAD** `/v3/limits/model`

Scope Types

- **system**
- **domain**
- **project**

Get limit enforcement model.

`identity:get_limit`

Default
`rule:admin_required or (role:reader and system_scope:all) or (domain_id:%(target.limit.domain.id)s or domain_id:%(target.limit.project.domain_id)s) or (project_id:%(target.limit.project_id)s and not None:%(target.limit.project_id)s)`

Operations

- **GET** `/v3/limits/{limit_id}`
- **HEAD** `/v3/limits/{limit_id}`

Scope Types

- **system**
- **domain**
- **project**

Show limit details.

`identity:list_limits`

Default
<empty string>

Operations

- **GET** `/v3/limits`
- **HEAD** `/v3/limits`

Scope Types

- **system**
- **domain**
- **project**

List limits.

`identity:create_limits`

Default
`rule:admin_required`

Operations

- **POST** `/v3/limits`

Scope Types

- **system**
- **project**

Create limits.

`identity:update_limit`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/limits/{limit_id}`

Scope Types

- **system**
- **project**

Update limit.

`identity:delete_limit`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/limits/{limit_id}`

Scope Types

- **system**
- **project**

Delete limit.

`identity:create_mapping`

Default
`rule:admin_required`

Operations

- **PUT** `/v3/OS-FEDERATION/mappings/{mapping_id}`

Scope Types

- **system**
- **project**

Create a new federated mapping containing one or more sets of rules.

`identity:get_mapping`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-FEDERATION/mappings/{mapping_id}`
- **HEAD** `/v3/OS-FEDERATION/mappings/{mapping_id}`

Scope Types

- **system**
- **project**

Get a federated mapping.

`identity:list_mappings`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-FEDERATION/mappings`
- **HEAD** `/v3/OS-FEDERATION/mappings`

Scope Types

- **system**
- **project**

List federated mappings.

`identity:delete_mapping`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/OS-FEDERATION/mappings/{mapping_id}`

Scope Types

- **system**
- **project**

Delete a federated mapping.

`identity:update_mapping`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/OS-FEDERATION/mappings/{mapping_id}`

Scope Types

- **system**
- **project**

Update a federated mapping.

`identity:get_policy`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/policies/{policy_id}`

Scope Types

- **system**
- **project**

Show policy details.

`identity:list_policies`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/policies`

Scope Types

- **system**
- **project**

List policies.

`identity:create_policy`

Default
`rule:admin_required`

Operations

- **POST** `/v3/policies`

Scope Types

- **system**
- **project**

Create policy.

`identity:update_policy`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/policies/{policy_id}`

Scope Types

- **system**
- **project**

Update policy.

`identity:delete_policy`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/policies/{policy_id}`

Scope Types

- **system**
- **project**

Delete policy.

`identity:create_policy_association_for_endpoint`

Default
`rule:admin_required`

Operations

- **PUT** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/endpoints/{endpoint_id}`

Scope Types

- **system**
- **project**

Associate a policy to a specific endpoint.

`identity:check_policy_association_for_endpoint`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/endpoints/{endpoint_id}`
- **HEAD** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/endpoints/{endpoint_id}`

Scope Types

- **system**
- **project**

Check policy association for endpoint.

`identity:delete_policy_association_for_endpoint`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/endpoints/{endpoint_id}`

Scope Types

- **system**
- **project**

Delete policy association for endpoint.

`identity:create_policy_association_for_service`

Default
`rule:admin_required`

Operations

- **PUT** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/services/{service_id}`

Scope Types

- **system**
- **project**

Associate a policy to a specific service.

`identity:check_policy_association_for_service`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/services/{service_id}`
- **HEAD** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/services/{service_id}`

Scope Types

- **system**
- **project**

Check policy association for service.

`identity:delete_policy_association_for_service`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/services/{service_id}`

Scope Types

- **system**

- **project**

Delete policy association for service.

`identity:create_policy_association_for_region_and_service`

Default
`rule:admin_required`

Operations

- **PUT** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/services/{service_id}/regions/{region_id}`

Scope Types

- **system**
- **project**

Associate a policy to a specific region and service combination.

`identity:check_policy_association_for_region_and_service`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/services/{service_id}/regions/{region_id}`
- **HEAD** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/services/{service_id}/regions/{region_id}`

Scope Types

- **system**
- **project**

Check policy association for region and service.

`identity:delete_policy_association_for_region_and_service`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/policies/{policy_id}/OS-ENDPOINT-POLICY/services/{service_id}/regions/{region_id}`

Scope Types

- **system**
- **project**

Delete policy association for region and service.

`identity:get_policy_for_endpoint`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** /v3/endpoints/{endpoint_id}/OS-ENDPOINT-POLICY/policy
- **HEAD** /v3/endpoints/{endpoint_id}/OS-ENDPOINT-POLICY/policy

Scope Types

- **system**
- **project**

Get policy for endpoint.

`identity:list_endpoints_for_policy`

Default

```
rule:admin_required or (role:reader and system_scope:all)
```

Operations

- **GET** /v3/policies/{policy_id}/OS-ENDPOINT-POLICY/endpoints

Scope Types

- **system**
- **project**

List endpoints for policy.

`identity:get_project`

Default

```
(rule:admin_required) or (role:reader and system_scope:all)
or (role:reader and domain_id:%(target.project.domain_id)s) or
project_id:%(target.project.id)s
```

Operations

- **GET** /v3/projects/{project_id}

Scope Types

- **system**
- **domain**
- **project**

Show project details.

`identity:list_projects`

Default

```
(rule:admin_required) or (role:reader and system_scope:all) or
(role:reader and domain_id:%(target.domain_id)s)
```

Operations

- **GET** /v3/projects

Scope Types

- **system**

- **domain**
- **project**

List projects.

`identity:list_user_projects`

Default

```
(rule:admin_required) or (role:reader and system_scope:all)
or (role:reader and domain_id:%(target.user.domain_id)s) or
user_id:%(target.user.id)s
```

Operations

- **GET** `/v3/users/{user_id}/projects`

Scope Types

- **system**
- **domain**
- **project**

List projects for user.

`identity:create_project`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
project.domain_id)s)
```

Operations

- **POST** `/v3/projects`

Scope Types

- **system**
- **domain**
- **project**

Create project.

`identity:update_project`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
project.domain_id)s)
```

Operations

- **PATCH** `/v3/projects/{project_id}`

Scope Types

- **system**
- **domain**
- **project**

Update project.

`identity:delete_project`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
project.domain_id)s)
```

Operations

- **DELETE** /v3/projects/{project_id}

Scope Types

- **system**
- **domain**
- **project**

Delete project.

`identity:list_project_tags`

Default

```
(rule:admin_required) or (role:reader and system_scope:all)
or (role:reader and domain_id:%(target.project.domain_id)s) or
project_id:%(target.project.id)s
```

Operations

- **GET** /v3/projects/{project_id}/tags
- **HEAD** /v3/projects/{project_id}/tags

Scope Types

- **system**
- **domain**
- **project**

List tags for a project.

`identity:get_project_tag`

Default

```
(rule:admin_required) or (role:reader and system_scope:all)
or (role:reader and domain_id:%(target.project.domain_id)s) or
project_id:%(target.project.id)s
```

Operations

- **GET** /v3/projects/{project_id}/tags/{value}
- **HEAD** /v3/projects/{project_id}/tags/{value}

Scope Types

- **system**
- **domain**
- **project**

Check if project contains a tag.

`identity:update_project_tags`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
project.domain_id)s)
```

Operations

- **PUT** `/v3/projects/{project_id}/tags`

Scope Types

- **system**
- **domain**
- **project**

Replace all tags on a project with the new set of tags.

`identity:create_project_tag`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
project.domain_id)s)
```

Operations

- **PUT** `/v3/projects/{project_id}/tags/{value}`

Scope Types

- **system**
- **domain**
- **project**

Add a single tag to a project.

`identity:delete_project_tags`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
project.domain_id)s)
```

Operations

- **DELETE** `/v3/projects/{project_id}/tags`

Scope Types

- **system**
- **domain**
- **project**

Remove all tags from a project.

`identity:delete_project_tag`

Default

```
(rule:admin_required) or (role:manager and domain_id:%(target.
project.domain_id)s)
```

Operations

- **DELETE** /v3/projects/{project_id}/tags/{value}

Scope Types

- **system**
- **domain**
- **project**

Delete a specified tag from project.

`identity:list_projects_for_endpoint`

Default
rule:admin_required or (role:reader and system_scope:all)

Operations

- **GET** /v3/OS-EP-FILTER/endpoints/{endpoint_id}/projects

Scope Types

- **system**
- **project**

List projects allowed to access an endpoint.

`identity:add_endpoint_to_project`

Default
rule:admin_required

Operations

- **PUT** /v3/OS-EP-FILTER/projects/{project_id}/endpoints/{endpoint_id}

Scope Types

- **system**
- **project**

Allow project to access an endpoint.

`identity:check_endpoint_in_project`

Default
rule:admin_required or (role:reader and system_scope:all)

Operations

- **GET** /v3/OS-EP-FILTER/projects/{project_id}/endpoints/{endpoint_id}
- **HEAD** /v3/OS-EP-FILTER/projects/{project_id}/endpoints/{endpoint_id}

Scope Types

- **system**

- **project**

Check if a project is allowed to access an endpoint.

`identity:list_endpoints_for_project`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-EP-FILTER/projects/{project_id}/endpoints`

Scope Types

- **system**
- **project**

List the endpoints a project is allowed to access.

`identity:remove_endpoint_from_project`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/OS-EP-FILTER/projects/{project_id}/endpoints/{endpoint_id}`

Scope Types

- **system**
- **project**

Remove access to an endpoint from a project that has previously been given explicit access.

`identity:create_protocol`

Default
`rule:admin_required`

Operations

- **PUT** `/v3/OS-FEDERATION/identity_providers/{idp_id}/protocols/{protocol_id}`

Scope Types

- **system**
- **project**

Create federated protocol.

`identity:update_protocol`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/OS-FEDERATION/identity_providers/{idp_id}/protocols/{protocol_id}`

Scope Types

- **system**
- **project**

Update federated protocol.

`identity:get_protocol`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-FEDERATION/identity_providers/{idp_id}/protocols/{protocol_id}`

Scope Types

- **system**
- **project**

Get federated protocol.

`identity:list_protocols`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-FEDERATION/identity_providers/{idp_id}/protocols`

Scope Types

- **system**
- **project**

List federated protocols.

`identity:delete_protocol`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/OS-FEDERATION/identity_providers/{idp_id}/protocols/{protocol_id}`

Scope Types

- **system**
- **project**

Delete federated protocol.

`identity:get_region`

Default
<empty string>

Operations

- **GET** `/v3/regions/{region_id}`
- **HEAD** `/v3/regions/{region_id}`

Scope Types

- **system**
- **domain**
- **project**

Show region details.

`identity:list_regions`

Default
<empty string>

Operations

- **GET** `/v3/regions`
- **HEAD** `/v3/regions`

Scope Types

- **system**
- **domain**
- **project**

List regions.

`identity:create_region`

Default
`rule:admin_required`

Operations

- **POST** `/v3/regions`
- **PUT** `/v3/regions/{region_id}`

Scope Types

- **system**
- **project**

Create region.

`identity:update_region`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/regions/{region_id}`

Scope Types

- **system**
- **project**

Update region.

`identity:delete_region`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/regions/{region_id}`

Scope Types

- **system**
- **project**

Delete region.

`identity:get_registered_limit`

Default
<empty string>

Operations

- **GET** `/v3/registered_limits/{registered_limit_id}`
- **HEAD** `/v3/registered_limits/{registered_limit_id}`

Scope Types

- **system**
- **domain**
- **project**

Show registered limit details.

`identity:list_registered_limits`

Default
<empty string>

Operations

- **GET** `/v3/registered_limits`
- **HEAD** `/v3/registered_limits`

Scope Types

- **system**
- **domain**
- **project**

List registered limits.

`identity:create_registered_limits`

Default
`rule:admin_required`

Operations

- **POST** `/v3/registered_limits`

Scope Types

- **system**
- **project**

Create registered limits.

`identity:update_registered_limit`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/registered_limits/{registered_limit_id}`

Scope Types

- **system**
- **project**

Update registered limit.

`identity:delete_registered_limit`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/registered_limits/{registered_limit_id}`

Scope Types

- **system**
- **project**

Delete registered limit.

`identity:list_revoke_events`

Default
`rule:service_or_admin`

Operations

- **GET** `/v3/OS-REVOKE/events`

Scope Types

- **system**
- **project**

List revocation events.

`identity:get_role`

Default

```
(rule:admin_required or (role:reader and system_scope:all)) or
(role:manager and rule:domain_managed_target_role)
```

Operations

- **GET** /v3/roles/{role_id}
- **HEAD** /v3/roles/{role_id}

Scope Types

- **system**
- **domain**
- **project**

Show role details.

`identity:list_roles`

Default

```
(rule:admin_required or (role:reader and system_scope:all)) or
(role:manager and not domain_id:None)
```

Operations

- **GET** /v3/roles
- **HEAD** /v3/roles

Scope Types

- **system**
- **domain**
- **project**

List roles.

`identity:create_role`

Default

```
rule:admin_required
```

Operations

- **POST** /v3/roles

Scope Types

- **system**
- **project**

Create role.

`identity:update_role`

Default

```
rule:admin_required
```

Operations

- **PATCH** /v3/roles/{role_id}

Scope Types

- **system**
- **project**

Update role.

`identity:delete_role`

Default
rule:admin_required

Operations

- **DELETE** /v3/roles/{role_id}

Scope Types

- **system**
- **project**

Delete role.

`identity:get_domain_role`

Default
rule:admin_required or (role:reader and system_scope:all)

Operations

- **GET** /v3/roles/{role_id}
- **HEAD** /v3/roles/{role_id}

Scope Types

- **system**
- **project**

Show domain role.

`identity:list_domain_roles`

Default
rule:admin_required or (role:reader and system_scope:all)

Operations

- **GET** /v3/roles?domain_id={domain_id}
- **HEAD** /v3/roles?domain_id={domain_id}

Scope Types

- **system**
- **project**

List domain roles.

`identity:create_domain_role`

Default
`rule:admin_required`

Operations

- **POST** `/v3/roles`

Scope Types

- **system**
- **project**

Create domain role.

`identity:update_domain_role`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/roles/{role_id}`

Scope Types

- **system**
- **project**

Update domain role.

`identity:delete_domain_role`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/roles/{role_id}`

Scope Types

- **system**
- **project**

Delete domain role.

`identity:list_role_assignments`

Default
`(rule:admin_required) or (role:reader and system_scope:all) or (role:reader and domain_id:%(target.domain_id)s)`

Operations

- **GET** `/v3/role_assignments`
- **HEAD** `/v3/role_assignments`

Scope Types

- **system**
- **domain**

- **project**

List role assignments.

`identity:list_role_assignments_for_tree`

Default

```
(rule:admin_required) or (role:reader and system_scope:all) or
(role:reader and domain_id:%(target.domain_id)s)
```

Operations

- **GET** `/v3/role_assignments?include_subtree`
- **HEAD** `/v3/role_assignments?include_subtree`

Scope Types

- **system**
- **domain**
- **project**

List all role assignments for a given tree of hierarchical projects.

`identity:s3tokens_validate`

Default

```
rule:service_or_admin
```

Operations

- **POST** `/v3/s3tokens`

Scope Types

- **system**
- **domain**
- **project**

Validate S3 credentials and create a Keystone token. Restricted to service users or administrators to prevent exploitation via presigned URLs.

`identity:ec2tokens_validate`

Default

```
rule:service_or_admin
```

Operations

- **POST** `/v3/ec2tokens`

Scope Types

- **system**
- **domain**
- **project**

Validate EC2 credentials and create a Keystone token. Restricted to service users or administrators.

`identity:get_service`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/services/{service_id}`

Scope Types

- **system**
- **project**

Show service details.

`identity:list_services`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/services`

Scope Types

- **system**
- **project**

List services.

`identity:create_service`

Default
`rule:admin_required`

Operations

- **POST** `/v3/services`

Scope Types

- **system**
- **project**

Create service.

`identity:update_service`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/services/{service_id}`

Scope Types

- **system**
- **project**

Update service.

`identity:delete_service`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/services/{service_id}`

Scope Types

- **system**
- **project**

Delete service.

`identity:create_service_provider`

Default
`rule:admin_required`

Operations

- **PUT** `/v3/OS-FEDERATION/service_providers/{service_provider_id}`

Scope Types

- **system**
- **project**

Create federated service provider.

`identity:list_service_providers`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-FEDERATION/service_providers`
- **HEAD** `/v3/OS-FEDERATION/service_providers`

Scope Types

- **system**
- **project**

List federated service providers.

`identity:get_service_provider`

Default
`rule:admin_required or (role:reader and system_scope:all)`

Operations

- **GET** `/v3/OS-FEDERATION/service_providers/{service_provider_id}`
- **HEAD** `/v3/OS-FEDERATION/service_providers/{service_provider_id}`

Scope Types

- **system**
- **project**

Get federated service provider.

`identity:update_service_provider`

Default
`rule:admin_required`

Operations

- **PATCH** `/v3/OS-FEDERATION/service_providers/{service_provider_id}`

Scope Types

- **system**
- **project**

Update federated service provider.

`identity:delete_service_provider`

Default
`rule:admin_required`

Operations

- **DELETE** `/v3/OS-FEDERATION/service_providers/{service_provider_id}`

Scope Types

- **system**
- **project**

Delete federated service provider.

`identity:revocation_list`

Default
`rule:service_or_admin`

Operations

- **GET** `/v3/auth/tokens/OS-PKI/revoked`

Scope Types

- **system**
- **project**

List revoked PKI tokens.

`identity:check_token`

Default
`rule:admin_required or (role:reader and system_scope:all) or rule:token_subject`

Operations

- **HEAD** /v3/auth/tokens

Scope Types

- **system**
- **domain**
- **project**

Check a token.

`identity:validate_token`

Default

```
rule:admin_required or (role:reader and system_scope:all) or
rule:service_role or rule:token_subject
```

Operations

- **GET** /v3/auth/tokens

Scope Types

- **system**
- **domain**
- **project**

Validate a token.

`identity:revoke_token`

Default

```
rule:admin_required or rule:token_subject
```

Operations

- **DELETE** /v3/auth/tokens

Scope Types

- **system**
- **domain**
- **project**

Revoke a token.

`identity:create_trust`

Default

```
user_id:%(trust.trustor_user_id)s
```

Operations

- **POST** /v3/OS-TRUST/trusts

Scope Types

- **project**

Create trust.

`identity:list_trusts`

Default

```
rule:admin_required or (role:reader and system_scope:all)
```

Operations

- **GET** /v3/OS-TRUST/trusts
- **HEAD** /v3/OS-TRUST/trusts

Scope Types

- **system**
- **project**

List trusts.

`identity:list_trusts_for_trustor`

Default

```
(rule:admin_required) or (role:reader and system_scope:all or
user_id:%(target.trust.trustor_user_id)s)
```

Operations

- **GET** /v3/OS-TRUST/trusts?trustor_user_id={trustor_user_id}
- **HEAD** /v3/OS-TRUST/trusts?trustor_user_id={trustor_user_id}

Scope Types

- **system**
- **project**

List trusts for trustor.

`identity:list_trusts_for_trustee`

Default

```
(rule:admin_required) or (role:reader and system_scope:all or
user_id:%(target.trust.trustee_user_id)s)
```

Operations

- **GET** /v3/OS-TRUST/trusts?trustee_user_id={trustee_user_id}
- **HEAD** /v3/OS-TRUST/trusts?trustee_user_id={trustee_user_id}

Scope Types

- **system**
- **project**

List trusts for trustee.

`identity:list_roles_for_trust`

Default

```
(rule:admin_required) or (role:reader and system_scope:all or
user_id:%(target.trust.trustor_user_id)s or user_id:%(target.
trust.trustee_user_id)s)
```

Operations

- **GET** /v3/OS-TRUST/trusts/{trust_id}/roles
- **HEAD** /v3/OS-TRUST/trusts/{trust_id}/roles

Scope Types

- **system**
- **project**

List roles delegated by a trust.

`identity:get_role_for_trust`

Default

```
(rule:admin_required) or (role:reader and system_scope:all or
user_id:%(target.trust.trustor_user_id)s or user_id:%(target.
trust.trustee_user_id)s)
```

Operations

- **GET** /v3/OS-TRUST/trusts/{trust_id}/roles/{role_id}
- **HEAD** /v3/OS-TRUST/trusts/{trust_id}/roles/{role_id}

Scope Types

- **system**
- **project**

Check if trust delegates a particular role.

`identity:delete_trust`

Default

```
rule:admin_required or user_id:%(target.trust.
trustor_user_id)s
```

Operations

- **DELETE** /v3/OS-TRUST/trusts/{trust_id}

Scope Types

- **system**
- **project**

Revoke trust.

`identity:get_trust`

Default

```
(rule:admin_required) or (role:reader and system_scope:all or
user_id:%(target.trust.trustor_user_id)s or user_id:%(target.
trust.trustee_user_id)s)
```

Operations

- **GET** /v3/OS-TRUST/trusts/{trust_id}
- **HEAD** /v3/OS-TRUST/trusts/{trust_id}

Scope Types

- **system**
- **project**

Get trust.

`identity:get_user`

Default

```
(rule:admin_required) or (role:reader and system_scope:all) or
(role:reader and token.domain.id:%(target.user.domain_id)s) or
user_id:%(target.user.id)s
```

Operations

- **GET** `/v3/users/{user_id}`
- **HEAD** `/v3/users/{user_id}`

Scope Types

- **system**
- **domain**
- **project**

Show user details.

`identity:list_users`

Default

```
(rule:admin_required) or (role:reader and system_scope:all) or
(role:reader and domain_id:%(target.domain_id)s)
```

Operations

- **GET** `/v3/users`
- **HEAD** `/v3/users`

Scope Types

- **system**
- **domain**
- **project**

List users.

`identity:list_projects_for_user`

Default

<empty string>

Operations

- **GET** “ /v3/auth/projects“

List all projects a user has access to via role assignments.

`identity:list_domains_for_user`

Default

<empty string>

Operations

- **GET** /v3/auth/domains

List all domains a user has access to via role assignments.

`identity:create_user`

Default

```
(rule:admin_required) or (role:manager and token.domain.
id:%(target.user.domain_id)s)
```

Operations

- **POST** /v3/users

Scope Types

- **system**
- **domain**
- **project**

Create a user.

`identity:update_user`

Default

```
(rule:admin_required) or (role:manager and token.domain.
id:%(target.user.domain_id)s)
```

Operations

- **PATCH** /v3/users/{user_id}

Scope Types

- **system**
- **domain**
- **project**

Update a user, including administrative password resets.

`identity:delete_user`

Default

```
(rule:admin_required) or (role:manager and token.domain.
id:%(target.user.domain_id)s)
```

Operations

- **DELETE** /v3/users/{user_id}

Scope Types

- **system**
- **domain**
- **project**

Delete a user.