
Barbican Documentation

Release 11.0.1.dev27

OpenStack Foundation

Apr 02, 2024

CONTENTS

1	What is Barbican?	3
2	API Guide	5
2.1	Cloud Administrator Guide - Key Manager service	5
2.2	CLI Reference	10
2.3	Key Manager service	11
2.4	Setting up Barbican	31
2.5	Barbican for Developers	102
2.6	Barbican API Documentation	135
3	Sample Files	195
3.1	Barbican Sample Configuration File	195
3.2	Barbican Sample Policy	195
4	Indices and tables	203
	Python Module Index	205
	Index	207



WHAT IS BARBICAN?

Barbican is the OpenStack Key Manager service. It provides secure storage, provisioning and management of secret data. This includes keying material such as Symmetric Keys, Asymmetric Keys, Certificates and raw binary data.

If you're trying to learn how to use Barbican, you can start by reading about [Secrets in the Barbican API Guide](#).

Once you're comfortable working with secrets you can dig into the rest of the API.

2.1 Cloud Administrator Guide - Key Manager service

The Key Manager service, code-named Barbican, is the default secret storage service for OpenStack. The service provides secure storage, provisioning and management of secrets.

2.1.1 Access Control

Role Based Access Control (RBAC)

Like many other services, the Key Manager service supports the protection of its APIs by enforcing policy rules defined in a policy file. The Key Manager service stores a reference to a policy JSON file in its configuration file, `/etc/barbican/barbican.conf`. Typically this file is named `policy.json` and it is stored in `/etc/barbican/policy.json`.

Each Key Manager API call has a line in the policy file that dictates which level of access applies:

```
API_NAME: RULE_STATEMENT or MATCH_STATEMENT
```

where `RULE_STATEMENT` can be another `RULE_STATEMENT` or a `MATCH_STATEMENT`:

```
RULE_STATEMENT: RULE_STATEMENT or MATCH_STATEMENT
```

`MATCH_STATEMENT` is a set of identifiers that must match between the token provided by the caller of the API and the parameters or target entities of the API in question. For example:

```
"secrets:post": "role:admin or role:creator"
```

indicates that to create a new secret via a POST request, you must have either the admin or creator role in your token.

Warning: The Key Manager service scopes the ownership of a secret at the project level. This means that many calls in the API will perform an additional check to ensure that the `project_id` of the token matches the `project_id` stored as the secret owner.

Default Policy

The policy engine in OpenStack is very flexible and allows for customized policies that make sense for your particular cloud. The Key Manager service comes with a sample `policy.json` file which can be used as the starting point for a customized policy. The sample policy defines 5 distinct roles:

key-manager:service-admin The cloud administrator in charge of the Key Manager service. This user has access to all management APIs like the project-quotas.

admin Project administrator. This user has full access to all resources owned by the project for which the admin role is scoped.

creator Users with this role are allowed to create new resources and can also delete resources which are owned by the project for which the creator role is scoped. They are also allowed full access to existing secrets owned by the project in scope.

observer Users with this role are allowed to access to existing resources but are not allowed to upload new secrets or delete existing secrets.

audit Users with this role are only allowed access to the resource metadata. So users with this role are unable to decrypt secrets.

Access Control List API

There are some limitations that result from scoping ownership of a secret at the project level. For example, it is not possible to grant a user access to a single secret, as granting a role on a project would allow access to all all secrets owned by that project.

Additionally, there is no easy way to upload a private secret (i.e. a secret that only you have access to) without creating a new project for which only you have roles assigned on it.

To address these limitations the Key Manager service includes an Access Control List (ACL) API. For full details see the [ACL API User Guide](#)

2.1.2 Barbican Service Management Utility

Description

`barbican-manage` is a utility that is used to control the barbican key manager service database and Hardware Secure Module (HSM) plugin device. Use cases include migrating the secret database or generating a Master Key Encryption Key (MKEK) in the HSM. This command set should only be executed by a user with admin privileges.

Options

The standard pattern for executing a `barbican-manage` command is:

```
barbican-manage <category> <command> [<args>]
```

Running `barbican-manage` without arguments shows a list of available command categories. Currently, there are 2 supported categories: `db` and `hsm`.

Running with a category argument shows a list of commands in that category:

- `barbican-manage db --help`
- `barbican-manage hsm --help`
- `barbican-manage --version` shows the version number of barbican service.

The following sections describe the available categories and arguments for `barbican-manage`.

Barbican Database

Warning: Before executing `barbican-manage db` commands, make sure you are familiar with [Database Migration](#) first.

```
barbican-manage db revision [--db-url] [--message] [--autogenerate]
```

Create a new database version file.

```
barbican-manage db upgrade [--db-url] [--version]
```

Upgrade to a future version database.

```
barbican-manage db history [--db-url] [--verbose]
```

Show database changeset history.

```
barbican-manage db current [--db-url] [--verbose]
```

Show current revision of database.

```
barbican-manage db clean [--db-url] [--verbose] [--min-days]
[--clean-unassociated-projects] [--soft-delete-expired-secrets]
[--log-file]
```

Clean up soft deletions in the database. More documentation can be found here: [Database Cleaning](#)

```
barbican-manage db sync_secret_stores [--db-url] [--verbose]
[--log-file]
```

Synchronize the `secret_store` database table with the configuration in `barbican.conf`. This is useful when multiple secret stores are enabled and new secret stores have been enabled.

Barbican PKCS11/HSM

```
barbican-manage hsm gen_mkek [--library-path] [--passphrase]
[--slot-id] [--label] [--length]
```

Create a new Master key encryption key in HSM. This MKEK will be used to encrypt all project key encryption keys. Its label must be unique.

```
barbican-manage hsm gen_hmac [--library-path] [--passphrase]
[--slot-id] [--label] [--length]
```

Create a new Master HMAC key in HSM. This HMAC key will be used to generate an authentication tag of encrypted project key encryption keys. Its label must be unique.

```
barbican-manage hsm rewrap_pkek [--dry-run]
```

Rewrap project key encryption keys after rotating to new MKEK and/or HMAC key(s) in HSM. The new MKEK and HMAC key should have already been generated using the above commands. The user will have to configure new MKEK and HMAC key labels in `/etc/barbican/barbican.conf` and restart barbican server before executing this command.

2.1.3 Database Cleaning

Entries in the Barbican database are soft deleted and can build up over time. These entries can be cleaned up with the clean up command. The command can be used with a cron job to clean the database automatically on intervals.

Commands

The command `barbican-manage db clean` can be used to clean up the database. By default, it will remove soft deletions that are at least 90 days old since deletion

`barbican-manage db clean --min-days 180` (`-m`) will go through the database and remove soft deleted entries that are at least 90 days old since deletion. The default value is 90 days. Passing a value of `--min-days 0` will delete all soft-deleted entries up to today.

`barbican-manage db clean --clean-unassociated-projects` (`-p`) will go through the database and remove projects that have no associated resources. The default value is False.

`barbican-manage db clean --soft-delete-expired-secrets` (`-e`) will go through the database and soft delete any secrets that are past their expiration date. The default value is False. If `-e` is used along with `---min-days 0` then all the expired secrets will be hard deleted.

`barbican-manage db clean --verbose` (`-V`) will print more information out into the terminal.

`barbican-manage db clean --log-file` (`-L`) will set the log file location. The creation of the log may fail if the user running the command does not have access to the log file location or if the target directory does not exist. The default value for `log_file` can be found in `/etc/barbican/barbican.conf`. The log will contain the verbose output from the command.

Cron Job

A cron job can be created on linux systems to run at a given interval to clean the barbican database.

Crontab

1. Start the crontab editor `crontab -e` with the user that runs the clean up command
2. Edit the crontab section to run the command at a given interval. `<minute 0-59> <hour 0-23, 0=midnight> <day 1-31> <month 1-12> <weekday 0-6, 0=Sunday> clean up command`

Crontab Examples

``00 00 * * * barbican-manage db clean -p -e`` -Runs a job everyday at midnight which will remove soft deleted entries that 90 days old since soft deletion, will clean unassociated projects, and will soft delete secrets that are expired.

``00 03 01 * * barbican-manage db clean -m 30`` -Runs a job every month at 3AM which will remove soft deleted entries that are at least 30 days old since deletion.

``05 01 07 * 6 barbican-manage db clean -m 180 -p -e -L /tmp/
barbican-clean-command.log`` -Runs a job every month at 1:05AM on the 7th day of the month and every Saturday. Entries that are 180 days old since soft deletion will be removed from the database. Unassociated projects will be removed. Expired secrets will be soft deleted. The log file will be saved to ``/tmp/barbican-clean-command.log``

2.1.4 Key Manager Service Upgrade Guide

This document outlines several steps and notes for operators to reference when upgrading their barbican from previous versions of OpenStack.

Plan to Upgrade

- The [release notes](#) should be read carefully before upgrading the barbican services. Starting with the Mitaka release, specific upgrade steps and considerations are well-documented in the release notes.
- Upgrades are only supported between sequential releases.
- When upgrading barbican, the following steps should be followed:
 1. Destroy all barbican services
 2. Upgrade source code to the next release
 3. Upgrade barbican database to the next release

```
barbican-db-manage upgrade
```

4. Start barbican services

Upgrade from Newton to Ocata

The `barbican-api-paste.ini` configuration file for the paste pipeline was updated to add the `http_proxy_to_wsgi` middleware. It can be used to help barbican respond with the correct URL refs when its put behind a TLS proxy (such as HAProxy). This middleware is disabled by default, but can be enabled via a configuration option in the `oslo_middleware` group.

See [Ocata release notes](#).

Upgrade from Mitaka to Newton

There are no extra instructions that should be noted for this upgrade.

See [Newton release notes](#).

Upgrade from Liberty to Mitaka

The Metadata API requires an update to the Database Schema. Existing deployments that are being upgraded to Mitaka should use the `barbican-manage` utility to update the schema.

If you are upgrading from previous version of barbican that uses the PKCS#11 Cryptographic Plugin driver, you will need to run the migration script.

```
python barbican/cmd/pkcs11_migrate_kek_signatures.py
```

See [Mitaka release notes](#).

2.2 CLI Reference

2.2.1 barbican-status

Synopsis

```
barbican-status <category> <command> [<args>]
```

Description

barbican-status is a tool that provides routines for checking the status of a Barbican deployment.

Options

The standard pattern for executing a **barbican-status** command is:

```
barbican-status <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
barbican-status
```

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:

```
barbican-status upgrade
```

These sections describe the available categories and arguments for **barbican-status**.

Upgrade

barbican-status upgrade check Performs a release-specific readiness check before restarting services with new code. This command expects to have complete configuration and access to databases and services.

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

8.0.0 (Stein)

- Placeholder to be filled in with checks as they are added in Stein.

2.3 Key Manager service

2.3.1 Key Manager service overview

The Key Manager service provides secure storage, provisioning and management of secrets, such as passwords, encryption keys, etc.

The Key Manager service consists of the following components:

barbican-api service Provides an OpenStack-native RESTful API that supports provisioning and managing Barbican secrets.

barbican-worker service Provides an Openstack RPC interface that interacts with `barbican-api` and reads from the barbican message queue. Supports the fulfillment of Barbican orders.

barbican-keystone-listener service Listens to messages from the Keystone notification service. Used to manage the representation of Keystone projects in the Barbican database when projects are deleted.

2.3.2 Install and configure

This section describes how to install and configure the Key Manager service, code-named barbican, on the controller node.

This section assumes that you already have a working OpenStack environment with at least the Identity Service (keystone) installed.

For simplicity, this configuration stores secrets on the local file system.

Note that installation and configuration vary by distribution.

Install and configure for openSUSE and SUSE Linux Enterprise

This section describes how to install and configure the Key Manager service for openSUSE Leap 42.2 and SUSE Linux Enterprise Server 12 SP2.

Prerequisites

Before you install and configure the Key Manager service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

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

```
# mysql
```

- Create the barbican database:

```
CREATE DATABASE barbican;
```

- Grant proper access to the barbican database:

```
GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@'localhost' \
  IDENTIFIED BY 'BARBICAN_DBPASS';
GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@'%' \
  IDENTIFIED BY 'BARBICAN_DBPASS';
```

Replace `BARBICAN_DBPASS` with a suitable password.

- Exit the database access client.

```
exit;
```

2. Source the admin credentials to gain access to admin-only CLI commands:

```
$ source admin-openrc
```

3. To create the service credentials, complete these steps:

- Create the barbican user:

```
$ openstack user create --domain default --password-prompt_
↵barbican
```


- Add the admin role to the barbican user:

```
$ openstack role add --project service --user barbican admin
```

- Create the creator role:

```
$ openstack role create creator
```

- Add the creator role to the barbican user:

```
$ openstack role add --project service --user barbican creator
```

- Create the barbican service entities:

```
$ openstack service create --name barbican --description "Key_↵
↵Manager" key-manager
```

4. Create the Key Manager service API endpoints:

```
$ openstack endpoint create --region RegionOne \
key-manager public http://controller:9311
$ openstack endpoint create --region RegionOne \
key-manager internal http://controller:9311
$ openstack endpoint create --region RegionOne \
key-manager admin http://controller:9311
```

Install and configure components

1. Install the packages:

```
# zypper install openstack-barbican-api openstack-barbican-keystone-
↵listener openstack-barbican-worker
```

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

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

```
[DEFAULT]
...
sql_connection = mysql+pymysql://barbican:BARBICAN_
↵DBPASS@controller/barbican
```

Replace `BARBICAN_DBPASS` with the password you chose for the Key Manager service database.

- In the `[DEFAULT]` section, configure RabbitMQ message queue access:

```
[DEFAULT]
...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

Replace `RABBIT_PASS` with the password you chose for the openstack account in RabbitMQ.

- In the `[keystone_authtoken]` section, configure Identity service access:

```
[keystone_authtoken]
...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = barbican
password = BARBICAN_PASS
```

Replace BARBICAN_PASS with the password you chose for the barbican user in the Identity service.

Note: Comment out or remove any other options in the [keystone_authtoken] section.

3. Populate the Key Manager service database:

If you wish the Key Manager service to automatically populate the database when the service is first started, set db_auto_create to True in the [DEFAULT] section. By default this will not be active and you can populate the database manually as below:

```
$ su -s /bin/sh -c "barbican-manage db upgrade" barbican
```

Note: Ignore any deprecation messages in this output.

4. Barbican has a plugin architecture which allows the deployer to store secrets in a number of different back-end secret stores. By default, Barbican is configured to store secrets in a basic file-based keystore. This key store is NOT safe for production use.

For a list of supported plugins and detailed instructions on how to configure them, see [Configure Secret Store Back-end](#)

Finalize installation

1. Copy the sample Apache vhost file into place:

```
# cp /etc/apache2/conf.d/barbican-api.conf.sample /etc/apache2/vhosts.d/
↪barbican-api.conf
```

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

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

Install and configure for Red Hat Enterprise Linux and CentOS

This section describes how to install and configure the Key Manager service for Red Hat Enterprise Linux 7 and CentOS 7.

Prerequisites

Before you install and configure the Key Manager service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

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

```
# mysql
```

- Create the barbican database:

```
CREATE DATABASE barbican;
```

- Grant proper access to the barbican database:

```
GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@'localhost' \
  IDENTIFIED BY 'BARBICAN_DBPASS';
GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@'%' \
  IDENTIFIED BY 'BARBICAN_DBPASS';
```

Replace `BARBICAN_DBPASS` with a suitable password.

- Exit the database access client.

```
exit;
```

2. Source the admin credentials to gain access to admin-only CLI commands:

```
$ source admin-openrc
```

3. To create the service credentials, complete these steps:

- Create the barbican user:

```
$ openstack user create --domain default --password-prompt _
↪barbican
```

- Add the admin role to the barbican user:

```
$ openstack role add --project service --user barbican admin
```

- Create the creator role:

```
$ openstack role create creator
```

- Add the creator role to the barbican user:

```
$ openstack role add --project service --user barbican creator
```

- Create the barbican service entities:

```
$ openstack service create --name barbican --description "Key_↵
↵Manager" key-manager
```

4. Create the Key Manager service API endpoints:

```
$ openstack endpoint create --region RegionOne \
key-manager public http://controller:9311
$ openstack endpoint create --region RegionOne \
key-manager internal http://controller:9311
$ openstack endpoint create --region RegionOne \
key-manager admin http://controller:9311
```

Install and configure components

1. Install the packages:

```
# yum install openstack-barbican-api
```

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

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

```
[DEFAULT]
...
sql_connection = mysql+pymysql://barbican:BARBICAN_↵
↵DBPASS@controller/barbican
```

Replace `BARBICAN_DBPASS` with the password you chose for the Key Manager service database.

- In the `[DEFAULT]` section, configure RabbitMQ message queue access:

```
[DEFAULT]
...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

Replace `RABBIT_PASS` with the password you chose for the openstack account in RabbitMQ.

- In the `[keystone_authtoken]` section, configure Identity service access:

```
[keystone_authtoken]
...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = barbican
password = BARBICAN_PASS
```

Replace `BARBICAN_PASS` with the password you chose for the barbican user in the Identity service.

Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

3. Populate the Key Manager service database:

If you wish the Key Manager service to automatically populate the database when the service is first started, set `db_auto_create` to `True` in the `[DEFAULT]` section. By default this will not be active and you can populate the database manually as below:

```
$ su -s /bin/sh -c "barbican-manage db upgrade" barbican
```

Note: Ignore any deprecation messages in this output.

4. Barbican has a plugin architecture which allows the deployer to store secrets in a number of different back-end secret stores. By default, Barbican is configured to store secrets in a basic file-based keystore. This key store is NOT safe for production use.

For a list of supported plugins and detailed instructions on how to configure them, see [Configure Secret Store Back-end](#)

Finalize installation

1. Create the `/etc/httpd/conf.d/wsgi-barbican.conf` file with the following content:

```
<VirtualHost [::1]:9311>
  ServerName controller

  ## Logging
  ErrorLog "/var/log/httpd/barbican_wsgi_main_error_ssl.log"
  LogLevel debug
  ServerSignature Off
  CustomLog "/var/log/httpd/barbican_wsgi_main_access_ssl.log"
  ↪combined

  WSGIApplicationGroup %{GLOBAL}
  WSGIDaemonProcess barbican-api display-name=barbican-api
  ↪group=barbican processes=2 threads=8 user=barbican
  WSGIProcessGroup barbican-api
  WSGIScriptAlias / "/usr/lib/python2.7/site-packages/barbican/api/
  ↪app.wsgi"
  WSGIPassAuthorization On
</VirtualHost>
```

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

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

Install and configure for Ubuntu

This section describes how to install and configure the Key Manager service for Ubuntu 14.04 (LTS).

Prerequisites

Before you install and configure the Key Manager service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

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

```
# mysql
```

- Create the barbican database:

```
CREATE DATABASE barbican;
```

- Grant proper access to the barbican database:

```
GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@'localhost' \
  IDENTIFIED BY 'BARBICAN_DBPASS';
GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@'%' \
  IDENTIFIED BY 'BARBICAN_DBPASS';
```

Replace `BARBICAN_DBPASS` with a suitable password.

- Exit the database access client.

```
exit;
```

2. Source the admin credentials to gain access to admin-only CLI commands:

```
$ source admin-openrc
```

3. To create the service credentials, complete these steps:

- Create the barbican user:

```
$ openstack user create --domain default --password-prompt \
↪barbican
```

- Add the admin role to the barbican user:

```
$ openstack role add --project service --user barbican admin
```

- Create the creator role:

```
$ openstack role create creator
```

- Add the creator role to the barbican user:

```
$ openstack role add --project service --user barbican creator
```

- Create the barbican service entities:

```
$ openstack service create --name barbican --description "Key_
↪Manager" key-manager
```

4. Create the Key Manager service API endpoints:

```
$ openstack endpoint create --region RegionOne \
key-manager public http://controller:9311
$ openstack endpoint create --region RegionOne \
key-manager internal http://controller:9311
$ openstack endpoint create --region RegionOne \
key-manager admin http://controller:9311
```

Install and configure components

1. Install the packages:

```
# apt-get update

# apt-get install barbican-api barbican-keystone-listener barbican-
↪worker
```

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

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

```
[DEFAULT]
...
sql_connection = mysql+pymysql://barbican:BARBICAN_
↪DBPASS@controller/barbican
```

Replace `BARBICAN_DBPASS` with the password you chose for the Key Manager service database.

- In the `[DEFAULT]` section, configure RabbitMQ message queue access:

```
[DEFAULT]
...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

Replace `RABBIT_PASS` with the password you chose for the openstack account in RabbitMQ.

- In the `[keystone_authtoken]` section, configure Identity service access:

```
[keystone_authtoken]
...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = barbican
password = BARBICAN_PASS
```

Replace `BARBICAN_PASS` with the password you chose for the barbican user in the Identity service.

Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

3. Populate the Key Manager service database:

If you wish the Key Manager service to automatically populate the database when the service is first started, set `db_auto_create` to `True` in the `[DEFAULT]` section. By default this will not be active and you can populate the database manually as below:

```
$ su -s /bin/sh -c "barbican-manage db upgrade" barbican
```

Note: Ignore any deprecation messages in this output.

4. Barbican has a plugin architecture which allows the deployer to store secrets in a number of different back-end secret stores. By default, Barbican is configured to store secrets in a basic file-based keystore. This key store is NOT safe for production use.

For a list of supported plugins and detailed instructions on how to configure them, see [Configure Secret Store Back-end](#)

Finalize installation

Restart the Key Manager services:

```
# service barbican-keystone-listener restart
# service barbican-worker restart
# service apache2 restart
```

Configure Secret Store Back-end

The Key Manager service has a plugin architecture that allows the deployer to store secrets in one or more secret stores. Secret stores can be software-based such as a software-only encryption mechanism, or hardware devices such as a hardware security module (HSM).

Secret Stores implement both the encryption mechanisms as well as the storage of the encrypted secrets.

This section compares all the plugins that are currently available and the security tradeoffs that need to be considered when deciding which pluing to use.

Simple Crypto Plugin

This back end plugin implements encryption using only software. The encrypted secrets are stored in the Barbican database.

This crypto plugin is configured by default in `/etc/barbican/barbican.conf`.

This plugin uses single symmetric key (kek - or key encryption key) - which is stored in plain text in the `/etc/barbican/barbican.conf` file to encrypt and decrypt all secrets.

Security	Master Key (KEK) stored in the configuration file
Maturity	Tested on every patch
Ease of Use	Simple to deploy Key rotation is disruptive (all secrets must be re-encrypted)
Scalability	Storage can be scaled in SQL DB Failover/HA is simple, just run more barbican-api instances High performance - Software crypto is fast
Cost	Free (as in beer)

Warning: This plugin stores its KEK in plain text in the configuration file, which will be present in any node running the *barbican-api* or *barbican-worker* services. Extreme care should be taken to prevent unauthorized access to these nodes. When using this plugin the KEK is the only thing protecting the secrets stored in the database.

The configuration for this plugin in `/etc/barbican/barbican.conf` is as follows:

```
# ===== Secret Store Plugin =====
[secretstore]
..
enabled_secretstore_plugins = store_crypto

# ===== Crypto plugin =====
[crypto]
..
enabled_crypto_plugins = simple_crypto

[simple_crypto_plugin]
# the kek should be a 32-byte value which is base64 encoded
kek = 'YWJjZGVmZ2hpamtsbW5vcHFyc3R1dnd4eXoxMjM0NTY='
```

PKCS#11 Crypto Plugin

This crypto plugin can be used to interface with a Hardware Security Module (HSM) using the PKCS#11 protocol.

Secrets are encrypted (and decrypted on retrieval) by a project specific Key Encryption Key (KEK), which resides in the HSM.

The configuration for this plugin in `/etc/barbican/barbican.conf`. Settings for some different HSMs are provided below:

Safenet

The PKCS#11 plugin configuration looks like:

```
# ===== Secret Store Plugin =====
[secretstore]
..
enabled_secretstore_plugins = store_crypto

[p11_crypto_plugin]
# Path to vendor PKCS11 library
library_path = '/usr/lib/libCryptoki2_64.so'

# Token serial number used to identify the token to be used.
↳Required
# when the device has multiple tokens with the same label. (string
# value)
token_serial_number = 12345678

# Token label used to identify the token to be used. Required
↳when
# token_serial_number is not specified. (string value)
#token_label = <None>

# Password to login to PKCS11 session
login = 'mypassword'

# Label to identify master KEK in the HSM (must not be the same
↳as HMAC label)
mkek_label = 'an_mkek'

# Length in bytes of master KEK
mkek_length = 32

# Label to identify HMAC key in the HSM (must not be the same as
↳MKEK label)
hmac_label = 'my_hmac_label'

# (Optional) HSM Slot ID that contains the token device to be
↳used.
# (integer value)
#slot_id = 1
```

(continues on next page)

(continued from previous page)

```
# Enable Read/Write session with the HSM?
# rw_session = True

# Length of Project KEKs to create
# pkek_length = 32

# How long to cache unwrapped Project KEKs
# pkek_cache_ttl = 900

# Max number of items in pkek cache
# pkek_cache_limit = 100
```

nCipher

For a nCipher nShield Connect XC, the plugin configuration looks like:

```
# ===== Secret Store Plugin =====
[secretstore]
..
enabled_secretstore_plugins = store_crypto

[p11_crypto_plugin]
# Path to vendor PKCS11 library
library_path = '/opt/nfast/toolkits/pkcs11/libcknfast.so'

# Token serial number used to identify the token to be used.
↳ Required
# when the device has multiple tokens with the same label. (string
# value)
token_serial_number = 12345678

# Token label used to identify the token to be used. Required
↳ when
# token_serial_number is not specified. (string value)
#token_label = <None>

# Password to login to PKCS11 session
login = 'XXX'

# Label to identify master KEK in the HSM (must not be the same
↳ as HMAC label)
mkek_label = 'thales_mkek_0'

# Length in bytes of master KEK
mkek_length = 32

# Label to identify HMAC key in the HSM (must not be the same as
↳ MKEK label)
hmac_label = 'thales_hmac_0'

# (Optional) HSM Slot ID that contains the token device to be
↳ used.
# (integer value)
# slot_id = 1
```

(continues on next page)

(continued from previous page)

```

# Enable Read/Write session with the HSM?
# rw_session = True

# Length of Project KEKs to create
# pkek_length = 32

# How long to cache unwrapped Project KEKs
# pkek_cache_ttl = 900

# Max number of items in pkek cache
# pkek_cache_limit = 100

# Secret encryption mechanism (string value)
# Deprecated group/name - [p11_crypto_plugin]/algorithm
encryption_mechanism = CKM_AES_CBC

# HMAC Key Type (string value)
hmac_key_type=CKK_SHA256_HMAC

# HMAC Key Generation Mechanism (string value)
hmac_keygen_mechanism = CKM_NC_SHA256_HMAC_KEY_GEN

# Generate IVs for CKM_AES_GCM mechanism. (boolean value)
# Deprecated group/name - [p11_crypto_plugin]/generate_iv
aes_gcm_generate_iv=True

# Always set CKA_SENSITIVE=CK_TRUE including
# CKA_EXTRACTABLE=CK_TRUE keys.
# default true
always_set_cka_sensitive=false

```

The HMAC and MKEK keys can be generated as follows:

```

barbican-manage hsm gen_hmac \
--library-path /opt/nfast/toolkits/pkcs11/libcknfast.so \
--passphrase XXX --slot-id 1 --label thales_hmac_0 \
--key-type CKK_SHA256_HMAC \
--mechanism CKM_NC_SHA256_HMAC_KEY_GEN

```

```

barbican-manage hsm gen_mkek \
--library-path /opt/nfast/toolkits/pkcs11/libcknfast.so \
--passphrase XXX --slot-id 1 --label thales_mkek_0

```

ATOS Bull

For an ATOS Bull HSM, the plugin configuration looks like:

```

# ===== Secret Store Plugin =====
[secretstore]
..
enabled_secretstore_plugins = store_crypto

```

(continues on next page)

(continued from previous page)

```
[p11_crypto_plugin]
# Path to vendor PKCS11 library
library_path = '/usr/lib64/libnethsm.so'

# Token serial number used to identify the token to be used.
↳Required
# when the device has multiple tokens with the same label. (string
# value)
token_serial_number = 12345678

# Token label used to identify the token to be used. Required
↳when
# token_serial_number is not specified. (string value)
#token_label = <None>

# Password to login to PKCS11 session
login = 'XXX'

# Label to identify master KEK in the HSM (must not be the same
↳as HMAC label)
mkek_label = 'atos_mkek_0'

# Length in bytes of master KEK
mkek_length = 32

# Label to identify HMAC key in the HSM (must not be the same as
↳MKEK label)
hmac_label = 'atos_hmac_0'

# (Optional) HSM Slot ID that contains the token device to be
↳used.
# (integer value)
# slot_id = 1

# Enable Read/Write session with the HSM?
# rw_session = True

# Length of Project KEKs to create
# pkek_length = 32

# How long to cache unwrapped Project KEKs
# pkek_cache_ttl = 900

# Max number of items in pkek cache
# pkek_cache_limit = 100

# Secret encryption mechanism (string value)
# Deprecated group/name - [p11_crypto_plugin]/algorithm
encryption_mechanism = CKM_AES_CBC

# HMAC Key Type (string value)
hmac_key_type = CKK_GENERIC_SECRET

# HMAC Key Generation Mechanism (string value)
hmac_keygen_mechanism = CKM_GENERIC_SECRET_KEY_GEN
```

(continues on next page)

(continued from previous page)

```
# Always set CKA_SENSITIVE=CK_TRUE including
# CKA_EXTRACTABLE=CK_TRUE keys.
# default true
always_set_cka_sensitive=false
```

The HMAC and MKEK keys can be generated as follows:

```
barbican-manage hsm gen_hmac --library-path /usr/lib64/libnethsm.
↳so \
--passphrase XXX --slot-id 1 --label atos_hmac_0 \
--key-type CKK_GENERIC_SECRET \
--mechanism CKM_GENERIC_SECRET_KEY_GEN
```

```
barbican-manage hsm gen_mkek --library-path /usr/lib64/libnethsm.
↳so \
--passphrase XXX --slot-id 1 --label atos_mkek_0
```

Utimaco

The PKCS#11 plugin configuration looks like:

```
# ===== Secret Store Plugin =====
[secretstore]
..
enabled_secretstore_plugins = store_crypto

[p11_crypto_plugin]
# Path to vendor PKCS11 library (string value)
library_path = '/opt/utimaco/lib/libcs_pkcs11_R2.so'

# Token serial number used to identify the token to be used.
↳Required
# when the device has multiple tokens with the same label. (string
# value)
token_serial_number = 12345678

# Token label used to identify the token to be used. Required
↳when
# token_serial_number is not specified. (string value)
#token_label = <None>

# Password to login to PKCS11 session (string value)
login = '$up3r$e<retP4ssw0rd'

# Master KEK label (as stored in the HSM) (string value)
mkek_label = 'my_mkek'

# Master KEK length in bytes. (integer value)
#mkek_length = <None>

# Master HMAC Key label (as stored in the HSM) (string value)
hmac_label = 'my_hmac_key'
```

(continues on next page)

(continued from previous page)

```

# (Optional) HSM Slot ID that contains the token device to be
↪used.
# (integer value)
# slot_id = 1

# Flag for Read/Write Sessions (boolean value)
#rw_session = true

# Project KEK length in bytes. (integer value)
#pkek_length = 32

# Project KEK Cache Time To Live, in seconds (integer value)
#pkek_cache_ttl = 900

# Project KEK Cache Item Limit (integer value)
#pkek_cache_limit = 100

# Secret encryption mechanism (string value)
# Deprecated group/name - [p11_crypto_plugin]/algorithm
encryption_mechanism = CKM_AES_CBC

# HMAC Key Type (string value)
#hmac_key_type = CKK_AES

# HMAC Key Generation Algorithm (string value)
#hmac_keygen_mechanism = CKM_AES_KEY_GEN

# File to pull entropy for seeding RNG (string value)
#seed_file =

# Amount of data to read from file for seed (integer value)
#seed_length = 32

# User friendly plugin name (string value)
#plugin_name = PKCS11 HSM

# Generate IVs for CKM_AES_GCM mechanism. (boolean value)
# Deprecated group/name - [p11_crypto_plugin]/generate_iv
#aes_gcm_generate_iv = true

# HMAC key wrap mechanism
hmac_keywrap_mechanism = CKM_AES_MAC

```

The HMAC and MKEK keys can be generated as follows:

```

barbican-manage hsm gen_mkek --library-path \
/opt/utimaco/lib/libcs_pkcs11_R2.so --passphrase XXX \
--slot-id 0 --label 'my_mkek'

```

```

barbican-manage hsm gen_hmac --library-path \
/opt/utimaco/lib/libcs_pkcs11_R2.so --passphrase XXX \
--slot-id 0 --label 'my_hmac_key'

```

KMIP Plugin

This secret store plugin is used to communicate with a KMIP device. The secret is securely stored in the KMIP device directly, rather than in the Barbican database. The Barbican database maintains a reference to the secrets location for later retrieval.

The plugin can be configured to authenticate to the KMIP device using either a username and password, or using a client certificate.

The configuration for this plugin in `/etc/barbican/barbican.conf` is as follows:

```
[secretstore]
..
enabled_secretstore_plugins = kmip_crypto

[kmip_plugin]
username = 'admin'
password = 'password'
host = localhost
port = 5696
keyfile = '/path/to/certs/cert.key'
certfile = '/path/to/certs/cert.crt'
ca_certs = '/path/to/certs/LocalCA.crt'
```

Dogtag Plugin

Dogtag is the upstream project corresponding to the Red Hat Certificate System, a robust, full-featured PKI solution that contains a Certificate Manager (CA) and a Key Recovery Authority (KRA) which is used to securely store secrets.

The KRA stores secrets as encrypted blobs in its internal database, with the master encryption keys being stored either in a software-based NSS security database, or in a Hardware Security Module (HSM).

Note that the software-based NSS database configuration provides a secure option for those deployments that do not require or cannot afford an HSM. This is the only current plugin to provide this option.

The KRA communicates with HSMs using PKCS#11. For a list of certified HSMs, see the latest [release notes](#). Dogtag and the KRA meet all the relevant Common Criteria and FIPS specifications.

The KRA is a component of FreeIPA. Therefore, it is possible to configure the plugin with a FreeIPA server. More detailed instructions on how to set up Barbican with FreeIPA are provided [here](#).

The plugin communicates with the KRA using a client certificate for a trusted KRA agent. That certificate is stored in an NSS database as well as a PEM file as seen in the configuration below.

The configuration for this plugin in `/etc/barbican/barbican.conf` is as follows:

```
[secretstore]
..
enabled_secretstore_plugins = dogtag_crypto

[dogtag_plugin]
pem_path = '/etc/barbican/kra_admin_cert.pem'
dogtag_host = localhost
dogtag_port = 8443
```

(continues on next page)

(continued from previous page)

```
nss_db_path = '/etc/barbican/alias'
nss_password = 'password123'
```

2.3.3 Verify operation

Verify operation of the Key Manager (barbican) service.

Note: Perform these commands on the controller node.

1. Install python-barbicanclient package:

- For openSUSE and SUSE Linux Enterprise:

```
$ zypper install python-barbicanclient
```

- For Red Hat Enterprise Linux and CentOS:

```
$ yum install python-barbicanclient
```

- For Ubuntu:

```
$ apt-get install python-barbicanclient
```

2. Source the admin credentials to be able to perform Barbican API calls:

```
$ . admin-openrc
```

3. Use the OpenStack CLI to store a secret:

```
$ openstack secret store --name mysecret --payload j4=]d21
+-----+
| Field          | Value                                                                 |
+-----+-----+
| Secret href    | http://10.0.2.15:9311/v1/secrets/655d7d30-c11a-49d9-          |
|                | a0f1-34cdf53a36fa |
| Name           | mysecret                                                            |
| Created        | None                                                                |
| Status         | None                                                                |
| Content types  | None                                                                |
| Algorithm       | aes                                                                |
| Bit length     | 256                                                                |
| Secret type    | opaque                                                            |
+-----+-----+
```

(continues on next page)

(continued from previous page)

```

| Mode          | cbc
↩
| Expiration    | None
↩
+-----+
↩

```

4. Confirm that the secret was stored by retrieving it:

```

$ openstack secret get http://10.0.2.15:9311/v1/secrets/655d7d30-c11a-
↩49d9-a0f1-34cdf53a36fa
+-----+
↩
| Field        | Value
↩
+-----+
↩
| Secret href  | http://10.0.2.15:9311/v1/secrets/655d7d30-c11a-49d9-
↩a0f1-34cdf53a36fa |
| Name         | mysecret
↩
| Created      | 2016-08-16 16:04:10+00:00
↩
| Status       | ACTIVE
↩
| Content types | {u'default': u'application/octet-stream'}
↩
| Algorithm     | aes
↩
| Bit length   | 256
↩
| Secret type  | opaque
↩
| Mode         | cbc
↩
| Expiration   | None
↩
+-----+
↩

```

Note: Some items are populated after the secret has been created and will only display when retrieving it.

5. Confirm that the secret payload was stored by retrieving it:

```

$ openstack secret get http://10.0.2.15:9311/v1/secrets/655d7d30-c11a-
↩49d9-a0f1-34cdf53a36fa --payload
+-----+
| Field | Value |
+-----+
| Payload | j4=]d21 |
+-----+

```

2.3.4 Next steps

Your OpenStack environment now includes the barbican service.

To add additional services, see <https://docs.openstack.org/install-guide> .

The Key Manager service (barbican) provides secure storage, provisioning and management of secret data. This includes keying material such as symmetric keys, asymmetric keys, certificates and raw binary data.

This chapter assumes a working setup of OpenStack following the [OpenStack Installation Tutorial](#).

2.4 Setting up Barbican

2.4.1 Using Keystone Middleware with Barbican

Prerequisites

To enable Keystone integration with Barbican youll need a relatively current version of Keystone. It is sufficient if you are installing an OpenStack cloud where all services including Keystone and Barbican are from the same release. If you dont have an instance of Keystone available, you can use one of the following ways to setup your own.

1. [Simple Dockerized Keystone](#)
2. [Installing Keystone](#)
3. An OpenStack cloud with Keystone (Devstack in the simplest case)

Hooking up Barbican to Keystone

Assuming that youve already setup your Keystone instance, connecting Barbican to Keystone is quite simple. When completed, Barbican should require a valid X-Auth-Token to be provided with all API calls except the get version call.

1. Turn off any active instances of Barbican
2. Edit `/etc/barbican/barbican-api-paste.ini`
 1. Change the pipeline `/v1` value from `unauthenticated_barbican_api` to the authenticated `barbican-api-keystone`. This step will not be necessary on barbican from OpenStack Newton or higher, since barbican will default to using Keystone authentication as of OpenStack Newton.

```
[composite:main]
use = egg:Paste#urlmap
/: barbican_version
/v1: barbican-api-keystone
```

2. Replace `authtoken` filter values to match your Keystone setup

```
[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_
↪factory
```

(continues on next page)

(continued from previous page)

```
auth_plugin = password
username = {YOUR_KEYSTONE_USERNAME}
password = {YOUR_KEYSTONE_PASSWORD}
user_domain_id = {YOUR_KEYSTONE_USER_DOMAIN}
project_name = {YOUR_KEYSTONE_PROJECT}
project_domain_id = {YOUR_KEYSTONE_PROJECT_DOMAIN}
www_authenticate_uri = http://{YOUR_KEYSTONE_ENDPOINT}:5000/v3
auth_url = http://{YOUR_KEYSTONE_ENDPOINT}:5000/v3
```

Alternatively, you can shorten this to

```
[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_
↪factory
```

and store Barbicans Keystone credentials in the [keystone_authtoken] section of /etc/barbican/barbican.conf

```
[keystone_authtoken]
auth_plugin = password
username = {YOUR_KEYSTONE_USERNAME}
password = {YOUR_KEYSTONE_PASSWORD}
user_domain_id = {YOUR_KEYSTONE_USER_DOMAIN}
project_name = {YOUR_KEYSTONE_PROJECT}
project_domain_id = {YOUR_KEYSTONE_PROJECT_DOMAIN}
www_authenticate_uri = http://{YOUR_KEYSTONE_ENDPOINT}:5000/v3
auth_url = http://{YOUR_KEYSTONE_ENDPOINT}:5000/v3
```

3. Start Barbican {barbican_home}/bin/barbican.sh start

2.4.2 Troubleshooting your Barbican Setup

If you cannot find the answers you're looking for within this document, you can ask questions on the Freenode IRC channel #openstack-barbican

Getting a Barbican HTTP 401 error after a successful authentication to Keystone

What you might see

You get a HTTP 401 Unauthorized response even with a valid token

```
curl -X POST -H "X-Auth-Token: $TOKEN" -H "Content-type: application/json" \
↪\
-d '{"payload": "my-secret-here", "payload_content_type": "text/plain"}' \
http://localhost:9311/v1/secrets
```

Caused by

Expired signing cert on the Barbican server.

How to avoid

Check for an expired Keystone signing certificate on your Barbican server. Look at the expiration date in `/tmp/barbican/cache/signing_cert.pem`. If it is expired then follow these steps.

1. On your Keystone server, verify that `signing_cert.pem` has the same expiration date as the one on your Barbican machine. You can normally find `signing_cert.pem` on your Keystone server in `/etc/keystone/ssl/certs`.
2. If the cert matches then follow these steps to create a new one
 1. Delete it from both your Barbican and Keystone servers.
 2. Edit `/etc/keystone/ssl/certs/index.txt.attr` and set `unique_subject` to `no`.
 3. Run `keystone-manage pki_setup` to create a new `signing_cert.pem`
 4. The updated cert will be downloaded to your Barbican server the next time you hit the Barbican API.
3. If the cert **doesn't match** then delete the `signing_cert.pem` from your Barbican server. Do not delete from Keystone. The cert from Keystone will be downloaded to your machine the next time you hit the Barbican API.

Returned refs use localhost instead of the correct hostname

What you might see

```
curl -X POST -H "X-Auth-Token: $TOKEN" -H "Content-type: application/json" \
↳ \
-d '{"payload": "my-secret-here", "payload_content_type": "text/plain"}' \
http://myhostname.com/v1/secrets

# Response:
{
  "secret_ref": "http://localhost:9311/v1/secrets/UUID_HERE"
}
```

Caused by

The default configuration on the response host name is not modified to the endpoints host name (typically the load balancers DNS name and port).

How to avoid

Change your `barbican.conf` files `host_href` setting from `localhost:9311` to the correct host name (myhostname.com in the example above).

Barbicans tox tests fail to run on my Mac

What you might see

```
clang: error: unknown argument: '-mno-fused-madd'
```

How to avoid

There is a [great blog article](#) that provides more details on the error and how to work around it. This link provides more details on the error and how to work around it.

Barbicans tox tests fail to find ffi.h on my Mac

What you might see

```
c/_cffi_backend.c:13:10: fatal error: 'ffi.h' file not found
...
ERROR: could not install deps [...]; v = InvocationError('...', 1)
```

How to avoid

Be sure that xcode and cmd line tools are up to date. Easiest way is to run `xcode-select --install` from an OS X command line. Be sure to say yes when asked if you want to install the command line tools. Now `ls /usr/include/ffi/ffi.h` should show that missing file exists, and the tox tests should run.

Barbicans tox tests fail with ImportError: No module named _bsddb

What you might see

```
ImportError: No module named _bsddb
```

How to avoid

Running tests via tox (which uses testr) will create a .testrepository directory containing, among other things, data files. Those datafiles may be created with bsddb, if it is available in the environment. This can cause problems if you run in an environment that does not have bsddb. To resolve this, delete your .testrepository directory and run tox again.

uWSGI logs OOPS ! failed loading app

What you might see

```
...
spawned uWSGI master process (pid: 59190)
spawned uWSGI worker 1 (pid: 59191, cores: 1)
spawned uWSGI worker 1 (pid: 59192, cores: 1)
Loading paste environment: config:/etc/barbican/barbican-api-paste.ini
WSGI app 0 (mountpoint='') ready in 0 seconds on interpreter \
    0x7fd098c08520 pid: 59191 (default app)
OOPS ! failed loading app in worker 1 (pid 59192) :( trying again...
Respawned uWSGI worker 1 (new pid: 59193)
Loading paste environment: config:/etc/barbican/barbican-api-paste.ini
OOPS ! failed loading app in worker 1 (pid 59193) :( trying again...
worker respawning too fast !!! i have to sleep a bit (2 seconds)...
...
```

Note: You will not see any useful logs or stack traces with this error!

Caused by

The vassal (worker) processes are not able to access the datastore.

How to avoid

Check the `sql_connection` in your `barbican.conf` file, to make sure that it references a valid reachable database.

Cannot register CLI option error when importing logging

What you might see

```
...
File ".../oslo_config/cfg.py", line 1275, in register_cli_opt
    raise ArgsAlreadyParsedError("cannot register CLI option")
ArgsAlreadyParsedError: arguments already parsed: cannot register CLI_
↪option
```

Caused by

An attempt to call the `oslo.configs.register_cli_opt()` function after the configuration arguments were parsed (see the comments and method in the `oslo.config.projects.cfg.py` file for details).

How to avoid

Instead of calling `import barbican.openstack.common.log as logging` to get a logger, call `from barbican.common import config` with this to get a logger to use in your source file: `LOG = config.getLogger(__name__)`.

Responder raised TypeError: 'NoneType' object has no attribute '__getitem__'

What you might see

```
...
2013-04-14 14:17:56 [FALCON] [ERROR] POST \
/da71dfbc-a959-4ad3-bdab-5ee190ce7515/csrs? => Responder raised \
TypeError: 'NoneType' object has no attribute '__getitem__'
```

Caused by

Forgetting to set your non-nullable FKs in entities you create via `XxxxResource` classes.

How to avoid

Dont forget to set any FKs defined on an entity prior to using the repository to create it.

uWSGI config issue: ImportError: No module named site

What you might see

```
...
uwsgi socket 0 bound to TCP address :9311 fd 3
Python version: 2.7.3 (...) [...]
Set PythonHome to ./venv
ImportError: No module named site
```


Caused by

- Cant locate the Python virtualenv for the Barbican project.
- Either the broker setting above is incorrect, or else you havent started a queue process yet (such as RabbitMQ)

How to avoid

Make sure the uWSGI config file at `etc/barbican/barbican-api-paste.ini` is configured correctly (see installation steps above), esp. if the virtualenv folder is named differently than the `.ini` file has.

REST Request Fails with JSON error

What you might see

```
{
  "title": "Malformed JSON"
}
```

Caused by

Barbican REST server cannot parse the incoming JSON message from your REST client.

How to avoid

Make sure you are submitting properly formed JSON. For example, are there commas after all but the last name/value pair in a list? Are there quotes around all name/values that are text-based? Are the types of values matching what is expected (i.e. integer and boolean types instead of quoted text)?

If you are using the Advanced REST Client with Chrome, and you tried to upload a file to the secrets PUT call, not only will this fail due to the multi-part format it uses, but it will also try to submit this file for every REST request you make thereafter, causing this error. Close the tab/window with the client, and restart it again.

Crypto Mime Type Not Supported when I try to run tests or hit the API

What you might see

A stack trace that has this in it (for example):

```
CryptoMimeTypeNotSupportedException: Crypto Mime Type of 'text/plain' not
↳supported
```

Caused by

The Barbican plugins are not installed into a place where the Python plugin manager can find them.

How to avoid

Make sure you run the `pip install -e ..`

Python cant find module errors with the uWSGI scripts

What you might see

```
*** has_emperor mode detected (fd: 6) ***
...
!!! UNABLE to load uWSGI plugin: dlopen(/python_plugin.so, 10): image not_
→found !!!
...
File "./site-packages/paste/deploy/loadwsgi.py", line 22, in import_
→string
    return pkg_resources.EntryPoint.parse("x=" + s).load(False)
File "./site-packages/distribute-0.6.35-py2.7.egg/pkg_resources.py",
→line 2015, in load
    entry = __import__(self.module_name, globals(),globals(), ['__name__
→'])
ImportError: No module named barbican.api.app
...
*** Starting uWSGI 1.9.13 (64bit) on [Fri Jul  5 09:59:29 2013] ***
```

Caused by

The Barbican source modules are not found in the Python path of applications such as uwsgi.

How to avoid

Make sure you are running from your virtual env, and that pip was executed **after** you activated your virtual environment. This especially includes the `pip install -e` command. Also, it is possible that your virtual env gets corrupted, so you might need to rebuild it.

unable to open database file None None errors running scripts

What you might see

```
...
File "./site-packages/sqlalchemy/engine/strategies.py", line 80, in_
→connect
    return dialect.connect(*cargs, **cparams)
File "./site-packages/sqlalchemy/engine/default.py", line 283, in connect
```

(continues on next page)

(continued from previous page)

```
    return self.dbapi.connect(*cargs, **cparams)
OperationalError: (OperationalError) unable to open database file None None
[emperor] removed uwsgi instance barbican-api.ini
...
```

Caused by

Destination folder for the sqlite database is not found, or is not writable.

How to avoid

Make sure the `/var/lib/barbican/` folder exists and is writable by the user that is running the Barbican API process.

ValueError: No JSON object could be decoded with Keystoneclient middleware

What you might see

```
...
2013-08-15 16:55:15.759 2445 DEBUG keystoneclient.middleware.auth_token \
[-] Token validation failure. _validate_user_token \
./site-packages/keystoneclient/middleware/auth_token.py:711
...
2013-08-15 16:55:15.759 2445 TRACE keystoneclient.middleware.auth_token \
raise ValueError("No JSON object could be decoded")
2013-08-15 16:55:15.759 24458 TRACE keystoneclient.middleware.auth_token \
ValueError: No JSON object could be decoded
...
2013-08-15 16:55:15.766 2445 WARNING keystoneclient.middleware.auth_token \
[-] Authorization failed for token ...
2013-08-15 16:55:15.766 2445 INFO keystoneclient.middleware.auth_token \
[-] Invalid user token - rejecting request...
```

Caused by

The `keystoneclient` middleware component is looking for a `cms` command in `openssl` that wasnt available before version `1.0.1`.

How to avoid

Update openssl.

accept-encoding of gzip,deflate,sdch not supported

What you might see

```
Secret retrieval issue seen - accept-encoding of 'gzip,deflate,sdch' not_
↳supported
```

Caused by

This might be an issue with the browser you are using, as performing the request via curl doesnt seem to be affected.

How to avoid

Other than using an command such as curl to make the REST request you may not have many other options.

2.4.3 No Auth barbican

As of OpenStack Newton, barbican will default to using Keystone like every other OpenStack service for identity and access control. Nonetheless, sometimes it may be useful to run barbican without any authentication service for development purposes.

To this end, `barbican-api-paste.ini` contains a filter pipeline without any authentication (no auth mode):

```
# Use this pipeline for barbican API - DEFAULT no authentication
[pipeline:barbican_api]
pipeline = unauthenticated-context apiapp
```

To enable this pipeline proceed as follows:

1. Turn off any active instances of barbican
2. Edit `/etc/barbican/barbican-api-paste.ini`

Change the pipeline `/v1` value from `authenticated barbican-api-keystone` to the `unauthenticated barbican_api`

```
[composite:main]
use = egg:Paste#urlmap
/: barbican_version
/v1: barbican_api
```

With every OpenStack service integrated with keystone, its API requires access token to retireve certain information and validate users information and privileges. If you are running barbican in no

auth mode, you have to specify `project_id` instead of an access token which was retrieved from the token instead. In case of API, replace `'X-Auth-Token: $TOKEN'` with `'X-Project-Id: {project_id}'` for every API request in [Barbican API Documentation](#).

You can also find detailed explanation to run barbican client with an unauthenticated context [here](#) and run barbican CLI in no auth mode [here](#).

2.4.4 Using Audit Middleware with Barbican

Background

[Audit middleware](#) is a python middleware logic which is added in service request processing pipeline via paste deploy filters. Audit middleware constructs audit event data in [CADF format](#).

Audit middleware supports delivery of CADF audit events via Oslo messaging notifier capability. Based on `notification_driver` configuration, audit events can be routed to messaging infrastructure (`notification_driver = messagingv2`) or can be routed to a log file (`notification_driver = log`).

Audit middleware creates two events per REST API interaction. First event has information extracted from request data and the second one has request outcome (response).

Enabling Audit for API Requests

Audit middleware is available as part of [keystonemiddleware](#) (≥ 1.6) library. Assuming a barbican deployment is already using keystone for token validation, auditing support requires only configuration changes. It has Oslo messaging library dependency as it uses this for audit event delivery. `pyCADF` library is used for creating events in CADF format.

- Enable Middleware : [Enabling Middleware Link](#) . Change is primarily in service paste deploy configuration.
- Configure Middleware : [Configuring Middleware Link](#) . Can use provided audit mapping file. If there are no custom mapping for actions or path, then related mapping values are derived from taxonomy defined in `pyCADF` library.

Note: Audit middleware filter should be included after Keystone middlewares `keystone_authtoken` middleware in request pipeline. This is needed so that audit middleware can utilize environment variables set by `keystone_authtoken` middleware.

Steps

1. Turn off any active instances of Barbican.
2. Copy `api_audit_map.conf` to `/etc/barbican` directory.
3. Edit `/etc/barbican/barbican-api-paste.ini`

Replace the `/v1` app pipeline from `barbican_api` to `barbican-api-keystone-audit` pipeline:

```
[pipeline:barbican-api-keystone-audit]
pipeline = authtoken context audit apiapp
```

4. Edit `barbican.conf` to update `notification_driver` value.
5. Start Barbican `{barbican_home}/bin/barbican.sh start`

Sample Audit Event

Following is the sample of audit event for symmetric key create request

```
{
  "priority": "INFO",
  "event_type": "audit.http.request",
  "timestamp": "2015-12-11 00:44:26.412076",
  "publisher_id": "uwsgi",
  "payload": {
    "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
    "eventTime": "2015-12-11T00:44:26.410768+0000",
    "target": {
      "typeURI": "service/security/keymanager/secrets",
      "addresses": [
        {
          "url": "http://{barbican_admin_host}:9311",
          "name": "admin"
        },
        {
          "url": "http://{barbican_internal_host}:9311",
          "name": "private"
        },
        {
          "url": "https://{barbican_public_host}:9311",
          "name": "public"
        }
      ],
      "name": "barbican_service_user",
      "id": "barbican"
    },
    "observer": {
      "id": "target"
    },
    "tags": [
      "correlation_id?value=openstack:7e0fe4a6-e258-477e-a1c9-
↪0fd0921a8435"
    ],
    "eventType": "activity",
    "initiator": {
      "typeURI": "service/security/account/user",
      "name": "cinder_user",
      "credential": {
        "token": "***",
        "identity_status": "Confirmed"
      },
      "host": {
        "agent": "curl/7.38.0",
        "address": "192.168.245.2"
      },
      "project_id": "8eabee0a4c4e40f882df8efbce695526",
      "id": "513e8682f23446ceb598b6b0f5c4482b"
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

    },
    "action": "create",
    "outcome": "pending",
    "id": "openstack:3a6a961c-9ada-4b81-9095-90968d896c41",
    "requestPath": "/v1/secrets"
  },
  "message_id": "afc3fd93-51e9-4c80-b330-983e66962265"
}

```

Ceilometer audit wiki can be referred to identify meaning of different fields in audit event to **7 Ws of Audit and Compliance**.

2.4.5 Using Secret Store Plugins in Barbican

Summary

By default, Barbican is configured to use one active secret store plugin in a deployment. This means that all of the new secrets are going to be stored via same plugin mechanism (i.e. same storage backend).

In **Newton** OpenStack release, support for configuring multiple secret store plugin backends is added ([Spec Link](#)). As part of this change, client can choose to select preferred plugin backend for storing their secret at a project level.

Enabling Multiple Barbican Backends

Multiple backends support may be needed in specific deployment/ use-case scenarios and can be enabled via configuration.

For this, a Barbican deployment may have more than one secret storage backend added in service configuration. Project administrators will have choice of pre-selecting one backend as the preferred choice for secrets created under that project. Any **new** secret created under that project will use the preferred backend to store its key material. When there is no project level storage backend selected, then new secret will use the global secret storage backend.

Multiple plugin configuration can be defined as follows.

```

[secretstore]
# Set to True when multiple plugin backends support is needed
enable_multiple_secret_stores = True
stores_lookup_suffix = software, kmip, pkcs11, dogtag

[secretstore:software]
secret_store_plugin = store_crypto
crypto_plugin = simple_crypto

[secretstore:kmip]
secret_store_plugin = kmip_plugin
global_default = True

[secretstore:dogtag]
secret_store_plugin = dogtag_plugin

```

(continues on next page)

(continued from previous page)

```
[secretstore:pkcs11]
secret_store_plugin = store_crypto
crypto_plugin = p11_crypto
```

When *enable_multiple_secret_stores* is enabled (True), then list property *stores_lookup_suffix* is used for looking up supported plugin names in configuration section. This section name is constructed using pattern `secretstore:{one_of_suffix}`. One of the plugin **must** be explicitly identified as global default i.e. *global_default = True*. Ordering of suffix and label used does not matter as long as there is a matching section defined in service configuration.

Note: For existing Barbican deployment case, its recommended to keep existing secretstore and crypto plugin (if applicable) name combination to be used as global default secret store. This is needed to be consistent with existing behavior.

Warning: When multiple plugins support is enabled, then *enabled_secretstore_plugins* and *enabled_crypto_plugins* values are **not** used to instantiate relevant plugins. Only above mentioned mechanism is used to identify and instantiate store and crypto plugins.

Multiple backend can be useful in following type of usage scenarios.

- In a deployment, a deployer may be okay in storing their dev/test resources using a low-security secret store, such as one backend using software-only crypto, but may want to use an HSM-backed secret store for production resources.
- In a deployment, for certain use cases where a client requires high concurrent access of stored keys, HSM might not be a good storage backend. Also scaling them horizontally to provide higher scalability is a costly approach with respect to database.
- HSM devices generally have limited storage capacity so a deployment will have to watch its stored keys size proactively to remain under the limit constraint. This is more applicable in KMIP backend than with PKCS11 backend because of plugins different storage approach. This aspect can also result from above use case scenario where deployment is storing non-sensitive (from dev/test environment) encryption keys in HSM.
- Barbican running as IaaS service or platform component where some class of client services have strict compliance requirements (e.g. FIPS) so will use HSM backed plugins whereas others may be okay storing keys in software-only crypto plugin.

2.4.6 barbican.conf

DEFAULT

admin_role

Type string

Default admin

Role used to identify an authenticated user as administrator.

allow_anonymous_access

Type boolean

Default False

Allow unauthenticated users to access the API with read-only privileges. This only applies when using ContextMiddleware.

max_allowed_request_size_in_bytes

Type integer

Default 25000

Maximum allowed http request size against the barbican-api.

max_allowed_secret_in_bytes

Type integer

Default 20000

Maximum allowed secret size in bytes.

host_href

Type string

Default http://localhost:9311

Host name, for use in HATEOAS-style references Note: Typically this would be the load balanced endpoint that clients would use to communicate back with this service. If a deployment wants to derive host from wsgi request instead then make this blank. Blank is needed to override default config value which is <http://localhost:9311>

sql_connection

Type string

Default sqlite:///barbican.sqlite

SQLAlchemy connection string for the reference implementation registry server. Any valid SQLAlchemy connection string is fine. See: http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.html#sqlalchemy.create_engine. Note: For absolute addresses, use `///` slashes after sqlite:.

sql_idle_timeout

Type integer

Default 3600

Period in seconds after which SQLAlchemy should reestablish its connection to the database. MySQL uses a default *wait_timeout* of 8 hours, after which it will drop idle connections. This can result in MySQL Gone Away exceptions. If you notice this, you can lower this value to ensure that SQLAlchemy reconnects before MySQL can drop the connection.

sql_max_retries

Type integer

Default 60

Maximum number of database connection retries during startup. Set to -1 to specify an infinite retry count.

sql_retry_interval

Type integer

Default 1

Interval between retries of opening a SQL connection.

db_auto_create

Type boolean

Default False

Create the Barbican database on service startup.

max_limit_paging

Type integer

Default 100

Maximum page size for the limit paging URL parameter.

default_limit_paging

Type integer

Default 10

Default page size for the limit paging URL parameter.

sql_pool_class

Type string

Default QueuePool

Accepts a class imported from the sqlalchemy.pool module, and handles the details of building the pool for you. If commented out, SQLAlchemy will select based on the database dialect. Other options are QueuePool (for SQLAlchemy-managed connections) and NullPool (to disabled SQLAlchemy management of connections). See <http://docs.sqlalchemy.org/en/latest/core/pooling.html> for more details

sql_pool_logging

Type boolean

Default False

Show SQLAlchemy pool-related debugging output in logs (sets DEBUG log level output) if specified.

sql_pool_size

Type integer

Default 5

Size of pool used by SQLAlchemy. This is the largest number of connections that will be kept persistently in the pool. Can be set to 0 to indicate no size limit. To disable pooling, use a NullPool with sql_pool_class instead. Comment out to allow SQLAlchemy to select the default.

sql_pool_max_overflow

Type integer

Default 10

The maximum overflow size of the pool used by SQLAlchemy. When the number of checked-out connections reaches the size set in `sql_pool_size`, additional connections will be returned up to this limit. It follows then that the total number of simultaneous connections the pool will allow is `sql_pool_size + sql_pool_max_overflow`. Can be set to -1 to indicate no overflow limit, so no limit will be placed on the total number of concurrent connections. Comment out to allow SQLAlchemy to select the default.

backdoor_port

Type string

Default <None>

Enable eventlet backdoor. Acceptable values are 0, <port>, and <start>:<end>, where 0 results in listening on a random tcp port number; <port> results in listening on the specified port number (and not enabling backdoor if that port is in use); and <start>:<end> results in listening on the smallest unused port number within the specified range of port numbers. The chosen port is displayed in the services log file.

backdoor_socket

Type string

Default <None>

Enable eventlet backdoor, using the provided path as a unix socket that can receive connections. This option is mutually exclusive with `backdoor_port` in that only one should be provided. If both are provided then the existence of this option overrides the usage of that option. Inside the path `{pid}` will be replaced with the PID of the current process.

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 1: 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 2: 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 3: 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.

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.

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 [%(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 %(tenant)s %(domain)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 ['amqp=WARN', 'amqpplib=WARN', 'boto=WARN', 'qpuid=WARN', 'sqlalchemy=WARN', 'suds=INFO', 'oslo.messaging=INFO', 'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN', 'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN', 'urllib3.util.retry=WARN', 'keystonemiddleware=WARN', 'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=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

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. 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.

rpc_conn_pool_size

Type integer

Default 30

Minimum Value 1

Size of RPC connection pool.

Table 4: 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 5: 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** openstack

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

run_external_periodic_tasks**Type** boolean**Default** True

Some periodic tasks can be run in a separate process. Should we run them here?

api_paste_config**Type** string**Default** api-paste.ini

File name for the paste.deploy config for api service

wsgi_log_format**Type** string**Default**

```
%(client_ip)s "%(request_line)s" status:
%(status_code)s len: %(body_length)s time:
%(wall_seconds).7f
```

A python format string that is used as the template to generate log lines. The following values can beformatted into it: `client_ip`, `date_time`, `request_line`, `status_code`, `body_length`, `wall_seconds`.

tcp_keepidle

Type integer

Default 600

Sets the value of TCP_KEEPIDLE in seconds for each server socket. Not supported on OS X.

wsgi_default_pool_size

Type integer

Default 100

Size of the pool of greenthreads used by wsgi

max_header_line

Type integer

Default 16384

Maximum line size of message headers to be accepted. `max_header_line` may need to be increased when using large tokens (typically those generated when keystone is configured to use PKI tokens with big service catalogs).

wsgi_keep_alive

Type boolean

Default True

If False, closes the client socket connection explicitly.

client_socket_timeout

Type integer

Default 900

Timeout for client connections socket operations. If an incoming connection is idle for this number of seconds it will be closed. A value of 0 means wait forever.

wsgi_server_debug

Type boolean

Default False

True if the server should send exception tracebacks to the clients on 500 errors. If False, the server will respond with empty bodies.

certificate

namespace

Type string

Default `barbican.certificate.plugin`

Extension namespace to search for plugins.

enabled_certificate_plugins

Type multi-valued

Default `simple_certificate`

List of certificate plugins to load.

certificate_event

namespace

Type string

Default `barbican.certificate.event.plugin`

Extension namespace to search for eventing plugins.

enabled_certificate_event_plugins

Type multi-valued

Default `simple_certificate_event`

List of certificate plugins to load.

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-Project-Id', 'X-Identity-Status', 'X-User-Id',
'X-Storage-Token', 'X-Domain-Id', 'X-User-Domain-Id',
'X-Project-Domain-Id', 'X-Roles']

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-Project-Id', 'X-Identity-Status', 'X-User-Id',
'X-Storage-Token', 'X-Domain-Id', 'X-User-Domain-Id',
'X-Project-Domain-Id', 'X-Roles']

Indicate which header field names may be used during the actual request.

crypto

namespace

Type string

Default barbican.crypto.plugin

Extension namespace to search for plugins.

enabled_crypto_plugins

Type multi-valued

Default simple_crypto

List of crypto plugins to load.

dogtag_plugin**pem_path****Type** string**Default** /etc/barbican/kra_admin_cert.pem

Path to PEM file for authentication

dogtag_host**Type** string**Default** localhost

Hostname for the Dogtag instance

dogtag_port**Type** port number**Default** 8443**Minimum Value** 0**Maximum Value** 65535

Port for the Dogtag instance

nss_db_path**Type** string**Default** /etc/barbican/alias

Path to the NSS certificate database

nss_password**Type** string**Default** <None>

Password for the NSS certificate databases

simple_cmc_profile**Type** string**Default** caOtherCert

Profile for simple CMC requests

auto_approved_profiles**Type** string**Default** caServerCert

List of automatically approved enrollment profiles

ca_expiration_time**Type** integer**Default** 1

Time in days for CA entries to expire

plugin_working_dir

Type string

Default /etc/barbican/dogtag

Working directory for Dogtag plugin

plugin_name

Type string

Default Dogtag KRA

User friendly plugin name

retries

Type integer

Default 3

Retries when storing or generating secrets

keystone_authtoken

www_authenticate_uri

Type string

Default <None>

Complete public Identity API endpoint. This endpoint should not be an admin endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If you're using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint.

Table 6: Deprecated Variations

Group	Name
keystone_authtoken	auth_uri

auth_uri

Type string

Default <None>

Complete public Identity API endpoint. This endpoint should not be an admin endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If you're using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint. This option is deprecated in favor of `www_authenticate_uri` and will be removed in the S release.

Warning: This option is deprecated for removal since Queens. Its value may be silently ignored in the future.

Reason The `auth_uri` option is deprecated in favor of `www_authenticate_uri` and will be removed in the S release.

auth_version

Type string

Default <None>

API version of the Identity API endpoint.

interface

Type string

Default `internal`

Interface to use for the Identity API endpoint. Valid values are `public`, `internal` (default) or `admin`.

delay_auth_decision

Type boolean

Default `False`

Do not handle authorization requests within the middleware, but delegate the authorization decision to downstream WSGI components.

http_connect_timeout

Type integer

Default <None>

Request timeout value for communicating with Identity API server.

http_request_max_retries

Type integer

Default `3`

How many times are we trying to reconnect when communicating with Identity API Server.

cache

Type string

Default <None>

Request environment key where the Swift cache object is stored. When `auth_token` middleware is deployed with a Swift cache, use this option to have the middleware share a caching backend with swift. Otherwise, use the `memcached_servers` option instead.

certfile

Type string

Default <None>

Required if identity server requires client certificate

keyfile

Type string

Default <None>

Required if identity server requires client certificate

cafile

Type string

Default <None>

A PEM encoded Certificate Authority to use when verifying HTTPs connections. Defaults to system CAs.

insecure

Type boolean

Default False

Verify HTTPS connections.

region_name

Type string

Default <None>

The region in which the identity server can be found.

memcached_servers

Type list

Default <None>

Optionally specify a list of memcached server(s) to use for caching. If left undefined, tokens will instead be cached in-process.

Table 7: Deprecated Variations

Group	Name
keystone_authtoken	memcache_servers

token_cache_time

Type integer

Default 300

In order to prevent excessive effort spent validating tokens, the middleware caches previously-seen tokens for a configurable duration (in seconds). Set to -1 to disable caching completely.

memcache_security_strategy

Type string

Default None

Valid Values None, MAC, ENCRYPT

(Optional) If defined, indicate whether token data should be authenticated or authenticated and encrypted. If MAC, token data is authenticated (with HMAC) in the cache. If ENCRYPT, token data is encrypted and authenticated in the cache. If the value is not one of these options or empty, `auth_token` will raise an exception on initialization.

memcache_secret_key

Type string

Default <None>

(Optional, mandatory if `memcache_security_strategy` is defined) This string is used for key derivation.

memcache_pool_dead_retry

Type integer

Default 300

(Optional) Number of seconds memcached server is considered dead before it is tried again.

memcache_pool_maxsize

Type integer

Default 10

(Optional) Maximum total number of open connections to every memcached server.

memcache_pool_socket_timeout

Type integer

Default 3

(Optional) Socket timeout in seconds for communicating with a memcached server.

memcache_pool_unused_timeout

Type integer

Default 60

(Optional) Number of seconds a connection to memcached is held unused in the pool before it is closed.

memcache_pool_conn_get_timeout

Type integer

Default 10

(Optional) Number of seconds that an operation will wait to get a memcached client connection from the pool.

memcache_use_advanced_pool

Type boolean

Default True

(Optional) Use the advanced (eventlet safe) memcached client pool.

include_service_catalog

Type boolean

Default True

(Optional) Indicate whether to set the X-Service-Catalog header. If False, middleware will not ask for service catalog on token validation and will not set the X-Service-Catalog header.

enforce_token_bind

Type string

Default permissive

Used to control the use and type of token binding. Can be set to: disabled to not check token binding. permissive (default) to validate binding information if the bind type is of a form known to the server and ignore it if not. strict like permissive but if the bind type is unknown the token will be rejected. required any form of token binding is needed to be allowed. Finally the name of a binding method that must be present in tokens.

service_token_roles

Type list

Default ['service']

A choice of roles that must be present in a service token. Service tokens are allowed to request that an expired token can be used and so this check should tightly control that only actual services should be sending this token. Roles here are applied as an ANY check so any role in this list must be present. For backwards compatibility reasons this currently only affects the allow_expired check.

service_token_roles_required

Type boolean

Default False

For backwards compatibility reasons we must let valid service tokens pass that dont pass the service_token_roles check as valid. Setting this true will become the default in a future release and should be enabled if possible.

service_type

Type string

Default <None>

The name or type of the service as it appears in the service catalog. This is used to validate tokens that have restricted access rules.

auth_type

Type unknown type

Default <None>

Authentication type to load

Table 8: Deprecated Variations

Group	Name
keystone_authtoken	auth_plugin

auth_section**Type** unknown type**Default** <None>

Config Section from which to load plugin specific options

keystone_notifications**enable****Type** boolean**Default** False

True enables keystone notification listener functionality.

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.

topic**Type** string**Default** notifications

Keystone notification queue topic name. This name needs to match one of values mentioned in Keystone deployments notification_topics configuration e.g. notification_topics=notifications, barbican_notifications. Multiple servers may listen on a topic and messages will be dispatched to one of the servers in a round-robin fashion. That's why Barbican service should have its own dedicated notification queue so that it receives all of Keystone notifications. Alternatively if the chosen oslo.messaging backend supports listener pooling (for example rabbitmq), setting a non-default pool_name option should be preferred.

pool_name**Type** string**Default** <None>

Pool name for notifications listener. Setting this to a distinctive value will allow barbican notifications listener to receive its own copy of all messages from the topic without interfering with other services listening on the same topic. This feature is supported only by some oslo.messaging backends (in particular by rabbitmq) and for those it is preferable to use it instead of separate notification topic for barbican.

allow_requeue**Type** boolean**Default** False

True enables requeue feature in case of notification processing error. Enable this only when underlying transport supports this feature.

version

Type string

Default 1.0

Version of tasks invoked via notifications

thread_pool_size

Type integer

Default 10

Define the number of max threads to be used for notification server processing functionality.

kmip_plugin

username

Type string

Default <None>

Username for authenticating with KMIP server

password

Type string

Default <None>

Password for authenticating with KMIP server

host

Type string

Default localhost

Address of the KMIP server

port

Type port number

Default 5696

Minimum Value 0

Maximum Value 65535

Port for the KMIP server

ssl_version

Type string

Default PROTOCOL_TLSv1_2

SSL version, maps to the module ssls constants

ca_certs

Type string

Default <None>

File path to concatenated certification authority certificates

certfile

Type string

Default <None>

File path to local client certificate

keyfile

Type string

Default <None>

File path to local client certificate keyfile

pkcs1_only

Type boolean

Default False

Only support PKCS#1 encoding of asymmetric keys

plugin_name

Type string

Default KMIP HSM

User friendly plugin name

oslo_messaging_amqp

container_name

Type string

Default <None>

Name for the AMQP container. must be globally unique. Defaults to a generated UUID

Table 9: Deprecated Variations

Group	Name
amqp1	container_name

idle_timeout

Type integer

Default 0

Timeout for inactive connections (in seconds)

Table 10: Deprecated Variations

Group	Name
amqp1	idle_timeout

trace

Type boolean

Default `False`

Debug: dump AMQP frames to stdout

Table 11: 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 12: Deprecated Variations

Group	Name
amqp1	ssl_ca_file

ssl_cert_file

Type string

Default `''`

Self-identifying certificate PEM file for client authentication

Table 13: 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 14: Deprecated Variations

Group	Name
amqp1	ssl_key_file

ssl_key_password**Type** string**Default** <None>

Password for decrypting ssl_key_file (if encrypted)

Table 15: 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 16: Deprecated Variations

Group	Name
amqp1	sasl_mechanisms

sasl_config_dir**Type** string**Default** ''

Path to directory that contains the SASL configuration

Table 17: Deprecated Variations

Group	Name
amqp1	sasl_config_dir

sasl_config_name**Type** string**Default** ''

Name of configuration file (without .conf suffix)

Table 18: 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** 0Increase the `connection_retry_interval` by this many seconds after each unsuccessful failover attempt.**connection_retry_interval_max****Type** integer**Default** 30**Minimum Value** 1Maximum 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 qpid). 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 19: Deprecated Variations

Group	Name
amqp1	server_request_prefix

broadcast_prefix

Type `string`

Default `broadcast`

address prefix used when broadcasting to all servers

Table 20: Deprecated Variations

Group	Name
amqp1	broadcast_prefix

group_request_prefix

Type `string`

Default `unicast`

address prefix when sending to any server in group

Table 21: 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 22: 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 23: Deprecated Variations

Group	Name
DEFAULT	notification_transport_url

topics

Type list

Default ['notifications']

AMQP topic used for OpenStack notifications.

Table 24: 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 25: Deprecated Variations

Group	Name
DEFAULT	amqp_auto_delete

ssl

Type boolean

Default False

Connect over SSL.

Table 26: 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 27: 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 28: 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 29: 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 30: 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

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.

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 31: 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 32: 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 33: 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 34: 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 35: 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 36: Deprecated Variations

Group	Name
oslo_messaging_rabbit	rabbit_quorum_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 37: Deprecated Variations

Group	Name
oslo_messaging_rabbit	rabbit_quorum_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 your 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 let 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 consumers when queue is down

use_queue_manager

Type boolean

Default False

Should we use constant queue names or random ones

hostname

Type string

Default np0037209945

Hostname used by queue manager

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). The stream 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 replace all fanout queues with only one stream queue.

oslo_middleware

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.

oslo_policy

enforce_scope

Type boolean

Default False

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.

enforce_new_defaults

Type boolean

Default False

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 OR'd 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.json`

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 38: Deprecated Variations

Group	Name
DEFAULT	<code>policy_file</code>

policy_default_rule

Type string

Default `default`

Default rule. Enforced when a requested rule is not found.

Table 39: 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 40: 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_cert

Type boolean

Default `False`

server identity verification for REST based policy check

remote_ssl_ca_cert_file

Type string

Default `<None>`

Absolute path to ca cert file for REST based policy check

remote_ssl_client_cert_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

p11_crypto_plugin

library_path

Type string

Default <None>

Path to vendor PKCS11 library

token_serial_number

Type string

Default <None>

Token serial number used to identify the token to be used.

token_label

Type string

Default <None>

DEPRECATED: Use token_labels instead. Token label used to identify the token to be used.

Warning: This option is deprecated for removal. Its value may be silently ignored in the future.

token_labels

Type list

Default <None>

List of labels for one or more tokens to be used. Typically this is a single label, but some HSM devices may require more than one label for Load Balancing or High Availability configurations.

login

Type string

Default <None>

Password to login to PKCS11 session

mkek_label

Type string

Default <None>

Master KEK label (as stored in the HSM)

mkek_length

Type integer

Default <None>

Master KEK length in bytes.

hmac_label

Type string

Default <None>

Master HMAC Key label (as stored in the HSM)

slot_id

Type integer

Default 1

(Optional) HSM Slot ID that contains the token device to be used.

rw_session

Type boolean

Default True

Flag for Read/Write Sessions

pkek_length

Type integer

Default 32

Project KEK length in bytes.

pkek_cache_ttl

Type integer

Default 900

Project KEK Cache Time To Live, in seconds

pkek_cache_limit

Type integer

Default 100

Project KEK Cache Item Limit

encryption_mechanism

Type string

Default CKM_AES_CBC

Secret encryption mechanism

Table 41: Deprecated Variations

Group	Name
p11_crypto_plugin	algorithm

hmac_key_type

Type string

Default CKK_AES

HMAC Key Type

hmac_keygen_mechanism

Type string

Default CKM_AES_KEY_GEN

HMAC Key Generation Algorithm

hmac_keywrap_mechanism

Type string

Default CKM_SHA256_HMAC

HMAC key wrap mechanism

seed_file

Type string

Default ''

File to pull entropy for seeding RNG

seed_length

Type integer

Default 32

Amount of data to read from file for seed

plugin_name

Type string

Default PKCS11_HSM

User friendly plugin name

aes_gcm_generate_iv

Type boolean

Default True

Generate IVs for CKM_AES_GCM mechanism.

Table 42: Deprecated Variations

Group	Name
p11_crypto_plugin	generate_iv

always_set_cka_sensitive

Type boolean

Default True

Always set CKA_SENSITIVE=CK_TRUE including CKA_EXTRACTABLE=CK_TRUE keys.

os_locking_ok

Type boolean

Default False

Enable CKF_OS_LOCKING_OK flag when initializing the PKCS#11 client library.

queue

enable

Type boolean

Default False

True enables queuing, False invokes workers synchronously

namespace

Type string

Default barbican

Queue namespace

topic

Type string

Default barbican.workers

Queue topic name

version

Type string

Default 1.1

Version of tasks invoked via queue

server_name

Type string

Default barbican.queue

Server name for RPC task processing server

asynchronous_workers

Type integer

Default 1

Number of asynchronous worker processes

quotas

quota_secrets

Type integer

Default -1

Number of secrets allowed per project

quota_orders

Type integer

Default -1

Number of orders allowed per project

quota_containers

Type integer

Default -1

Number of containers allowed per project

quota_consumers

Type integer

Default -1

Number of consumers allowed per project

quota_cas

Type integer

Default -1

Number of CAs allowed per project

retry_scheduler

initial_delay_seconds

Type floating point

Default 10.0

Seconds (float) to wait before starting retry scheduler

periodic_interval_max_seconds

Type floating point

Default 10.0

Seconds (float) to wait between periodic schedule events

secretstore

namespace

Type string

Default barbican.secretstore.plugin

Extension namespace to search for plugins.

enabled_secretstore_plugins

Type multi-valued

Default store_crypto

List of secret store plugins to load.

enable_multiple_secret_stores

Type boolean

Default False

Flag to enable multiple secret store plugin backend support. Default is False

stores_lookup_suffix

Type list

Default <None>

List of suffix to use for looking up plugins which are supported with multiple backend support.

simple_crypto_plugin**kek**

Type string

Default dGhpcnR5X3R3b19ieXRlX2tleWJsYWhibGFoYmxhaGg=

Key encryption key to be used by Simple Crypto Plugin

plugin_name

Type string

Default Software Only Crypto

User friendly plugin name

snakeoil_ca_plugin**ca_cert_path**

Type string

Default <None>

Path to CA certificate file

ca_cert_key_path

Type string

Default <None>

Path to CA certificate key file

ca_cert_chain_path

Type string

Default <None>

Path to CA certificate chain file

ca_cert_pkcs7_path

Type string

Default <None>

Path to CA chain pkcs7 file

subca_cert_key_directory

Type string

Default /etc/barbican/snakeoil-cas

Directory in which to store certs/keys for subcas

ssl

ca_file

Type string

Default <None>

CA certificate file to use to verify connecting clients.

Table 43: Deprecated Variations

Group	Name
DEFAULT	ssl_ca_file

cert_file

Type string

Default <None>

Certificate file to use when starting the server securely.

Table 44: Deprecated Variations

Group	Name
DEFAULT	ssl_cert_file

key_file

Type string

Default <None>

Private key file to use when starting the server securely.

Table 45: Deprecated Variations

Group	Name
DEFAULT	ssl_key_file

version

Type string

Default <None>

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.

ciphers

Type string

Default <None>

Sets the list of available ciphers. value should be a string in the OpenSSL cipher list format.

2.4.7 Policy configuration

Configuration

The following is an overview of all available policies in Barbican. For a sample configuration file.

barbican

admin

Default `role:admin`

(no description provided)

observer

Default `role:observer`

(no description provided)

creator

Default `role:creator`

(no description provided)

audit

Default `role:audit`

(no description provided)

service_admin

Default `role:key-manager:service-admin`

(no description provided)

admin_or_creator

Default `rule:admin or rule:creator`

(no description provided)

all_but_audit

Default `rule:admin or rule:observer or rule:creator`

(no description provided)

all_users

Default rule:admin or rule:observer or rule:creator or rule:audit or rule:service_admin

(no description provided)

secret_project_match

Default project_id:%(target.secret.project_id)s

(no description provided)

secret_acl_read

Default 'read':%(target.secret.read)s

(no description provided)

secret_private_read

Default 'False':%(target.secret.read_project_access)s

(no description provided)

secret_creator_user

Default user_id:%(target.secret.creator_id)s

(no description provided)

container_project_match

Default project_id:%(target.container.project_id)s

(no description provided)

container_acl_read

Default 'read':%(target.container.read)s

(no description provided)

container_private_read

Default 'False':%(target.container.read_project_access)s

(no description provided)

container_creator_user

Default user_id:%(target.container.creator_id)s

(no description provided)

secret_non_private_read

Default rule:all_users and rule:secret_project_match and not rule:secret_private_read

(no description provided)

secret_decrypt_non_private_read

Default rule:all_but_audit and rule:secret_project_match and not rule:secret_private_read

(no description provided)

container_non_private_read

Default rule:all_users and rule:container_project_match and not rule:container_private_read

(no description provided)

secret_project_admin

Default rule:admin and rule:secret_project_match

(no description provided)

secret_project_creator

Default rule:creator and rule:secret_project_match and rule:secret_creator_user

(no description provided)

secret_project_creator_role

Default rule:creator and rule:secret_project_match

(no description provided)

container_project_admin

Default rule:admin and rule:container_project_match

(no description provided)

container_project_creator

Default rule:creator and rule:container_project_match and rule:container_creator_user

(no description provided)

container_project_creator_role

Default rule:creator and rule:container_project_match

(no description provided)

secret_acls:get

Default rule:all_but_audit and rule:secret_project_match

Operations

- **GET** /v1/secrets/{secret-id}/acl

Scope Types

Retrieve the ACL settings for a given secret.If no ACL is defined for that secret, then Default ACL is returned.

secret_acls:delete

Default rule:secret_project_admin or rule:secret_project_creator or (rule:secret_project_creator_role and rule:secret_non_private_read)

Operations

- **DELETE** /v1/secrets/{secret-id}/acl

Scope Types

Delete the ACL settings for a given secret.

secret_acls:put_patch

Default rule:secret_project_admin or rule:secret_project_creator or (rule:secret_project_creator_role and rule:secret_non_private_read)

Operations

- **PUT** /v1/secrets/{secret-id}/acl
- **PATCH** /v1/secrets/{secret-id}/acl

Scope Types

Create new, replaces, or updates existing ACL for a given secret.

container_acls:get

Default rule:all_but_audit and rule:container_project_match

Operations

- **GET** /v1/containers/{container-id}/acl

Scope Types

Retrieve the ACL settings for a given container.

container_acls:delete

Default rule:container_project_admin or rule:container_project_creator or (rule:container_project_creator_role and rule:container_non_private_read)

Operations

- **DELETE** /v1/containers/{container-id}/acl

Scope Types

Delete ACL for a given container. No content is returned in the case of successful deletion.

container_acls:put_patch

Default rule:container_project_admin or rule:container_project_creator or (rule:container_project_creator_role and rule:container_non_private_read)

Operations

- **PUT** /v1/containers/{container-id}/acl
- **PATCH** /v1/containers/{container-id}/acl

Scope Types

Create new or replaces existing ACL for a given container.

consumer: get

Default rule:admin or rule:observer or rule:creator
or rule:audit or rule:container_non_private_read
or rule:container_project_creator
or rule:container_project_admin or
rule:container_acl_read

Operations

- **GET** /v1/containers/{container-id}/consumers/
{consumer-id}

Scope Types

List a specific consumer for a given container.

consumers: get

Default rule:admin or rule:observer or rule:creator
or rule:audit or rule:container_non_private_read
or rule:container_project_creator
or rule:container_project_admin or
rule:container_acl_read

Operations

- **GET** /v1/containers/{container-id}/consumers

Scope Types

List a containers consumers.

consumers: post

Default rule:admin or rule:container_non_private_read
or rule:container_project_creator
or rule:container_project_admin or
rule:container_acl_read

Operations

- **POST** /v1/containers/{container-id}/consumers

Scope Types

Creates a consumer.

consumers: delete

Default rule:admin or rule:container_non_private_read
or rule:container_project_creator
or rule:container_project_admin or
rule:container_acl_read

Operations

- **DELETE** /v1/containers/{container-id}/consumers/
{consumer-id}

Scope Types

Deletes a consumer.

containers:post

Default rule:admin_or_creator

Operations

- **POST** /v1/containers

Scope Types

Creates a container.

containers:get

Default rule:all_but_audit

Operations

- **GET** /v1/containers

Scope Types

Lists a projects containers.

container:get

Default rule:container_non_private_read
or rule:container_project_creator
or rule:container_project_admin or
rule:container_acl_read

Operations

- **GET** /v1/containers/{container-id}

Scope Types

Retrieves a single container.

container:delete

Default rule:container_project_admin or
rule:container_project_creator

Operations

- **DELETE** /v1/containers/{uuid}

Scope Types

Deletes a container.

container_secret:post

Default rule:container_project_admin or
rule:container_project_creator or
rule:container_project_creator_role and
rule:container_non_private_read

Operations

- **POST** /v1/containers/{container-id}/secrets

Scope Types

Add a secret to an existing container.

container_secret:delete

Default rule:container_project_admin or
rule:container_project_creator or
rule:container_project_creator_role and
rule:container_non_private_read

Operations

- **DELETE** /v1/containers/{container-id}/secrets/{secret-id}

Scope Types

Remove a secret from a container.

orders:get

Default rule:all_but_audit

Operations

- **GET** /v1/orders

Scope Types

Gets list of all orders associated with a project.

orders:post

Default rule:admin_or_creator

Operations

- **POST** /v1/orders

Scope Types

Creates an order.

orders:put

Default rule:admin_or_creator

Operations

- **PUT** /v1/orders

Scope Types

Unsupported method for the orders API.

order:get

Default rule:all_users and project_id:%(target.order.project_id)s

Operations

- **GET** /v1/orders/{order-id}

Scope Types

Retrieves an orders metadata.

order:delete

Default rule:admin and project_id:%(target.order.project_id)s

Operations

- **DELETE** /v1/orders/{order-id}

Scope Types

Deletes an order.

quotas:get

Default rule:all_users

Operations

- **GET** /v1/quotas

Scope Types

List quotas for the project the user belongs to.

project_quotas:get

Default rule:service_admin

Operations

- **GET** /v1/project-quotas
- **GET** /v1/project-quotas/{uuid}

Scope Types

List quotas for the specified project.

project_quotas:put

Default rule:service_admin

Operations

- **PUT** /v1/project-quotas/{uuid}

Scope Types

Create or update the configured project quotas for the project with the specified UUID.

project_quotas:delete

Default rule:service_admin

Operations

- **DELETE** /v1/quotas}

Scope Types

Delete the project quotas configuration for the project with the requested UUID.

secret_meta:get

Default rule:secret_non_private_read or
rule:secret_project_creator or rule:secret_project_admin
or rule:secret_acl_read

Operations

- **GET** /v1/secrets/{secret-id}/metadata
- **GET** /v1/secrets/{secret-id}/metadata/{meta-key}

Scope Types

metadata/: Lists a secrets user-defined metadata. || metadata/{key}: Retrieves a secrets user-added metadata.

secret_meta:post

Default rule:secret_project_admin or rule:secret_project_creator or (rule:secret_project_creator_role and rule:secret_non_private_read)

Operations

- **POST** /v1/secrets/{secret-id}/metadata/{meta-key}

Scope Types

Adds a new key/value pair to the secrets user-defined metadata.

secret_meta:put

Default rule:secret_project_admin or rule:secret_project_creator or (rule:secret_project_creator_role and rule:secret_non_private_read)

Operations

- **PUT** /v1/secrets/{secret-id}/metadata
- **PUT** /v1/secrets/{secret-id}/metadata/{meta-key}

Scope Types

metadata/: Sets the user-defined metadata for a secret || metadata/{key}: Updates an existing key/value pair in the secrets user-defined metadata.

secret_meta:delete

Default rule:secret_project_admin or rule:secret_project_creator or (rule:secret_project_creator_role and rule:secret_non_private_read)

Operations

- **DELETE** /v1/secrets/{secret-id}/metadata/{meta-key}

Scope Types

Delete secret user-defined metadata by key.

secret:decrypt

Default rule:secret_decrypt_non_private_read or rule:secret_project_creator or rule:secret_project_admin or rule:secret_acl_read

Operations

- **GET** /v1/secrets/{uuid}/payload

Scope Types

Retrieve a secrets payload.

secret: get

Default rule:secret_non_private_read or
rule:secret_project_creator or rule:secret_project_admin
or rule:secret_acl_read

Operations

- **GET** /v1/secrets/{secret-id}

Scope Types

Retrieves a secrets metadata.

secret: put

Default rule:admin_or_creator and rule:secret_project_match

Operations

- **PUT** /v1/secrets/{secret-id}

Scope Types

Add the payload to an existing metadata-only secret.

secret: delete

Default rule:secret_project_admin or rule:secret_project_creator
or (rule:secret_project_creator_role and not
rule:secret_private_read)

Operations

- **DELETE** /v1/secrets/{secret-id}

Scope Types

Delete a secret by uuid.

secrets: post

Default rule:admin_or_creator

Operations

- **POST** /v1/secrets

Scope Types

Creates a Secret entity.

secrets: get

Default rule:all_but_audit

Operations

- **GET** /v1/secrets

Scope Types

Lists a projects secrets.

secretstores:get

Default rule:admin

Operations

- **GET** /v1/secret-stores

Scope Types

Get list of available secret store backends.

secretstores:get_global_default

Default rule:admin

Operations

- **GET** /v1/secret-stores/global-default

Scope Types

Get a reference to the secret store that is used as default secret store backend for the deployment.

secretstores:get_preferred

Default rule:admin

Operations

- **GET** /v1/secret-stores/preferred

Scope Types

Get a reference to the preferred secret store if assigned previously.

secretstore_preferred:post

Default rule:admin

Operations

- **POST** /v1/secret-stores/{ss-id}/preferred

Scope Types

Set a secret store backend to be preferred store backend for their project.

secretstore_preferred:delete

Default rule:admin

Operations

- **DELETE** /v1/secret-stores/{ss-id}/preferred

Scope Types

Remove preferred secret store backend setting for their project.

secretstore:get

Default rule:admin

Operations

- **GET** /v1/secret-stores/{ss-id}

Scope Types

Get details of secret store by its ID.

transport_key: get

Default rule:all_users

Operations

- **GET** /v1/transport_keys/{key-id}

Scope Types

Get a specific transport key.

transport_key: delete

Default rule:admin

Operations

- **DELETE** /v1/transport_keys/{key-id}

Scope Types

Delete a specific transport key.

transport_keys: get

Default rule:all_users

Operations

- **GET** /v1/transport_keys

Scope Types

Get a list of all transport keys.

transport_keys: post

Default rule:admin

Operations

- **POST** /v1/transport_keys

Scope Types

Create a new transport key.

2.5 Barbican for Developers

If you're new to OpenStack development you should start by reading the [OpenStack Developers Guide](#).

Once you've read the OpenStack guide you'll be ready to set up a local barbican development environment.

2.5.1 Setting up a Barbican Development Environment

These instructions are designed to help you setup a standalone version of Barbican which uses SQLite as a database backend. This is not suitable for production due to the lack of authentication and an interface to a secure encryption system such as an HSM (Hardware Security Module). In addition, the SQLite backend has known issues with thread-safety. This setup is purely to aid in development workflows.

Installing system dependencies

Ubuntu 15.10:

```
# Install development tools
sudo apt-get install git python-tox

# Install dependency build requirements
sudo apt-get install libffi-dev libssl-dev python-dev gcc
```

Fedora 30:

```
# Install development tools
sudo dnf install git python3-tox

# Install dependency build requirements
sudo dnf install gcc libffi-devel openssl-devel redhat-rpm-config
```

Setting up a virtual environment

We highly recommend using virtual environments for development. You can learn more about [Virtual Environments](#) in The Python Tutorial.

If you installed tox in the previous step you should already have virtualenv installed as well.

```
# Clone barbican source
git clone https://opendev.org/openstack/barbican
cd barbican

# Create and activate a virtual environment
virtualenv .barbicanenv
. .barbicanenv/bin/activate

# Install barbican in development mode
pip install -e $PWD
```

Configuring Barbican

Barbican uses `oslo.config` for configuration. By default the `api` process will look for the configuration file in `$HOME/barbican.conf` or `/etc/barbican/barbican.conf`. The sample configuration files included in the source code assume that you'll be using `/etc/barbican/` for configuration and `/var/lib/barbican` for the database file location.

```
# Create the directories and copy the config files
sudo mkdir /etc/barbican
sudo mkdir /var/lib/barbican
sudo chown $(whoami) /etc/barbican
sudo chown $(whoami) /var/lib/barbican
cp -r etc/barbican /etc
tox -e genconfig
cp etc/osloconfig-generator/barbican.conf /etc/barbican/barbican.conf
sed -i 's/\/v1: barbican-api-keystone/\/v1: barbican_api/' /etc/barbican/
↳barbican-api-paste.ini
```

All the locations are configurable, so you don't have to use `/etc` and `/var/lib` in your development machine if you don't want to.

Running Barbican

If you made it this far you should be able to run the barbican development server using this command:

```
bin/barbican-api
```

An instance of barbican will be listening on `http://localhost:9311`. Note that the default configuration uses the unauthenticated context. This means that requests should include the `X-Project-Id` header instead of including a keystone token in the `X-Auth-Token` header. For example:

```
curl -v -H 'X-Project-Id: 12345' \
      -H 'Accept: application/json' \
      http://localhost:9311/v1/secrets
```

For more information on configuring Barbican with Keystone auth see the [Keystone Configuration](#) page.

Building the Documentation

You can build the html documentation using `tox`:

```
tox -e docs
```

Running the Unit Tests

You can run the unit test suite using tox:

```
tox -e py36
```

2.5.2 Running Barbican on DevStack

Barbican is currently available via the plugin interface within DevStack

We provide two ways of deploying a DevStack environment with a running Barbican. The easy mode uses vagrant and automatically creates the VM with all necessary dependencies to run DevStack. It is recommended to use this process if it is your first time.

If you are familiar with DevStack you can use the steps in the manual setup section to install Barbican onto your already running DevStack installation.

Warning: This process takes anywhere from 10-30 minutes depending on your internet connection.

Easy Mode

To simplify the setup process of running Barbican on DevStack, there is a `Vagrantfile` that will automatically setup up a VM containing Barbican running on DevStack.

Warning: Upon following these steps, you will not be able to use tox tools if you setup a shared folder. This is because making hard-links is required, but not permitted if the project is in a shared folder. If you wish to use tox, comment out the *Create Synced Folder* section in `barbican/devstack/barbican-vagrant/Vagrantfile`.

1. Obtain Barbican vagrant file If you dont already have the file then clone the repo below

```
git clone https://opendev.org/openstack/barbican.git
```

2. Move the `barbican-vagrant` directory outside of the Barbican directory and into your current directory for vagrant files. If you do not have one, then just copy it into your home directory.

```
cp -r barbican/devstack/barbican-vagrant <directory>
```

3. Get into the `barbican-vagrant` directory

```
cd barbican-vagrant
```

4. Start create a new VM based on the cloned configuration

```
vagrant up
```

5. Once the VM has been successfully started and provisioned, ssh into the VM.

```
vagrant ssh
```

6. Once inside the VM, change your directory to the `devstack` folder.

```
cd /opt/stack/devstack/
```

7. Start DevStack

```
./stack.sh
```

Manual Setup

These steps assume you are running within a clean Ubuntu 14.04 virtual machine (local or cloud instance). If you are running locally, do not forget to expose the following ports

1. Barbican - 9311
2. Keystone API - 5000
3. Keystone Admin API - 5000

Installation

1. Make sure you are logged in as a non-root user with `sudo` privileges
2. Install git

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

3. Clone DevStack

```
git clone https://opendev.org/openstack/devstack.git
```

4. Add the Barbican plugin to the `local.conf` file and verify the minimum services required are included. You can pull down a specific branch by appending the name to the end of the git URL. If you leave the space empty like below, then `origin/master` will be pulled.

```
enable_plugin barbican https://opendev.org/openstack/barbican
enable_service rabbit mysql key
```

If this is your first time and you do not have a `local.conf` file, there is an example in the [Barbican GitHub](#). Copy the file and place it in the `devstack/` directory.

5. Start DevStack

```
cd devstack/
./stack.sh
```

When you're ready to dive deeper in to barbican take a look at:

2.5.3 Contributing to Barbican

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 the Barbican project, which is responsible for the following OpenStack deliverables:

barbican

The OpenStack Key Manager service.

code: <https://opendev.org/openstack/barbican>

docs: <https://docs.openstack.org/barbican>

api-ref: <https://docs.openstack.org/barbican/latest/api/index.html#api-reference>

StoryBoard: <https://storyboard.openstack.org/#!/project/openstack/barbican>

barbican-ui

Horizon extension for the OpenStack Key Manager API.

code: <https://opendev.org/openstack/barbican-ui>

StoryBoard: <https://storyboard.openstack.org/#!/project/openstack/barbican-ui>

python-barbicanclient

Python client library for the OpenStack Key Manager API.

code: <https://opendev.org/openstack/python-barbicanclient>

docs: <https://docs.openstack.org/python-barbicanclient>

StoryBoard: <https://storyboard.openstack.org/#!/project/openstack/python-barbicanclient>

barbican-tempest-plugin

Additional Barbican tempest-based tests beyond those in the main OpenStack Integration Test Suite (tempest).

code: <https://opendev.org/openstack/barbican-tempest-plugin>

StoryBoard: <https://storyboard.openstack.org/#!/project/openstack/barbican-tempest-plugin>

ansible-role-lunasa-hsm

Ansible role to manage Luna SA Hardware Security Module (HSM) client software

code: <https://opendev.org/openstack/ansible-role-lunasa-hsm>

StoryBoard: <https://storyboard.openstack.org/#!/project/openstack/ansible-role-lunasa-hsm>

See the `CONTRIBUTING.rst` file in each code repository for more information about contributing to that specific deliverable. Additionally, you should look over the docs links above; most components have helpful developer information specific to that deliverable.

Communication

IRC People working on the Barbican project may be found in the `#openstack-barbican` channel on Freenode during working hours in their timezone. The channel is logged, so if you ask a question when no one is around, you can check the log to see if its been answered: <http://eavesdrop.openstack.org/irclogs/%23openstack-barbican/>

weekly meeting Tuesdays at 13:00 UTC in `#openstack-barbican` on Freenode. Meetings are logged: <http://eavesdrop.openstack.org/meetings/barbican/>

More information (including a link to the Agenda, some pointers on meeting etiquette, and an ICS file to put the meeting on your calendar) can be found at: http://eavesdrop.openstack.org/#Barbican_Meeting

mailing list We use the openstack-discuss@lists.openstack.org mailing list for asynchronous discussions or to communicate with other OpenStack teams. Use the prefix `[barbican]` in your subject line (its a high-volume list, so most people use email filters).

More information about the mailing list, including how to subscribe and read the archives, can be found at: <http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-discuss>

meet-ups The Barbican project usually has a presence at the OpenDev/OpenStack Project Team Gathering that takes place at the beginning of each development cycle. Planning happens on an etherpad whose URL is announced at the weekly meetings and on the mailing list.

Contacting the Core Team

The barbican-core team is an active group of contributors who are responsible for directing and maintaining the Barbican project. As a new contributor, your interaction with this group will be mostly through code reviews, because only members of barbican-core can approve a code change to be merged into the code repository.

Note: Although your contribution will require reviews by members of barbican-core, these aren't the only people whose reviews matter. Anyone with a gerrit account can post reviews, so you can ask other developers you know to review your code and you can review theirs. (A good way to learn your way around the codebase is to review other peoples patches.)

If you're thinking, "I'm new at this, how can I possibly provide a helpful review?", take a look at [How to Review Changes the OpenStack Way](#).

You can learn more about the role of core reviewers in the OpenStack governance documentation: <https://docs.openstack.org/contributors/common/governance.html#core-reviewer>

The membership list of barbican-core is maintained in gerrit: <https://review.opendev.org/#/admin/groups/178,members>

New Feature Planning

The Barbican project uses both specs and blueprints to track new features. Heres a quick rundown of what they are and how the Barbican project uses them.

specs

Exist in the barbican-specs repository. Each spec must have a story in StoryBoard associated with it for tracking purposes.

A spec is required for any new Barbican core feature, anything that changes the Key Manager API, or anything that entails a mass change to the existing codebase.

The specs repository is: <https://opendev.org/openstack/barbican-specs>
It contains a `README.rst` file explaining how to file a spec.

You can read rendered specs docs at:
<https://specs.openstack.org/openstack/barbican-specs/>

blueprints

Exist in StoryBoard, where they can be targeted to release milestones.
You file one at <https://storyboard.openstack.org/#!/project/openstack/barbican-specs>

Examples of changes that can be covered by a blueprint only are:

- adding a new backend; or
- adding support for a defined capability that already exists in one or more existing backends.

Feel free to ask in `#openstack-barbican` or at the weekly meeting if you have an idea you want to develop and youre not sure whether it requires a blueprint *and* a spec or simply a blueprint.

The Barbican project observes the OpenStack-wide deadlines, for example, final release of non-client libraries (barbican), final release for client libraries (python-barbicanclient), feature freeze, etc. These are also noted and explained on the release schedule for the current development cycle.

Task Tracking

We track our tasks in [StoryBoard](#). See the top of the page for the URL of each Barbican project deliverable.

If youre looking for some smaller, easier work item to pick up and get started on, search for the low-hanging-fruit tag in the Bugs section.

When you start working on a bug, make sure you assign it to yourself. Otherwise someone else may also start working on it, and we dont want to duplicate efforts. Also, if you find a bug in the code and want to post a fix, make sure you file a bug (and assign it to yourself!) just in case someone else comes across the problem in the meantime.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so in the StoryBoard of the affected deliverable.

Getting Your Patch Merged

The Barbican project policy is that a patch must have two +2s before it can be merged. (Exceptions are documentation changes, which require only a single +2, and specs, for which the PTL may require more than two +2s, depending on the complexity of the proposal.)

Patches lacking unit tests are unlikely to be approved. Check out the testing-barbican section of the Barbican Contributors Guide for a discussion of the kinds of testing we do with barbican.

In addition, some changes may require a release note. Any patch that changes functionality, adds functionality, or addresses a significant bug should have a release note. You can find more information about how to write a release note in the release-notes section of the Barbican Contributors Guide.

Keep in mind that the best way to make sure your patches are reviewed in a timely manner is to review other peoples patches. Were engaged in a cooperative enterprise here.

You can see whos been doing what with Barbican recently in Stackalytics: <https://www.stackalytics.com/report/activity?module=barbican-group>

Project Team Lead Duties

All common PTL duties are enumerated in the [PTL guide](#).

2.5.4 Getting Involved

The best way to join the community and get involved is to talk with others online or at a meetup and offer contributions. Here are some of the many ways you can contribute to the Barbican project:

- Development and Code Reviews
- Bug reporting/Bug fixes
- Wiki and Documentation
- Blueprints/Specifications
- Testing
- Deployment scripts

Freenode IRC (Chat)

You can find Barbicaners in our publicly accessible channel on [freenode](#) #openstack-barbican. All conversations are logged and stored for your convenience at [eavesdrop.openstack.org](#). For more information regarding OpenStack IRC channels please visit the [OpenStack IRC Wiki](#).

Mailing List

The mailing list email is openstack@lists.openstack.org. This is a common mailing list across the OpenStack projects. If you wish to ask questions or have a discussion related to Barbican include [barbican] in your email subject line. To participate on the mailing list:

- [Subscribe](#) to the mailing list
- Browse the [mailing list archives](#)

Launchpad

Most of the tools used for OpenStack require a [launchpad.net](#) ID for authentication.

Storyboard

The barbican project moved from Launchpad to [Storyboard](#) for work and task tracking. This provides an aggregate view called a Project Group and individual Projects. A good starting place is the [project group](#) representing the whole of the barbican community, as opposed to the [barbican project](#) storyboard which represents barbican as a repository.

Source Repository

Like other OpenStack related projects, the official Git repository is available on [opendev.org](#); however, the repository is also mirrored to GitHub for easier browsing.

- [Barbican on GitHub](#)

Gerrit

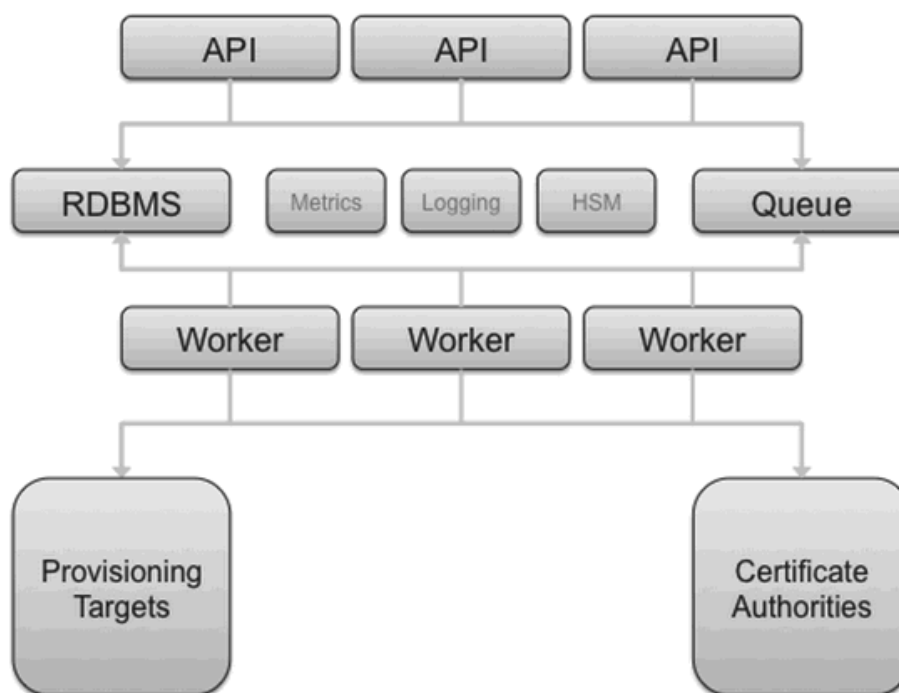
Like other OpenStack related projects, we utilize the OpenStack Gerrit review system for all code reviews. If you're unfamiliar with using the OpenStack Gerrit review system, please review the [Gerrit Workflow](#) wiki documentation.

2.5.5 Architecture

This document describes the architecture and technology selections for Barbican. In general, a goal is to utilize the OpenStack architecture and technology selections as much as possible. An overall architecture is presented first, followed by technology selection details to implement the system.

Overall Architecture

The next figure presents an overall logical diagram for Barbican.

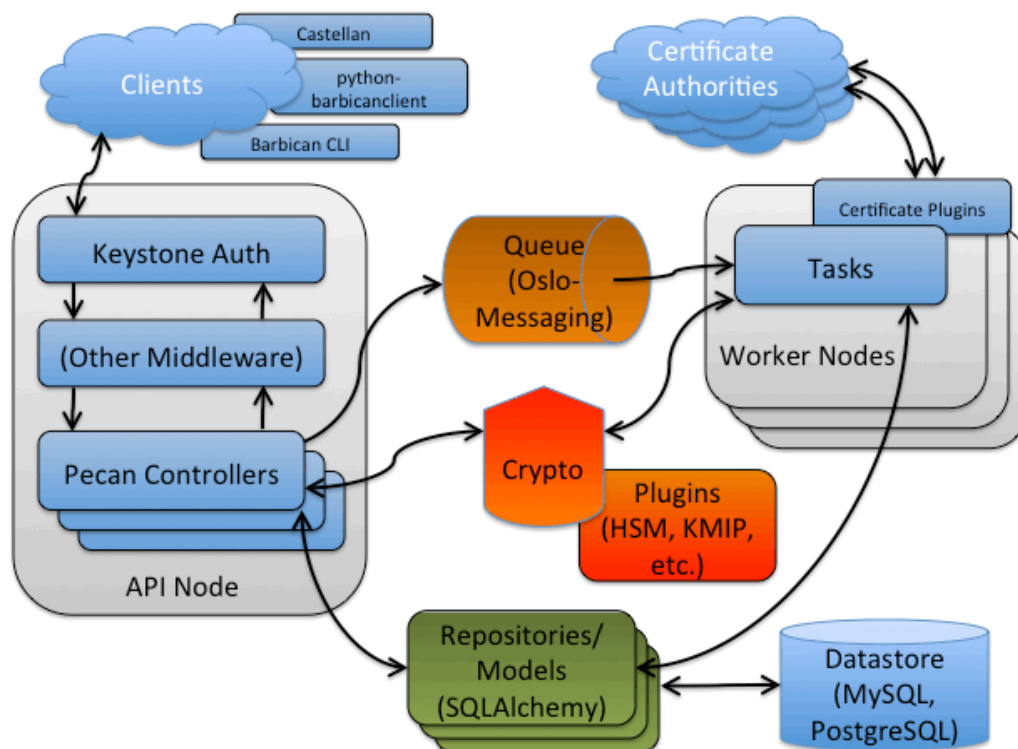


The API node(s) handle incoming REST requests to Barbican. These nodes can interact with the database directly if the request can be completed synchronously (such as for GET requests), otherwise the queue supports asynchronous processing by worker nodes. The latter could include interactions with third parties such as certificate authorities. As implied in the diagram, the architecture supports multiple API and worker nodes being added/removed to/from the network, to support advanced features such as auto scaling. Eventually, the database could be replicated across data centers supporting region-agnostic storage and retrieval of secured information, albeit with lags possible during data synchronization.

Technology Selection

In general, components from the [Oslo commons](#) project are used within Barbican, such as config, messaging and logging.

The next figure examines the components within Barbican.



Several potential clients of the Barbican REST interface are noted, including [Castellan](#) which presents a generic key management interface for other OpenStack projects with Barbican as an available plugin.

The API node noted in the previous section is a WSGI server. Similar to OpenStack projects such as [Glance](#) it utilizes [paste](#) to support configurable middleware such as to interface with [Keystone](#) for authentication and authorization services. [Pecan](#) (a lean Python web framework inspired by [CherryPy](#), [TurboGears](#), and [Pylons](#)) is utilized to map resources to REST routes. These resources contain the controller business logic for Barbican and can interface with encryption/decryption processes (via crypto components), datastore (via repository components) and asynchronous tasks (via queue components).

The crypto components provide a means to encrypt and decrypt information that accommodates a variety of encryption mechanisms and cryptographic backends (such as key management interoperability protocol (KMIP) or hardware security module (HSM)) via a plugin interface.

The repository components provide an interface and database session context for the datastore, with model components representing entities such as Secrets (used to store encrypted information such as data encryption keys). [SQLAlchemy](#) is used as the object relational model (ORM) layer to the database, including [MySQL](#) and [PostgreSQL](#).

For asynchronous processing, [Oslo Messaging](#) is used to interact with the queue, including [RabbitMQ](#). The worker node processes tasks from the queue. Task components are similar to API resources in that

they implement business logic and also interface with the datastore and follow on asynchronous tasks as needed. These asynchronous tasks can interface with external systems, such as certificate authorities for SSL/TLS certificate processing.

2.5.6 Project Structure

1. `barbican/` (Barbican-specific Python source files)
 1. `api/` (REST API related source files)
 1. `controllers/` (Pecan-based controllers handling REST-based requests)
 2. `middleware/` (Middleware business logic to process REST requests)
 2. `cmd/` (Barbican admin command source files)
 3. `common/` (Modules shared across other Barbican folders)
 4. `locale/` (Translation templates)
 5. `model/` (SQLAlchemy-based model classes)
 6. `plugin/` (Plugin related logic, interfaces and look-up management)
 1. `resources.py` (Supports interactions with plugins)
 2. `crypto/` (Hardware security module (HSM) logic and plugins)
 3. `interface/` (Certificate manager and secret store interface classes)
 4. (The remaining modules here are implementations of above interfaces)
 7. `queue/` (Client and server interfaces to the queue)
 1. `client.py` (Allows clients to publish tasks to queue)
 2. `server.py` (Runs the worker service, responds to enqueued tasks)
 8. `tasks/` (Worker-related controllers and implementations)
 9. `tests/` (Unit tests)
2. `bin/` (Start-up scripts for the Barbican nodes)
3. **`devstack/`** (Barbican DevStack plugin, DevStack gate configuration and Vagrantfile for installing DevStack VM)
4. `etc/barbican/` (Configuration files)
5. `functionaltests` (Functional Barbican tests)
6. `doc/source` (Sphinx documentation)
7. `releasenotes` (Barbican Release Notes)

2.5.7 Dataflow

Bootup flow when the Barbican API service begins

This is the sequence of calls for booting up the Barbican API server:

1. `bin/barbican.sh start`: Launches a WSGI service that performs a PasteDeploy process, invoking the middleware components found in `barbican/api/middleware` as configured in `etc/barbican/barbican-api-paste`. The middleware components invoke and then execute the Pecan application created via `barbican/api/app.py:create_main_app()`, which also defines the controllers (defined in `barbican/api/controllers/`) used to process requested URI routes.

Typical flow when the Barbican API executes

For **synchronous** calls, the following sequence is generally followed:

1. A client sends an HTTP REST request to the Barbican API server.
2. The WSGI server and routing invokes a method on one of the `XxxxController` classes in `barbican/api/controllers/xxxx.py`, keyed to an HTTP verb (so one of POST, GET, DELETE, or PUT).
 1. Example - GET /secrets:
 1. In `barbican/api/controllers/secrets.py`, the `SecretControllers.on_get()` is invoked.
 2. A `SecretRepo` repository class (found in `barbican/model/respositories.py`) is then used to retrieve the entity of interest, in this case as a `Secret` entity defined in `barbican/model/models.py`.
 3. The payload is decrypted as needed, via `barbican/plugin/resources.py` `get_secret()` function.
 4. A response JSON is formed and returned to the client.

For **asynchronous** calls, the following sequence is generally followed:

1. A client sends an HTTP REST request to the Barbican API server.
2. The WSGI server and routing again invokes a method on one of the `XxxxxController` classes in `barbican/api/controllers/`.
3. A remote procedure call (RPC) task is enqueue for later processing by a worker node.
 1. Example - POST /orders:
 1. In `barbican/api/controllers/orders.py`, the `OrdersControllers.on_post()` is invoked.
 2. The `OrderRepo` repository class (found in `barbican/model/respositories.py`) is then used to create the `barbican/model/models.py` `Order` entity in a `PENDING` state.
 3. The `Queue APIs` `process_type_order()` method on the `TaskClient` class (found in `barbican/queue/client.py`) is invoked to send a message to the queue for asynchronous processing.

4. A response JSON is formed and returned to the client.
4. The Queue service receives the message sent above, invoking a corresponding method on `barbican/queue/server.py` `Tasks` class. This method then invokes the `process_and_suppress_exceptions()` method on one of the `barbican/tasks/resources.py` `BaseTask` implementors. This method can then utilize repository classes as needed to retrieve and update entities. It may also interface with third party systems via plugins'. The `barbican/queue/client.py` `TaskClient` class above may also be invoked from a worker node for follow on asynchronous processing steps.

1. Example - POST /orders (continued):

1. Continuing the example above, the queue would invoke the `process_type_order()` method on `barbican/queue/server.py` `Tasks` class. Note the method is named the same as the `TaskClient` method above by convention.
2. This method then invokes `process_and_suppress_exceptions()` on the `barbican/tasks/resources.py` `BeginTypeOrder` class. This class is responsible for processing all newly-POST-ed orders.

2.5.8 Adding/Updating Dependencies

Adding new Dependency

If you need to add a new dependency to Barbican, you must edit a few things:

1. Add the package name (and minimum version if applicable) to the `requirements.txt` file in the root directory.

Note: All dependencies and their version specifiers must come from the OpenStack [global requirements](#) repository.

2. We support deployment on CentOS 6.4, so you should check CentOS + EPEL 6 yum repos to figure out the name of the rpm package that provides the package youre adding. Add this package name as a dependency in `rpmbuild/SPECS/barbican.spec`.
3. If there is no package available in CentOS or EPEL, or if the latest available packages version is lower than the minimum required version we must build an rpm for it ourselves. Add a line to `rpmbuild/package_dependencies.sh` so that jenkins will build an rpm using fpm and upload it to the cloudkeep yum repo.

2.5.9 Database Migrations

Database migrations are managed using the [Alembic](#) library. The consensus for [OpenStack](#) and [SQLAlchemy](#) is that this library is preferred over `sqlalchemy-migrate`.

Database migrations can be performed two ways: (1) via the API startup process, and (2) via a separate script.

Database migrations can be optionally enabled during the API startup process. Corollaries for this are that a new deployment should begin with only one node to avoid migration race conditions.

Policy

A Barbican deployment goal is to update application and schema versions with zero downtime. The challenge is that at all times the database schema must be able to support two deployed application versions, so that a single migration does not break existing nodes running the previous deployment. For example, when deleting a column we would first deploy a new version that ignores the column. Once all nodes are ignoring the column, a second deployment would be made to remove the column from the database.

To achieve this goal, the following rules will be observed for schema changes:

1. Do not remove columns or tables directly, but rather:
 - a. Create a version of the application not dependent on the removed column/table
 - b. Replace all nodes with this new application version
 - c. Create an Alembic version file to remove the column/table
 - d. Apply this change in production manually, or automatically with a future version of the application
2. Changing column attributes (types, names or widths) should be handled as follows:
 - a. TODO: This Stack Overflow [Need to alter column types in production database](#) page and many others summarize the grief involved in doing these sorts of migrations
 - b. TODO: What about old and new application versions happening simultaneously?
 - i. Maybe have the new code perform migration to new column on each read similar to how a no-sql db migration would occur?
3. Transforming column attributes (ex: splitting one name column into a first and last name):
 - a. TODO: An [Alembic example](#), but not robust for large datasets.

Overview

Prior to invoking any migration steps below, change to your `barbican projects` folder and activate your virtual environment per the [Developer Guide](#).

If you are using PostgreSQL, please ensure you are using SQLAlchemy version 0.9.3 or higher, otherwise the generated version files will not be correct.

You cannot use these migration tools and techniques with SQLite databases.

Consider taking a look at the [Alembic tutorial](#). As a brief summary: Alembic keeps track of a linked list of version files, each one applying a set of changes to the database schema that a previous version file in the linked list modified. Each version file has a unique Alembic-generated ID associated with it. Alembic generates a table in the project table space called `alembic_version` that keeps track of the unique ID of the last version file applied to the schema. During an update, Alembic uses this stored version ID to determine what if any follow on version files to process.

Generating Change Versions

To make schema changes, new version files need to be added to the `barbican/model/migration/alembic_migrations/versions/` folder. This section discusses two ways to add these files.

Automatically

Alembic autogenerates a new script by comparing a clean database (i.e., one without your recent changes) with any modifications you make to the `Models.py` or other files. This being said, automatic generation may miss changes it is more of an automatic assist with expert review. See [What does Auto-generate Detect](#) in the Alembic documentation for more details.

First, you must start Barbican using a version of the code that does not include your changes, so that it creates a clean database. This example uses Barbican launched with DevStack (see [Barbican DevStack](#) wiki page for instructions).

1. Make changes to the `barbican/model/models.py` SQLAlchemy models or checkout your branch that includes your changes using git.
2. Execute

```
barbican-db-manage -d <Full URL to database, including user/pw> revision -m '<your-summary-of-changes>' --autogenerate
```

 - a. For example:

```
barbican-db-manage -d mysql+pymysql://root:password@127.0.0.1/barbican?charset=utf8 revision -m 'Make unneeded verification columns nullable' --autogenerate
```
3. Examine the generated version file, found in `barbican/model/migration/alembic_migrations/versions/`:
 - a. **Verify generated update/rollback steps, especially for modifications to existing columns/tables**
 - b. Remove autogenerated comments such as: `### commands auto generated by Alembic - please adjust! ###`
 - c. **If you added new columns, follow this guidance:**
 1. For non-nullable columns you will need to add default values for the records already in the table, per what you configured in the `barbican.model.models.py` module. You can add the `server_default` keyword argument for the SQLAlchemy `Column` call per [SQLAlchemy's server_default](#). For boolean attributes, use `server_default=0` for False, or `server_default=1` for True. For DateTime attributes, use `server_default=str(timeutils.utcnow())` to default to the current time.
 2. If you add *any* constraint, please *always* name them in the `barbican.model.models.py` module, and also in the Alembic version modules when creating/dropping constraints, otherwise MySQL migrations might crash.
 - d. **If you added new tables, follow this guidance:**
 1. Make sure you added your new table to the `MODELS` element of the `barbican/model/models.py` module.
 2. Note that when Barbican boots up, it will add the new table to the database. It will also try to apply the database version (that also tries to add this table) via alembic. Therefore, please edit the generated script file to add these lines:

- a. `ctx = op.get_context()` (to get the alembic migration context in current transaction)
- b. `con = op.get_bind()` (get the database connection)
- c. `table_exists = ctx.dialect.has_table(con.engine, 'your-new-table-name-here')`
- d. `if not table_exists:`
- e. `...remaining create table logic here...`

Note: For anything but trivial or brand new columns/tables, database backups and maintenance-window downtimes might be called for.

Manually

1. Execute: `barbican-db-manage revision -m "<insert your change description here>"`
2. This will generate a new file in the `barbican/model/migration/alembic_migrations/versions/` folder, with this sort of file format: `<unique-Alembic-ID>_<your-change-description-from-above-but-truncated>.py`. Note that only the first 20 characters of the description are used.
3. You can then edit this file per tutorial and the [Alembic Operation Reference](#) page for available operations you may make from the version files. **You must properly fill in the `upgrade()` methods.**

Applying Changes

Barbican utilizes the Alembic version files as managing delta changes to the database. Therefore the first Alembic version file does **not** contain all time-zero database tables.

To create the initial Barbican tables in the database, execute the Barbican application per the [Via Application](#) section.

Thereafter, it is suggested that only the `barbican-db-manage` command above be used to update the database schema per the [Manually](#) section. Also, automatic database updates from the Barbican application should be disabled by adding/updating `db_auto_create = False` in the `barbican.conf` configuration file.

Note : Before attempting any upgrade, you should make a full database backup of your production data. As of Kilo, database downgrades are not supported in OpenStack, and the only method available to get back to a prior database version will be to restore from backup.

Via Application

The last section of the [Alembic tutorial](#) describes the process used by the Barbican application to create and update the database table space automatically.

By default, when the Barbican API boots up it will try to create the Barbican database tables (using SQLAlchemy), and then try to apply the latest version files (using Alembic). In this mode, the latest version of the Barbican application can create a new database table space updated to the latest schema version, or else it can update an existing database table space to the latest schema revision (called `head` in the docs).

To bypass this automatic behavior, add `db_auto_create = False` to the `barbican.conf` file.

Manually

Run `barbican-db-manage -d <Full URL to database, including user/pw> upgrade -v head`, which will cause Alembic to apply the changes found in all version files after the version currently written in the target database, up until the latest version file in the linked chain of files.

To upgrade to a specific version, run this command: `barbican-db-manage -d <Full URL to database, including user/pw> upgrade -v <Alembic-ID-of-version>`. The `Alembic-ID-of-version` is a unique ID assigned to the change such as `as1a0c2cdafb38`.

Downgrade

Upgrades involve complex operations and can fail. Before attempting any upgrade, you should make a full database backup of your production data. As of Kilo, database downgrades are not supported, and the only method available to get back to a prior database version will be to restore from backup.

You must complete these steps to successfully roll back your environment:

1. Roll back configuration files.
2. Restore databases from backup.
3. Roll back packages.

Rolling back upgrades is a tricky process because distributions tend to put much more effort into testing upgrades than downgrades. Broken downgrades often take significantly more effort to troubleshoot and resolve than broken upgrades. Only you can weigh the risks of trying to push a failed upgrade forward versus rolling it back. Generally, consider rolling back as the very last option.

The backup instructions provided in [Backup tutorial](#) ensure that you have proper backups of your databases and configuration files. Read through this section carefully and verify that you have the requisite backups to restore.

Note : The backup tutorial reference file only updated to Juno, DB backup operation will be similar for Kilo. The link will be updated when the reference has updated.

For more information and examples about downgrade operation please see [Downgrade tutorial](#) as reference.

TODO Items

1. *[Done - It works!]* Verify alembic works with the current SQLAlchemy model configuration in Barbican (which was borrowed from Glance).
2. *[Done - It works, I was able to add/remove columns while app was running]* Verify that SQLAlchemy is tolerant of schema miss-matches. For example, if a column is added to a table schema, will this break existing deployments that aren't expecting this column?
3. *[Done - It works]* Add auto-migrate code to the boot up of models (see the `barbican\model\repositories.py` file).
4. *[Done - It works]* Add guard in Barbican model logic to guard against running migrations with SQLite databases.
5. Add detailed deployment steps for production, so how new nodes are rolled in and old ones rolled out to complete move to new versions.
6. *[In Progress]* Add a best-practices checklist section to this page.
 - a. This would provide guidance on safely migrating schemas, dos and donts, etc.
 - b. This could also provide code guidance, such as ensuring that new schema changes (eg. that new column) aren't required for proper functionality of the previous version of the code.
 - c. If a server bounce is needed, notification guidelines to the devop team would be spelled out here.

2.5.10 Plugin Developers Guide

This guide describes how to develop custom plugins for use by Barbican. While Barbican provides useful plugin implementations, some OpenStack operators may require customized implementations, perhaps to interact with an existing corporate database or service. This approach also gives flexibility to operators of OpenStack clouds by allowing them to choose the right implementation for their cloud.

Plugin Status

A Barbican plugin may be considered `stable`, `experimental` or `out-of-tree`.

- A *stable* status indicates that the plugin is fully supported by the OpenStack Barbican Team
- An *experimental* status indicates that we intend to support the plugin, but it may be missing features or may not be fully tested at the gate. Plugins in this status may occasionally break.
- An *out-of-tree* status indicates that no formal support will be provided, and the plugin may be removed in a future release.

Graduation Process

By default, new plugins proposed to be in-tree will be in the *experimental* status. To be considered *stable* a plugin must meet the following requirements:

- 100% unit test coverage, including branch coverage.
- Gate job that executes the functional test suite against an instance of Barbican configured to use the plugin. The gate may be a devstack gate, or a third-party gate.
- Implement new features within one cycle after the new blueprint feature is approved.

Demotion Process

Plugins should not stay in the *experimental* status for a long time. Plugins that stay in *experimental* for more than **two** releases are expected to move into *stable*, as described by the Graduation Process, or move into *out-of-tree*.

Plugins in the *stable* status may be deprecated by the team, and moved to *out-of-tree*.

Plugins that stay in the *out-of-tree* status for more than **two** releases may be removed from the tree.

Architecture

Barbicans plugin architecture enables developers to create their own implementations of features such as secret storage and generation and event handling. The plugin pattern used defines an abstract class, whose methods are invoked by Barbican logic (referred to as Barbican core in this guide) in a particular sequence. Typically plugins do not interact with Barbicans data model directly, so Barbican core also handles persisting any required information on the plugins behalf.

In general, Barbican core will invoke a variation of the plugins `supports()` method to determine if a requested action can be implemented by the plugin. Once a supporting plugin is selected, Barbican core will invoke one or more methods on the plugin to complete the action.

The links below provide further guidance on the various plugin types used by Barbican, as well as configuration and deployment options.

Secret Store Plugin Development

This guide describes how to develop a custom secret store plugin for use by Barbican.

Barbican supports two storage modes for secrets: a secret store mode (detailed on this page), and a *cryptographic mode*. The secret store mode offloads both encryption/decryption and encrypted secret storage to the plugin implementation. Barbican includes plugin interfaces to a Red Hat Dogtag service and to a Key Management Interoperability Protocol (KMIP) compliant security appliance.

Since the secret store mode defers the storage of encrypted secrets to plugins, Barbican core does not need to store encrypted secrets into its data store, unlike the *cryptographic mode*. To accommodate the discrepancy between the two secret storage modes, a secret store to cryptographic plugin adapter has been included in Barbican core, as detailed in *The Cryptographic Plugin Adapter* section below.

secret_store Module

The `barbican.plugin.interface.secret_store` module contains the classes needed to implement a custom plugin. These classes include the `SecretStoreBase` abstract base class which custom plugins should inherit from, as well as several Data Transfer Object (DTO) classes used to transfer data between Barbican and the plugin.

Data Transfer Objects

The DTO classes are used to wrap data that is passed from Barbican to the plugin as well as data that is returned from the plugin back to Barbican. They provide a level of isolation between the plugins and Barbicans internal data models.

```
class barbican.plugin.interface.secret_store.SecretDTO (type, secret,  
key_spec, content_type,  
transport_key=None)
```

This object is a secret data transfer object (DTO).

This object encapsulates a key and attributes about the key. The attributes include a `KeySpec` that contains the algorithm and bit length. The attributes also include information on the encoding of the key.

```
class barbican.plugin.interface.secret_store.AsymmetricKeyMetadataDTO (private_key_meta,  
public_key_meta,  
license_key_meta,  
passphrase_meta)
```

This DTO encapsulates metadata(s) for asymmetric key components.

These components are `private_key_meta`, `public_key_meta` and `passphrase_meta`.

Secret Parameter Objects

The secret parameter classes encapsulate information about secrets to be stored within Barbican and/or its plugins.

```
class barbican.plugin.interface.secret_store.SecretType  
Constant to define the symmetric key type.
```

Used by `getSecret` to retrieve a symmetric key.

```
class barbican.plugin.interface.secret_store.KeyAlgorithm  
Constant for the Diffie Hellman algorithm.
```

```
class barbican.plugin.interface.secret_store.KeySpec (alg=None,  
bit_length=None,  
mode=None,  
passphrase=None)
```

This object specifies the algorithm and bit length for a key.

Plugin Base Class

Barbican secret store plugins should implement the abstract base class `SecretStoreBase`. Concrete implementations of this class should be exposed to Barbican using `stevedore` mechanisms explained in the configuration portion of this guide.

```
class barbican.plugin.interface.secret_store.SecretStoreBase
```

```
abstract delete_secret (secret_metadata)
```

Deletes a secret from the secret store.

Deletes a secret from a secret store. It can no longer be referenced after this call.

Parameters `secret_metadata` `secret_metadata`

```
abstract generate_asymmetric_key (key_spec)
```

Generate a new asymmetric key pair and store it.

Generates a new asymmetric key pair and stores it in the secret store. An object of type `AsymmetricKeyMetadataDTO` will be returned containing attributes of metadata for newly created key pairs. The metadata is stored by Barbican and passed into other methods to aid the plugins. This can be useful for plugins that generate a unique ID in the external data store and use it to retrieve the key pairs in the future.

Parameters `key_spec` `KeySpec` that contains details on the type of key to generate

Returns An object of type `AsymmetricKeyMetadataDTO` containing metadata about the key pair.

```
abstract generate_supports (key_spec)
```

Returns a boolean indicating if the secret type is supported.

This checks if the algorithm and bit length are supported by the generate methods. This is useful to call before calling `generate_symmetric_key` or `generate_asymmetric_key` to see if the key type is supported before trying to generate it.

Parameters `key_spec` `KeySpec` that contains details on the algorithm and bit length

Returns boolean indicating if the algorithm is supported

```
abstract generate_symmetric_key (key_spec)
```

Generate a new symmetric key and store it.

Generates a new symmetric key and stores it in the secret store. A dictionary is returned that contains metadata about the newly created symmetric key. The dictionary of metadata is stored by Barbican and passed into other methods to aid the plugins. This can be useful for plugins that generate a unique ID in the external data store and use it to retrieve the key in the future. The returned dictionary may be empty if the `SecretStore` does not require it.

Parameters `key_spec` `KeySpec` that contains details on the type of key to generate

Returns an optional dictionary containing metadata about the key

```
abstract get_plugin_name ()
```

Gets user friendly plugin name.

This plugin name is expected to be read from config file. There will be a default defined for plugin name which can be customized in specific deployment if needed.

This name needs to be unique across a deployment.

abstract get_secret (*secret_type, secret_metadata*)

Retrieves a secret from the secret store.

Retrieves a secret from the secret store and returns a SecretDTO that contains the secret.

The *secret_metadata* parameter is the metadata returned from one of the generate or store methods. This data is used by the plugins to retrieve the key.

The *secret_type* parameter may be useful for secret stores to know the expected format of the secret. For instance if the type is SecretDTO.PRIVATE then a PKCS8 structure is returned. This way secret stores do not need to manage the secret type on their own.

Parameters

- **secret_type** secret type
- **secret_metadata** secret metadata

Returns SecretDTO that contains secret

get_transport_key ()

Gets a transport key.

Returns the current valid transport key associated with this plugin. The transport key is expected to be a base64 encoded x509 certificate containing a public key. Admins are responsible for deleting old keys from the database using the DELETE method on the TransportKey resource.

By default, returns None. Plugins that support transport key wrapping should override this method.

is_transport_key_current (*transport_key*)

Determines if the provided transport key is the current valid key

Returns true if the transport key is the current valid transport key. If the key is not valid, then barbican core will request a new transport key from the plugin.

Returns False by default. Plugins that support transport key wrapping should override this method.

abstract store_secret (*secret_dto*)

Stores a key.

The SecretDTO contains the bytes of the secret and properties of the secret. The SecretStore retrieves the secret bytes, stores them, and returns a dictionary of metadata about the secret. This can be useful for plugins that generate a unique ID in the external data store and use it to retrieve the secret in the future. The returned dictionary may be empty if the SecretStore does not require it.

Parameters **secret_dto** SecretDTO for secret

Returns an optional dictionary containing metadata about the secret

abstract store_secret_supports (*key_spec*)

Returns a boolean indicating if the secret can be stored.

Checks if the secret store can store the secret, give the attributes of the secret in the `KeySpec`. For example, some plugins may need to know the attributes in order to store the secret, but other plugins may be able to store the secret as a blob if no attributes are given.

Parameters `key_spec` `KeySpec` for the secret

Returns a boolean indicating if the secret can be stored

Barbican Core Plugin Sequence

The sequence that Barbican invokes methods on `SecretStoreBase` depends on the requested action as detailed next. Note that these actions are invoked via the `barbican.plugin.resources` module, which in turn is invoked via Barbicans API and Worker processes.

For secret storage actions, Barbican core calls the following methods:

1. `get_transport_key()` - If a transport key is requested to upload secrets for storage, this method asks the plugin to provide the transport key.
2. `store_secret_supports()` - Asks the plugin if it can support storing a secret based on the `KeySpec` parameter information as described above.
3. `store_secret()` - Asks the plugin to perform encryption of an unencrypted secret payload as provided in the `SecretDTO` above, and then to store that secret. The plugin then returns a dictionary of information about that secret (typically a unique reference to that stored secret that only makes sense to the plugin). Barbican core will then persist this dictionary as a JSON attribute within its data store, and also hand it back to the plugin for secret retrievals later. The name of the plugin used to perform this storage is also persisted by Barbican core, to ensure we retrieve this secret only with this plugin.

For secret retrievals, Barbican core will select the same plugin as was used to store the secret, and then invoke its `get_secret()` method to return the unencrypted secret.

For symmetric key generation, Barbican core calls the following methods:

1. `generate_supports()` - Asks the plugin if it can support generating a symmetric key based on the `KeySpec` parameter information as described above.
2. `generate_symmetric_key()` - Asks the plugin to both generate and store a symmetric key based on the `KeySpec` parameter information. The plugin can then return a dictionary of information for the stored secret similar to the storage process above, which Barbican core will persist for later retrieval of this generated secret.

For asymmetric key generation, Barbican core calls the following methods:

1. `generate_supports()` - Asks the plugin if it can support generating an asymmetric key based on the `KeySpec` parameter information as described above.
2. `generate_asymmetric_key()` - Asks the plugin to both generate and store an asymmetric key based on the `KeySpec` parameter information. The plugin can then return an `AsymmetricKeyMetadataDTO` object as described above, which contains secret metadata for each of the three secrets generated and stored by this plugin: private key, public key and an optional passphrase. Barbican core will then persist information for these secrets, and also create a container to group them.

The Cryptographic Plugin Adapter

Barbican core includes a specialized secret store plugin used to adapt to cryptographic plugins, called `StoreCryptoAdapterPlugin`. This plugin functions as a secret store plugin, but it directs secret related operations to *cryptographic plugins* for encryption/decryption/generation operations. Because cryptographic plugins do not store encrypted secrets, this adapter plugin provides this storage capability via Barbicans data store.

This adapter plugin also uses `stevedore` to access and utilize cryptographic plugins that can support secret operations.

Cryptographic Plugin Development

This guide describes how to develop a custom cryptographic plugin for use by Barbican.

Barbican supports two storage modes for secrets: a cryptographic mode (detailed on this page), and a *secret store mode*. The cryptographic mode stores encrypted secrets in Barbicans data store, utilizing a cryptographic process or appliance (such as a hardware security module (HSM)) to perform the encryption/decryption. Barbican includes a PKCS11-based interface to SafeNet HSMs.

Note that cryptographic plugins are not invoked directly from Barbican core, but rather via a *secret store mode* plugin adapter class, further described in *The Cryptographic Plugin Adapter*.

crypto Module

The `barbican.plugin.crypto` module contains the classes needed to implement a custom plugin. These classes include the `CryptoPluginBase` abstract base class which custom plugins should inherit from, as well as several Data Transfer Object (DTO) classes used to transfer data between Barbican and the plugin.

Data Transfer Objects

The DTO classes are used to wrap data that is passed from Barbican to the plugin as well as data that is returned from the plugin back to Barbican. They provide a level of isolation between the plugins and Barbicans internal data models.

```
class barbican.plugin.crypto.base.KEKMetaDTO (kek_datum)
    Key Encryption Key Meta DTO
```

Key Encryption Keys (KEKs) in Barbican are intended to represent a distinct key that is used to perform encryption on secrets for a particular project.

`KEKMetaDTO` objects are provided to cryptographic backends by Barbican to allow plugins to persist metadata related to the projects KEK.

For example, a plugin that interfaces with a Hardware Security Module (HSM) may want to use a different encryption key for each project. Such a plugin could use the `KEKMetaDTO` object to save the key ID used for that project. Barbican will persist the KEK metadata and ensure that it is provided to the plugin every time a request from that same project is processed.

plugin_name

String attribute used by Barbican to identify the plugin that is bound to the KEK metadata. Plugins should not change this attribute.

kek_label

String attribute used to label the projects KEK by the plugin. The value of this attribute should be meaningful to the plugin. Barbican does not use this value.

algorithm

String attribute used to identify the encryption algorithm used by the plugin. e.g. AES, 3DES, etc. This value should be meaningful to the plugin. Barbican does not use this value.

mode

String attribute used to identify the algorithm mode used by the plugin. e.g. CBC, GCM, etc. This value should be meaningful to the plugin. Barbican does not use this value.

bit_length

Integer attribute used to identify the bit length of the KEK by the plugin. This value should be meaningful to the plugin. Barbican does not use this value.

plugin_meta

String attribute used to persist any additional metadata that does not fit in any other attribute. The value of this attribute is defined by the plugin. It could be used to store external system references, such as Key IDs in an HSM, URIs to an external service, or any other data that the plugin deems necessary to persist. Because this is just a plain text field, a plugin may even choose to persist data such as key value pairs in a JSON object.

class `barbican.plugin.crypto.base.EncryptDTO` (*unencrypted*)

Secret Encryption DTO

Data Transfer Object used to pass all the necessary data for the plugin to perform encryption of a secret.

Currently, this DTO only contains the raw bytes to be encrypted by the plugin, but in the future this may contain more information.

unencrypted

The secret data in Bytes to be encrypted by the plugin.

class `barbican.plugin.crypto.base.DecryptDTO` (*encrypted*)

Secret Decryption DTO

Data Transfer Object used to pass all the necessary data for the plugin to perform decryption of a secret.

Currently, this DTO only contains the data produced by the plugin during encryption, but in the future this DTO will contain more information, such as a transport key for secret wrapping back to the client.

encrypted

The data that was produced by the plugin during encryption. For some plugins this will be the actual bytes that need to be decrypted to produce the secret. In other implementations, this may just be a reference to some external system that can produce the unencrypted secret.

class `barbican.plugin.crypto.base.GenerateDTO` (*algorithm*, *bit_length*,
mode, *passphrase=None*)

Secret Generation DTO

Data Transfer Object used to pass all the necessary data for the plugin to generate a secret on behalf of the user.

generation_type

String attribute used to identify the type of secret that should be generated. This will be either "symmetric" or "asymmetric".

algorithm

String attribute used to specify what type of algorithm the secret will be used for. e.g. "AES" for a "symmetric" type, or "RSA" for "asymmetric".

mode

String attribute used to specify what algorithm mode the secret will be used for. e.g. "CBC" for "AES" algorithm.

bit_length

Integer attribute used to specify the bit length of the secret. For example, this attribute could specify the key length for an encryption key to be used in AES-CBC.

```
class barbican.plugin.crypto.base.ResponseDTO(cypher_text,  
kek_meta_extended=None)
```

Data transfer object for secret generation response.

Barbican guarantees that both the `cypher_text` and `kek_metadata_extended` will be persisted and then given back to the plugin when requesting a decryption operation.

`kek_metadata_extended` takes the idea of Key Encryption Key (KEK) metadata further by giving plugins the option to store secret-level KEK metadata. One example of using secret-level KEK metadata would be plugins that want to use a unique KEK for every secret that is encrypted. Such a plugin could use `kek_metadata_extended` to store the Key ID for the KEK used to encrypt this particular secret.

Parameters

- **cypher_text** Byte data resulting from the encryption of the secret data.
- **kek_meta_extended** Optional String object to be persisted alongside the cyphertext.

Plugin Base Class

Barbican cryptographic plugins should implement the abstract base class `CryptoPluginBase`. Concrete implementations of this class should be exposed to barbican using `stevedore` mechanisms explained in the configuration portion of this guide.

```
class barbican.plugin.crypto.base.CryptoPluginBase  
Base class for all Crypto plugins.
```

Barbican requests operations by invoking the methods on an instance of the implementing class. Barbicans plugin manager handles the life-cycle of the Data Transfer Objects (DTOs) that are passed into these methods, and persist the data that is assigned to these DTOs by the plugin.

```
abstract bind_kek_metadata(kek_meta_dto)
```

Key Encryption Key Metadata binding function

Bind a key encryption key (KEK) metadata to the sub-system handling encryption/decryption, updating information about the key encryption key (KEK) metadata in the supplied `kek_metadata` data-transfer-object instance, and then returning this instance.

This method is invoked prior to the `encrypt()` method above. Implementors should fill out the supplied `kek_meta_dto` instance (an instance of `KEKMetadata` above) as needed to completely describe the kek metadata and to complete the binding process. Barbican will persist the contents of this instance once this method returns.

Parameters `kek_meta_dto` Key encryption key metadata to bind, with the `kek_label` attribute guaranteed to be unique, and the `plugin_name` attribute already configured.

Returns `kek_meta_dto`: Returns the specified DTO, after modifications.

abstract decrypt (*decrypt_dto, kek_meta_dto, kek_meta_extended, project_id*)
Decrypt `encrypted_datum` in the context of the provided project.

Parameters

- **decrypt_dto** data transfer object containing the cyphertext to be decrypted.
- **kek_meta_dto** Key encryption key metadata to use for decryption
- **kek_meta_extended** Optional per-secret KEK metadata to use for decryption.
- **project_id** Project ID associated with the encrypted datum.

Returns `str` unencrypted byte data

abstract encrypt (*encrypt_dto, kek_meta_dto, project_id*)
Encryption handler function

This method will be called by Barbican when requesting an encryption operation on a secret on behalf of a project.

Parameters

- **encrypt_dto** (*EncryptDTO*) *EncryptDTO* instance containing the raw secret byte data to be encrypted.
- **kek_meta_dto** (*KEKMetaDTO*) *KEKMetaDTO* instance containing information about the projects Key Encryption Key (KEK) to be used for encryption. Plugins may assume that binding via *bind_kek_metadata()* has already taken place before this instance is passed in.
- **project_id** Project ID associated with the unencrypted data.

Returns A response DTO containing the cyphertext and KEK information.

Return type *ResponseDTO*

abstract generate_asymmetric (*generate_dto, kek_meta_dto, project_id*)
Create a new asymmetric key.

Parameters

- **generate_dto** data transfer object for the record associated with this generation request. Some relevant parameters can be extracted from this object, including `bit_length`, `algorithm` and `passphrase`
- **kek_meta_dto** Key encryption key metadata to use for decryption
- **project_id** Project ID associated with the data.

Returns A tuple containing objects for `private_key`, `public_key` and optionally one for `passphrase`. The objects will be of type `ResponseDTO`. Each object containing encrypted data and `kek_meta_extended`, the former the resultant cypher text, the latter being optional per-secret metadata needed to decrypt (over and above the per-project metadata managed outside of the plugins)

abstract generate_symmetric (*generate_dto, kek_meta_dto, project_id*)

Generate a new key.

Parameters

- **generate_dto** data transfer object for the record associated with this generation request. Some relevant parameters can be extracted from this object, including `bit_length`, `algorithm` and `mode`
- **kek_meta_dto** Key encryption key metadata to use for decryption
- **project_id** Project ID associated with the data.

Returns An object of type `ResponseDTO` containing encrypted data and `kek_meta_extended`, the former the resultant cypher text, the latter being optional per-secret metadata needed to decrypt (over and above the per-project metadata managed outside of the plugins)

abstract get_plugin_name ()

Gets user friendly plugin name.

This plugin name is expected to be read from config file. There will be a default defined for plugin name which can be customized in specific deployment if needed.

This name needs to be unique across a deployment.

abstract supports (*type_enum, algorithm=None, bit_length=None, mode=None*)

Used to determine if the plugin supports the requested operation.

Parameters

- **type_enum** Enumeration from `PluginSupportsType` class
- **algorithm** String algorithm name if needed

Barbican Core Plugin Sequence

Barbican invokes a different sequence of methods on the `CryptoPluginBase` plugin depending on the requested action. Note that these actions are invoked via the secret store adapter class `StoreCryptoAdapterPlugin` which is further described in *The Cryptographic Plugin Adapter*.

For secret storage actions, Barbican core calls the following methods:

1. `supports()` - Asks the plugin if it can support the `barbican.plugin.crypto.base.PluginSupportTypes.ENCRYPT_DECRYPT` operation type.
2. `bind_kek_metadata()` - Allows a plugin to bind an internal key encryption key (KEK) to a project-ID, typically as a label or reference to the actual KEK stored within the cryptographic appliance. This KEK information is stored into Barbicans data store on behalf of the plugin, and then provided back to the plugin for subsequent calls.
3. `encrypt()` - Asks the plugin to perform encryption of an unencrypted secret payload, utilizing the KEK bound to the project-ID above. Barbican core will then persist the encrypted data

returned from this method for later retrieval. The name of the plugin used to perform this encryption is also persisted into Barbican core, to ensure we decrypt this secret only with this plugin.

For secret decryptions and retrievals, Barbican core will select the same plugin as was used to store the secret, and then invoke its `decrypt()` method, providing it both the previously-persisted encrypted secret data as well as the project-ID KEK used to encrypt the secret.

For symmetric key generation, Barbican core calls the following methods:

1. `supports()` - Asks the plugin if it can support the `barbican.plugin.crypto.base.PluginSupportTypes.SYMMETRIC_KEY_GENERATION` operation type.
2. `bind_kek_metadata()` - Same comments as for secret storage above.
3. `generate_symmetric()` - Asks the plugin to both generate a symmetric key, and then encrypted it with the project-ID KEK. Barbican core persists this newly generated and encrypted secret similar to secret storage above.

For asymmetric key generation, Barbican core calls the following methods:

1. `supports()` - Asks the plugin if it can support the `barbican.plugin.crypto.base.PluginSupportTypes.ASYMMETRIC_KEY_GENERATION` operation type.
2. `bind_kek_metadata()` - Same comments as for secret storage above.
3. `generate_asymmetric()` - Asks the plugin to generate and encrypt asymmetric public and private key (and optional passphrase) information, which Barbican core will persist as a container of separate encrypted secrets.

2.5.11 Writing and Running Barbican Tests

As a part of every code review that is submitted to the Barbican project there are a number of gating jobs which aid in the prevention of regression issues within Barbican. As a result, a Barbican developer should be familiar with running Barbican tests locally.

For your convenience we provide the ability to run all tests through the `tox` utility. If you are unfamiliar with `tox` please see refer to the [tox documentation](#) for assistance.

Unit Tests

Currently, we provide `tox` environments for Python 2.7 and 3.5. By default all available test environments within the `tox` configuration will execute when calling `tox`. If you want to run them independently, you can do so with the following command:

```
# Executes tests on Python 2.7
tox -e py27
```

Note: If you do not have the appropriate Python versions available, consider setting up PyEnv to install multiple versions of Python. See the documentation regarding [Setting up a Barbican Development Environment](#) for more information.

Note: Individual unit tests can also be run, using the following commands:

```
# runs a single test with the function named
# test_can_create_new_secret_one_step
tox -e py27 -- test_can_create_new_secret_one_step

# runs only tests in the WhenTestingSecretsResource class and
# the WhenTestingCAsResource class
tox -e py27 -- '(WhenTestingSecretsResource|WhenTestingCAsResource)'
```

The function name or class specified must be one located in the *barbican/tests* directory.

Groups of tests can also be run with a regex match after the `--`. For more information on what can be done with `testr`, please see: <http://testrepository.readthedocs.org/en/latest/MANUAL.html>

You can also setup breakpoints in the unit tests. This can be done by adding `import pdb; pdb.set_trace()` to the line of the unit test you want to examine, then running the following command:

```
# Executes tests on Python 2.7
tox -e debug
```

Note: For a list of `pdb` commands, please see: <https://docs.python.org/2/library/pdb.html>

Python 3.5

In order to run the unit tests within the Python 3.5 unit testing environment you need to make sure you have all necessary packages installed.

- On Ubuntu/Debian:

```
sudo apt-get install python3-dev
```

- On Fedora 21/RHEL7/CentOS7:

```
sudo yum install python3-devel
```

- On Fedora 22 and higher:

```
sudo dnf install python3-devel
```

You then specify to run the unit tests within the Python 3.5 environment when invoking `tox`

```
# Executes tests on Python 3.5
tox -e py35
```

Functional Tests

Unlike running unit tests, the functional tests require Barbican and Keystone services to be running in order to execute. For more information on *setting up a Barbican development environment* and using *Keystone with Barbican*, see our accompanying project documentation.

Once you have the appropriate services running and configured you can execute the functional tests through tox.

```
# Execute Barbican Functional Tests
tox -e functional
```

By default, the functional tox job will use `testr` to execute the functional tests as used in the gating job.

Note: In order to run an individual functional test function, you must use the following command:

```
# runs a single test with the function named
# test_secret_create_then_check_content_types
tox -e functional -- test_secret_create_then_check_content_types

# runs only tests in the SecretsTestCase class and
# the OrdersTestCase class
tox -e functional -- '(SecretsTestCase|OrdersTestCase)'
```

The function name or class specified must be one located in the *functionaltests* directory.

Groups of tests can also be run with a regex match after the `--`. For more information on what can be done with `testr`, please see: <http://testrepository.readthedocs.org/en/latest/MANUAL.html>

Remote Debugging

In order to be able to hit break-points on API calls, you must use remote debugging. This can be done by adding `import rpdb; rpdb.set_trace()` to the line of the API call you wish to test. For example, adding the breakpoint in `def on_post` in `barbican.api.controllers.secrets.py` will allow you to hit the breakpoint when a `POST` is done on the secrets URL.

Note: After performing the `POST` the application will freeze. In order to use `rpdb`, you must open up another terminal and run the following:

```
# enter rpdb using telnet
telnet localhost 4444
```

Once in `rpdb`, you can use the same commands as `pdb`, as seen here: <https://docs.python.org/2/library/pdb.html>

2.6 Barbican API Documentation

2.6.1 User Guide

API guide docs are built to: <https://docs.openstack.org/api-guide/key-manager/>

2.6.2 API Reference

Secrets API - Reference

GET /v1/secrets

Lists a projects secrets.

The list of secrets can be filtered by the parameters passed in via the URL.

The actual secret payload data will not be listed here. Clients must instead make a separate call to retrieve the secret payload data for each individual secret.

Parameters

Name	Type	Description
offset	integer	The starting index within the total list of the secrets that you would like to retrieve.
limit	integer	The maximum number of records to return (up to 100). The default limit is 10.
name	string	Selects all secrets with name similar to this value.
alg	string	Selects all secrets with algorithm similar to this value.
mode	string	Selects all secrets with mode similar to this value.
bits	integer	Selects all secrets with bit_length equal to this value.
secret_type	string	Selects all secrets with secret_type equal to this value.
acl_only	boolean	Selects all secrets with an ACL that contains the user. Project scope is ignored.
created	string	Date filter to select all secrets with <i>created</i> matching the specified criteria. See Date Filters below for more detail.
updated	string	Date filter to select all secrets with <i>updated</i> matching the specified criteria. See Date Filters below for more detail.
expiration	string	Date filter to select all secrets with <i>expiration</i> matching the specified criteria. See Date Filters below for more detail.
sort	string	Determines the sorted order of the returned list. See Sorting below for more detail.

Date Filters:

The values for the `created`, `updated`, and `expiration` parameters are comma-separated lists of time stamps in ISO 8601 format. The time stamps can be prefixed with any of these comparison operators: `gt`: (greater-than), `gte`: (greater-than-or-equal), `lt`: (less-than), `lte`: (less-than-or-equal).

For example, to get a list of secrets that will expire in January of 2020:

```
GET /v1/secrets?expiration=gte:2020-01-01T00:00:00,lt:2020-02-01T00:00:00
```

Sorting:

The value of the `sort` parameter is a comma-separated list of sort keys. Supported sort keys include `created`, `expiration`, `mode`, `name`, `secret_type`, `status`, and `updated`.

Each sort key may also include a direction. Supported directions are `:asc` for ascending and `:desc` for descending. The service will use `:asc` for every key that does not include a direction.

For example, to sort the list from most recently created to oldest:

```
GET /v1/secrets?sort=created:desc
```

Request:

```
GET /v1/secrets?offset=1&limit=2&sort=created
Headers:
  Accept: application/json
  X-Auth-Token: {keystone_token}
  (or X-Project-Id: {project id})
```

Response:

```
{
  "next": "http://{barbican_host}:9311/v1/secrets?limit=2&offset=3",
  "previous": "http://{barbican_host}:9311/v1/secrets?limit=2&offset=0",
  "secrets": [
    {
      "algorithm": null,
      "bit_length": null,
      "content_types": {
        "default": "application/octet-stream"
      },
      "created": "2015-04-07T03:37:19.805835",
      "creator_id": "3a7e3d2421384f56a8fb6cf082a8efab",
      "expiration": null,
      "mode": null,
      "name": "opaque octet-stream base64",
      "secret_ref": "http://{barbican_host}:9311/v1/secrets/{uuid}",
      "secret_type": "opaque",
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        "status": "ACTIVE",
        "updated": "2015-04-07T03:37:19.808337"
    },
    {
        "algorithm": null,
        "bit_length": null,
        "content_types": {
            "default": "application/octet-stream"
        },
        "created": "2015-04-07T03:41:02.184159",
        "creator_id": "3a7e3d2421384f56a8fb6cf082a8efab",
        "expiration": null,
        "mode": null,
        "name": "opaque random octet-stream base64",
        "secret_ref": "http://{barbican_host}:9311/v1/secrets/{uuid}",
        "secret_type": "opaque",
        "status": "ACTIVE",
        "updated": "2015-04-07T03:41:02.187823"
    }
],
"total": 5
}

```

Response Attributes

Name	Type	Description
se-crets	list	Contains a list of secrets. The attributes in the secret objects are the same as for an individual secret.
to-tal	in-te-ger	The total number of secrets available to the user.
next	string	A HATEOAS URL to retrieve the next set of secrets based on the offset and limit parameters. This attribute is only available when the total number of secrets is greater than offset and limit parameter combined.
pre-vious	string	A HATEOAS URL to retrieve the previous set of secrets based on the offset and limit parameters. This attribute is only available when the request offset is greater than 0.

HTTP Status Codes

Code	Description
200	Successful Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource

POST /v1/secrets

Creates a Secret entity. If the `payload` attribute is not included in the request, then only the metadata for the secret is created, and a subsequent PUT request is required.

Attributes

Attribute Name	Type	Description	Default
<code>name</code>	string	(optional) The name of the secret set by the user.	None
<code>expiration</code>	string	(optional) This is a UTC timestamp in ISO 8601 format YYYY-MM-DDTHH:MM:SSZ. If set, the secret will not be available after this time.	None
<code>algorithm</code>	string	(optional) Metadata provided by a user or system for informational purposes.	None
<code>bit_length</code>	integer	(optional) Metadata provided by a user or system for informational purposes. Value must be greater than zero.	None
<code>mode</code>	string	(optional) Metadata provided by a user or system for informational purposes.	None
<code>payload</code>	string	(optional) The secrets data to be stored. <code>payload_content_type</code> must also be supplied if payload is included.	None
<code>payload_content_type</code>	string	(optional) (required if payload is included) The media type for the content of the payload. For more information see <i>Secret Types</i>	None
<code>payload_content_encoding</code>	string	(optional) (required if payload is encoded) The encoding used for the payload to be able to include it in the JSON request. Currently only <code>base64</code> is supported.	None
<code>secret_type</code>	string	(optional) Used to indicate the type of secret being stored. For more information see <i>Secret Types</i>	<code>opaque</code>

Request:

```
POST /v1/secrets
Headers:
  Content-Type: application/json
  X-Auth-Token: <token>

Content:
{
  "name": "AES key",
  "expiration": "2015-12-28T19:14:44.180394",
  "algorithm": "aes",
  "bit_length": 256,
  "mode": "cbc",
  "payload": "YmVlcg==",
  "payload_content_type": "application/octet-stream",
  "payload_content_encoding": "base64"
}
```

Response:

```
201 Created
{
  "secret_ref": "https://{barbican_host}/v1/secrets/{secret_uuid}"
}
```

HTTP Status Codes

Code	Description
201	Successfully created a Secret
400	Bad Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
403	Forbidden. The user has been authenticated, but is not authorized to create a secret. This can be based on the users role or the projects quota.
415	Unsupported media-type

GET /v1/secrets/{uuid}

Retrieves a secrets metadata.

Request:

```
GET /v1/secrets/{uuid}
Headers:
  Accept: application/json
  X-Auth-Token: {token}
  (or X-Project-Id: {project_id})
```

Response:

```
200 OK
{
  "status": "ACTIVE",
  "created": "2015-03-23T20:46:51.650515",
  "updated": "2015-03-23T20:46:51.654116",
  "expiration": "2015-12-28T19:14:44.180394",
  "algorithm": "aes",
  "bit_length": 256,
  "mode": "cbc",
  "name": "AES key",
  "secret_ref": "https://{barbican_host}/v1/secrets/{secret_uuid}",
  "secret_type": "opaque",
  "content_types": {
```

(continues on next page)

(continued from previous page)

```
    "default": "application/octet-stream"
  }
}
```

Payload Request:

Warning: DEPRECATION WARNING: Previous releases of the API allowed the payload to be retrieved from this same endpoint by changing the Accept header to be one of the values listed in the `content_types` attribute of the Secret metadata. This was found to be problematic in some situations, so new applications should make use of the `/v1/secrets/{uuid}/payload` endpoint instead.

```
GET /v1/secrets/{uuid}
Headers:
  Accept: application/octet-stream
  X-Auth-Token: <token>
```

Payload Response:

```
200 OK

beer
```

HTTP Status Codes

Code	Description
200	Successful request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Not Found
406	Not Acceptable

PUT /v1/secrets/{uuid}

Add the payload to an existing metadata-only secret, such as one made by sending a POST `/v1/secrets` request that does not include the `payload` attribute.

Note: This action can only be done for a secret that doesnt have a payload.

Headers

Name	Description	Default
Content-Type	Corresponds with the payload_content_type attribute of a normal secret creation request.	text/plain
Content-Encoding	(optional) Corresponds with the payload_content_encoding attribute of a normal secret creation request.	None

Request:

```
PUT /v1/secrets/{uuid}
Headers:
  X-Auth-Token: <token>
  Content-Type: application/octet-stream
  Content-Encoding: base64

Content:
YmxhaA==
```

Response:

```
204 No Content
```

HTTP Status Codes

Code	Description
204	Successful request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Not Found

DELETE /v1/secrets/{uuid}

Delete a secret by uuid

Request:

```
DELETE /v1/secrets/{uuid}
Headers:
  X-Auth-Token: <token>
```

Response:

```
204 No Content
```

HTTP Status Codes

Code	Description
204	Successful request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Not Found

GET /v1/secrets/{uuid}/payload

Retrieve a secrets payload

Accept Header Options:

When making a request for a secrets payload, you must set the accept header to one of the values listed in the `content_types` attribute of a secrets metadata.

Request:

```
GET /v1/secrets/{uuid}/payload
Headers:
  Accept: text/plain
  X-Auth-Token: <token>
```

Response:

```
200 OK

beer
```

HTTP Status Codes

Code	Description
200	Successful request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Not Found
406	Not Acceptable

Secret Types - Reference

Every secret in Barbican has a type. Secret types are used to describe different kinds of secret data that are stored in Barbican. The type for a particular secret is listed in the secrets metadata as the `secret_type` attribute.

The possible secret types are:

- `symmetric` - Used for storing byte arrays such as keys suitable for symmetric encryption.
- `public` - Used for storing the public key of an asymmetric keypair.
- `private` - Used for storing the private key of an asymmetric keypair.
- `passphrase` - Used for storing plain text passphrases.
- `certificate` - Used for storing cryptographic certificates such as X.509 certificates.
- `opaque` - Used for backwards compatibility with previous versions of the API without typed secrets. New applications are encouraged to specify one of the other secret types.

Symmetric

The `symmetric` secret type is used to store byte arrays of sensitive data, such as keys that are used for symmetric encryption. The content-type used with symmetric secrets is `application/octet-stream`. When storing a symmetric secret with a single POST request, the data must be encoded so that it may be included inside the JSON body of the request. In this case, the content encoding of `base64` can be used.

Example 1.1

Create an encryption key for use in AES-256-CBC encryption and store it in Barbican. First, we'll see how this can be done in a single POST request from the command line using `curl`.

```
# Create an encryption_key file with 256 bits of random data
dd bs=32 count=1 if=/dev/urandom of=encryption_key

# Encode the contents of the encryption key using base64 encoding
KEY_BASE64=$(base64 < encryption_key)

# Send a request to store the key in Barbican
curl -vv -H "X-Auth-Token: $TOKEN" -H 'Accept: application/json' \
-H 'Content-Type: application/json' \
```

(continues on next page)

(continued from previous page)

```
-d '{"name": "AES encryption key",
  "secret_type": "symmetric",
  "payload": "'"$KEY_BASE64"'",
  "payload_content_type": "application/octet-stream",
  "payload_content_encoding": "base64",
  "algorithm": "AES",
  "bit_length": 256,
  "mode": "CBC"}' \
http://localhost:9311/v1/secrets | python -m json.tool
```

This should return a reference (URI) for the secret that was created:

```
{
  "secret_ref": "http://localhost:9311/v1/secrets/48d24158-b4b4-45b8-9669-
→d9f0ef793c23"
}
```

We can use this reference to retrieve the secret metadata:

```
curl -vv -H "X-Auth-Token: $TOKEN" -H 'Accept: application/json' \
http://localhost:9311/v1/secrets/48d24158-b4b4-45b8-9669-d9f0ef793c23 |
python -m json.tool
```

The metadata will list the available content-types for the symmetric secret:

```
{
  "algorithm": "AES",
  "bit_length": 256,
  "content_types": {
    "default": "application/octet-stream"
  },
  "created": "2015-04-08T06:24:16.600393",
  "creator_id": "3a7e3d2421384f56a8fb6cf082a8efab",
  "expiration": null,
  "mode": "CBC",
  "name": "AES encryption key",
  "secret_ref": "http://localhost:9311/v1/secrets/48d24158-b4b4-45b8-
→9669-d9f0ef793c23",
  "secret_type": "symmetric",
  "status": "ACTIVE",
  "updated": "2015-04-08T06:24:16.614204"
}
```

The `content_types` attribute describes the content-types that can be used to retrieve the payload. In this example, there is only the default content type of `application/octet-stream`. We can use it to retrieve the payload:

```
# Retrieve the payload and save it to a file
curl -vv -H "X-Auth-Token: $TOKEN" \
-H 'Accept: application/octet-stream' \
-o retrieved_key \
http://localhost:9311/v1/secrets/48d24158-b4b4-45b8-9669-d9f0ef793c23/
→payload
```

The `retrieved_key` file now contains the byte array we started with. Note that barbican returned the byte array in binary format, not base64. This is because the `payload_content_encoding` is only

used when submitting the secret to barbican.

Public

The public secret type is used to store the public key of an asymmetric keypair. For example, a public secret can be used to store the public key of an RSA keypair. Currently, there is only one file format accepted for public secrets: A DER-encoded `SubjectPublicKeyInfo` structure as defined by X.509 RFC 5280 that has been Base64 encoded with a PEM header and footer. This is the type of public key that is generated by the `openssl` tool by default. The content-type used with public secrets is `application/octet-stream`. When storing a public secret with a single POST request, the contents of the file must be encoded since JSON does not accept newline characters. In this case, the contents of the file must be Base64 encoded and the content encoding of `base64` can be used.

Example 2.1

Create an RSA keypair and store the public key in Barbican. For this example, we will be using a metadata-only POST followed by a PUT.

```
# Create the RSA keypair
openssl genrsa -out private.pem 2048

# Extract the public key
openssl rsa -in private.pem -out public.pem -pubout

# Submit a metadata-only POST
curl -vv -H "X-Auth-Token: $TOKEN" \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d '{"name": "RSA Public Key",
    "secret_type": "public",
    "algorithm": "RSA"}' \
http://localhost:9311/v1/secrets | python -m json.tool
```

This should return a reference (URI) for the secret that was created:

```
200 OK

{
  "secret_ref": "http://localhost:9311/v1/secrets/cd20d134-c229-417a-a753-
→86432ad13bad"
}
```

We can use this reference to add the payload with a PUT request:

```
curl -vv -X PUT -H "X-Auth-Token: $TOKEN" \
-H 'Accept: application/json' \
-H 'Content-Type: application/octet-stream' \
--data-binary @public.pem \
http://localhost:9311/v1/secrets/cd20d134-c229-417a-a753-86432ad13bad
```

The server should respond with a 2xx response to indicate that the PUT request was processed successfully:

```
204 - No Content
```

Now we should be able to request the metadata and see the new content-type listed there:

```
curl -vv -H "X-Auth-Token: $TOKEN" \  
-H 'Accept: application/json' \  
http://localhost:9311/v1/secrets/cd20d134-c229-417a-a753-86432ad13bad |  
python -m json.tool
```

```
{  
  "algorithm": "RSA",  
  "bit_length": null,  
  "content_types": {  
    "default": "application/octet-stream"  
  },  
  "created": "2015-04-08T21:45:59.239976",  
  "creator_id": "3a7e3d2421384f56a8fb6cf082a8efab",  
  "expiration": null,  
  "mode": null,  
  "name": "RSA Public Key",  
  "secret_ref": "http://localhost:9311/v1/secrets/cd20d134-c229-417a-  
→a753-86432ad13bad",  
  "secret_type": "public",  
  "status": "ACTIVE",  
  "updated": "2015-04-08T21:52:57.523969"  
}
```

Finally, we can use the default content-type listed in `content_types` to retrieve the public key:

```
curl -vv -H "X-Auth-Token: $TOKEN" \  
-H 'Accept: application/octet-stream' \  
-o retrieved_public.pem \  
http://localhost:9311/v1/secrets/cd20d134-c229-417a-a753-86432ad13bad/  
→payload
```

The `retrieved_public.pem` file now has the same contents as the `public.pem` file we started with.

Example 2.2

Create an RSA keypair and store the public key in Barbican. For this example we will be using a single POST request.

```
# Create the RSA keypair  
openssl genrsa -out private.pem 2048  
  
# Extract the public key  
openssl rsa -in private.pem -out public.pem -pubout  
  
# Base64 encode the contents of the public key  
PUB_BASE64=$(base64 < public.pem)  
  
curl -vv -H "X-Auth-Token: $TOKEN" \  
-H 'Accept: application/json' \  
-H 'Content-Type: application/json' \  

```

(continues on next page)

(continued from previous page)

```
-d '{"name": "RSA Public Key",
  "secret_type": "public",
  "payload": "'"$PUB_BASE64"'",
  "payload_content_type": "application/octet-stream",
  "payload_content_encoding": "base64",
  "algorithm": "RSA"}' \
http://localhost:9311/v1/secrets | python -m json.tool
```

This should return a reference (URI) for the secret that was created.

```
200 OK

{
  "secret_ref": "http://localhost:9311/v1/secrets/d553f0ac-c79d-43b4-b165-
↳32594b612ad4"
}
```

Secret Metadata API - Reference

GET /v1/secrets/{uuid}/metadata

Lists a secrets user-defined metadata.

If a secret does not contain any user metadata, an empty list will be returned.

Request:

```
GET /v1/secrets/{uuid}/metadata
Headers:
  Accept: application/json
  X-Auth-Token: <token>
```

Response:

```
{
  'metadata': {
    'description': 'contains the AES key',
    'geolocation': '12.3456, -98.7654'
  }
}
```

Response Attributes

Name	Type	Description
meta-data	list	Contains a list of the secret metadatas key/value pairs. The provided keys must be lowercase. If not they will be converted to lowercase.

HTTP Status Codes

Code	Description
200	Successful Request
401	Invalid X-Auth-Token or the token doesnt have permissions to access this resource.
403	Forbidden. The user has been authenticated, but is not authorized to retrieve secret metadata. This can be based on the users role.
404	Not Found

PUT /v1/secrets/{uuid}/metadata

Sets the metadata for a secret. Any metadata that was previously set will be deleted and replaced with this metadata.

Parameters

Name	Type	Description
meta-data	list	Contains a list of the secret metadatas key/value pairs. The provided keys must be lowercase. If not they will be converted to lowercase.

Request:

```

PUT /v1/secrets/{uuid}/metadata
Headers:
  Content-Type: application/json
  X-Auth-Token: <token>

Content:
{
  'metadata': {
    'description': 'contains the AES key',
    'geolocation': '12.3456, -98.7654'
  }
}
    
```


Response:

```
201 OK
{
  "metadata_ref": "https://{barbican_host}/v1/secrets/{secret_uuid}/
↳metadata"
}
```

HTTP Status Codes

Code	Description
201	Successfully created/updated Secret Metadata
400	Bad Request
401	Invalid X-Auth-Token or the token doesnt have permissions to access this resource.
403	Forbidden. The user has been authenticated, but is not authorized to create secret metadata. This can be based on the users role.

GET /v1/secrets/{uuid}/metadata/{key}

Retrieves a secrets user-added metadata.

Request:

```
GET /v1/secrets/{uuid}/metadata/{key}
Headers:
  Accept: application/json
  X-Auth-Token: <token>
```

Response:

```
200 OK
{
  "key": "access-limit",
  "value": "0"
}
```

HTTP Status Codes

Code	Description
200	Successful request
401	Invalid X-Auth-Token or the token doesnt have permissions to access this resource
403	Forbidden. The user has been authenticated, but is not authorized to retrieve secret metadata. This can be based on the users role.
404	Not Found

POST /v1/secrets/{uuid}/metadata/

Adds a new key/value pair to the secrets user metadata. The key sent in the request must not already exist in the metadata. The key must also be in lowercase, otherwise it will automatically be changed to lowercase.

Request:

```
POST /v1/secrets/{uuid}/metadata/
Headers:
  X-Auth-Token: <token>
  Content-Type: application/json

Content:
{
  "key": "access-limit",
  "value": "11"
}
```

Response:

```
201 Created
Secret Metadata Location: http://example.com:9311/v1/secrets/{uuid}/
↪metadata/access-limit
{
  "key": "access-limit",
  "value": "11"
}
```

HTTP Status Codes

Code	Description
201	Successful request
400	Bad Request
401	Invalid X-Auth-Token or the token doesnt have permissions to access this resource.
403	Forbidden. The user has been authenticated, but is not authorized to create secret metadata. This can be based on the users role.
409	Conflict. The provided metadata key already exists.

PUT /v1/secrets/{uuid}/metadata/{key}

Updates an existing key/value pair in the secrets user metadata. The key sent in the request must already exist in the metadata. The key must also be in lowercase, otherwise it will automatically be changed to lowercase.

Request:

```
PUT /v1/secrets/{uuid}/metadata/{key}
Headers:
  X-Auth-Token: <token>
  Content-Type: application/json

Content:
{
  "key": "access-limit",
  "value": "11"
}
```

Response:

```
200 OK

{
  "key": "access-limit",
  "value": "11"
}
```

HTTP Status Codes

Code	Description
200	Successful request
400	Bad Request
401	Invalid X-Auth-Token or the token doesnt have permissions to access this resource.
403	Forbidden. The user has been authenticated, but is not authorized to update secret metadata. This can be based on the users role.
404	Not Found

DELETE /v1/secrets/{uuid}/metadata/{key}

Delete secret metadata by key.

Request:

```
DELETE /v1/secrets/{uuid}/metadata/{key}
Headers:
  X-Auth-Token: <token>
```

Response:

```
204 No Content
```

HTTP Status Codes

Code	Description
204	Successful request
401	Invalid X-Auth-Token or the token doesnt have permissions to access this resource.
403	Forbidden. The user has been authenticated, but is not authorized to delete secret metadata. This can be based on the users role.
404	Not Found

Secret Stores API - Reference

Barbican provides API to manage secret stores available in a deployment. APIs are provided for listing available secret stores and to manage project level secret store mapping. There are two types of secret stores. One is global default secret store which is used for all projects. And then project *preferred* secret store which is used to store all *new* secrets created in that project. For an introduction to multiple store backends support, see [Using Multiple Secret Store Plugins](#) . This document will focus on the details of the Barbican `/v1/secret-stores` REST API.

When multiple secret store backends support is not enabled in service configuration, then all of these API will return resource not found (http status code 404) error. Error message text will highlight that the support is not enabled in configuration.

GET /v1/secret-stores

Project administrator can request list of available secret store backends. Response contains list of secret stores which are currently configured in barbican deployment. If multiple store backends support is not enabled, then list will return resource not found (404) error.

Request/Response:

```
Request:

GET /secret-stores
Headers:
  X-Auth-Token: "f9cf2d480ba3485f85bdb9d07a4959f1"
  Accept: application/json

Response:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "secret_stores": [
    {
      "status": "ACTIVE",
      "updated": "2016-08-22T23:46:45.114283",
      "name": "PKCS11 HSM",
      "created": "2016-08-22T23:46:45.114283",
      "secret_store_ref": "http://localhost:9311/v1/secret-stores/
↪4d27b7a7-b82f-491d-88c0-746bd67dad8",
      "global_default": True,
      "crypto_plugin": "p11_crypto",
      "secret_store_plugin": "store_crypto"
    },
    {
      "status": "ACTIVE",
      "updated": "2016-08-22T23:46:45.124554",
      "name": "KMIP HSM",
      "created": "2016-08-22T23:46:45.124554",
      "secret_store_ref": "http://localhost:9311/v1/secret-stores/
↪93869b0f-60eb-4830-adb9-e2f7154a080b",
      "global_default": False,
      "crypto_plugin": None,
      "secret_store_plugin": "kmip_plugin"
    },
    {
      "status": "ACTIVE",
      "updated": "2016-08-22T23:46:45.127866",
      "name": "Software Only Crypto",
      "created": "2016-08-22T23:46:45.127866",
```

(continues on next page)

(continued from previous page)

```

        "secret_store_ref": "http://localhost:9311/v1/secret-stores/
↪0da45858-9420-42fe-a269-011f5f35deaa",
        "global_default": False,
        "crypto_plugin": "simple_crypto",
        "secret_store_plugin": "store_crypto"
    }
}

```

Response Attributes

Name	Type	Description
secret_stores	list	A list of secret store references
name	string	store and crypto plugin name delimited by + (plus) sign.
secret_store_ref	string	URL for referencing a specific secret store

HTTP Status Codes

Code	Description
200	Successful Request
401	Authentication error. Missing or invalid X-Auth-Token.
403	The user was authenticated, but is not authorized to perform this action
404	Not Found. When multiple secret store backends support is not enabled.

GET /v1/secret-stores/{secret_store_id}

A project administrator (user with admin role) can request details of secret store by its ID. Returned response will highlight whether this secret store is currently configured as global default or not.

Request/Response:

```

Request:
GET /secret-stores/93869b0f-60eb-4830-adb9-e2f7154a080b
Headers:
X-Auth-Token: "f9cf2d480ba3485f85bdb9d07a4959f1"
Accept: application/json

Response:
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "ACTIVE",
  "updated": "2016-08-22T23:46:45.124554",
  "name": "KMIP HSM",
  "created": "2016-08-22T23:46:45.124554",

```

(continues on next page)

(continued from previous page)

```

    "secret_store_ref": "http://localhost:9311/v1/secret-stores/93869b0f-
→60eb-4830-adb9-e2f7154a080b",
    "global_default": False,
    "crypto_plugin": None,
    "secret_store_plugin": "kmip_plugin"
}

```

Response Attributes

Name	Type	Description
name	string	store and crypto plugin name delimited by + (plus) sign
global_default	boolean	flag indicating if this secret store is global default or not
status	list	Status of the secret store
updated	time	Date and time secret store was last updated
created	time	Date and time secret store was created
secret_store_ref	string	URL for referencing a specific secret store

HTTP Status Codes

Code	Description
200	Successful Request
401	Authentication error. Missing or invalid X-Auth-Token.
403	The user was authenticated, but is not authorized to perform this action
404	Not Found. When multiple secret store backends support is not enabled or that secret store id does not exist.

GET /v1/secret-stores/preferred

A project administrator (user with admin role) can request a reference to the preferred secret store if assigned previously. When a preferred secret store is set for a project, then new project secrets are stored using that store backend. If multiple secret store support is not enabled, then this resource will return 404 (Not Found) error.

Request/Response:

```

Request:

GET /v1/secret-stores/preferred
Headers:
  X-Auth-Token: "f9cf2d480ba3485f85bdb9d07a4959f1"
  Accept: application/json

Response:

```

(continues on next page)

(continued from previous page)

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "ACTIVE",
  "updated": "2016-08-22T23:46:45.114283",
  "name": "PKCS11 HSM",
  "created": "2016-08-22T23:46:45.114283",
  "secret_store_ref": "http://localhost:9311/v1/secret-stores/4d27b7a7-
↪b82f-491d-88c0-746bd67dad8",
  "global_default": True,
  "crypto_plugin": "p11_crypto",
  "secret_store_plugin": "store_crypto"
}

```

Response Attributes

Name	Type	Description
secret_store_ref	string	A URL that references a specific secret store

HTTP Status Codes

Code	Description
200	Successful Request
401	Authentication error. Missing or invalid X-Auth-Token.
403	The user was authenticated, but is not authorized to perform this action
404	Not found. No preferred secret store has been defined or multiple secret store backends support is not enabled.

POST /v1/secret-stores/{secret_store_id}/preferred

A project administrator can set a secret store backend to be preferred store backend for his/her project. From there on, any new secret stored in that project will use specified plugin backend for storage and reading thereafter. Existing secret storage will not be impacted as each secret captures its plugin backend information when initially stored. If multiple secret store support is not enabled, then this resource will return 404 (Not Found) error.

Request/Response:

```
Request :

POST /v1/secret-stores/7776adb8-e865-413c-8ccc-4f09c3fe0213/preferred
Headers:
  X-Auth-Token: "f9cf2d480ba3485f85bdb9d07a4959f1"

Response:

HTTP/1.1 204 No Content
```

HTTP Status Codes

Code	Description
204	Successful Request
401	Authentication error. Missing or invalid X-Auth-Token.
403	The user was authenticated, but is not authorized to perform this action
404	The requested entity was not found or multiple secret store backends support is not enabled.

DELETE /v1/secret-stores/{secret_store_id}/preferred

A project administrator can remove preferred secret store backend setting. If multiple secret store support is not enabled, then this resource will return 404 (Not Found) error.

Request/Response:

```
Request :

DELETE /v1/secret-stores/7776adb8-e865-413c-8ccc-4f09c3fe0213/preferred
Headers:
  X-Auth-Token: "f9cf2d480ba3485f85bdb9d07a4959f1"

Response:

HTTP/1.1 204 No Content
```

HTTP Status Codes

Code	Description
204	Successful Request
401	Authentication error. Missing or invalid X-Auth-Token.
403	The user was authenticated, but is not authorized to perform this action
404	The requested entity was not found or multiple secret store backends support is not enabled.

GET /v1/secret-stores/global-default

A project or service administrator can request a reference to the secret store that is used as default secret store backend for the deployment.

Request/Response:

```
Request:

GET /v1/secret-stores/global-default
Headers:
  X-Auth-Token: "f9cf2d480ba3485f85bdb9d07a4959f1"
  Accept: application/json

Response:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "ACTIVE",
  "updated": "2016-08-22T23:46:45.114283",
  "name": "PKCS11 HSM",
  "created": "2016-08-22T23:46:45.114283",
  "secret_store_ref": "http://localhost:9311/v1/secret-stores/4d27b7a7-
↪b82f-491d-88c0-746bd67dad8",
  "global_default": True,
  "crypto_plugin": "p11_crypto",
  "secret_store_plugin": "store_crypto"
}
```

Response Attributes

Name	Type	Description
secret_store_ref	string	A URL that references a specific secret store

HTTP Status Codes

Code	Description
200	Successful Request
401	Authentication error. Missing or invalid X-Auth-Token.
403	The user was authenticated, but is not authorized to perform this action
404	Not Found. When multiple secret store backends support is not enabled.

Containers API - Reference

GET /v1/containers

Lists a projects containers.

Returned containers will be ordered by creation date; oldest to newest.

Parameters

Name	Type	Description
offset	integer	The starting index within the total list of the containers that you would like to retrieve.
limit	integer	The maximum number of containers to return (up to 100). The default limit is 10.

Response Attributes

Name	Type	Description
containers	list	Contains a list of dictionaries filled with container data
total	integer	The total number of containers available to the user
next	string	A HATEOAS URL to retrieve the next set of containers based on the offset and limit parameters. This attribute is only available when the total number of containers is greater than offset and limit parameter combined.
previous	string	A HATEOAS URL to retrieve the previous set of containers based on the offset and limit parameters. This attribute is only available when the request offset is greater than 0.

Request:

```
GET /v1/containers
Headers:
  X-Auth-Token: <token>
```

Response:

```
{
  "containers": [
    {
      "consumers": [],
      "container_ref": "https://{barbican_host}/v1/containers/{uuid}
→",
      "created": "2015-03-26T21:10:45.417835",
      "name": "container name",
      "secret_refs": [
        {
          "name": "private_key",
          "secret_ref": "https://{barbican_host}/v1/secrets/
→{uuid}"
        }
      ],
      "status": "ACTIVE",
      "type": "generic",
      "updated": "2015-03-26T21:10:45.417835"
    }
  ],
  "total": 1
}
```

HTTP Status Codes

Code	Description
200	Successful Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource

GET /v1/containers/{uuid}

Retrieves a single container.

Response Attributes

Name	Type	Description
name	string	(optional) Human readable name for the container
type	string	Type of container. Options: generic, rsa, certificate
secret_refs	list	A list of dictionaries containing references to secrets

Request:

```
GET /v1/containers/{uuid}
Headers:
  X-Auth-Token: <token>
```

Response:

```
{
  "type": "generic",
  "status": "ACTIVE",
  "name": "container name",
  "consumers": [],
  "container_ref": "https://{barbican_host}/v1/containers/{uuid}",
  "secret_refs": [
    {
      "name": "private_key",
      "secret_ref": "https://{barbican_host}/v1/secrets/{uuid}"
    }
  ],
  "created": "2015-03-26T21:10:45.417835",
  "updated": "2015-03-26T21:10:45.417835"
}
```

HTTP Status Codes

Code	Description
200	Successful Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Container not found or unavailable

POST /v1/containers

Create a container

There are three different types of containers that can be created: generic, rsa, and certificate.

Generic

This type of container holds any number of references to secrets. Each secret reference is accompanied by a name. Unlike other container types, no specific restrictions are enforced on the contents name attribute.

RSA

This type of container is designed to hold references to only three different secrets. These secrets are enforced by their accompanied names: public_key, private_key, and private_key_passphrase.

Certificate

This type of container is designed to hold a reference to a certificate and optionally `private_key`, `private_key_passphrase`, and intermediates.

Request Attributes

Name	Type	Description
<code>name</code>	string	(optional) Human readable name for identifying your container
<code>type</code>	string	Type of container. Options: <code>generic</code> , <code>rsa</code> , <code>certificate</code>
<code>secret_refs</code>	list	A list of dictionaries containing references to secrets

Request:

```
POST /v1/containers
Headers:
  X-Auth-Token: <token>

Content:
{
  "type": "generic",
  "name": "container name",
  "secret_refs": [
    {
      "name": "private_key",
      "secret_ref": "https://{barbican_host}/v1/secrets/{secret_uuid}"
    }
  ]
}
```

Response:

```
{
  "container_ref": "https://{barbican_host}/v1/containers/{container_
  →uuid}"
}
```

HTTP Status Codes

Code	Description
201	Successful creation of the container
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
403	Forbidden. The user has been authenticated, but is not authorized to create a container. This can be based on the users role or the projects quota.

DELETE /v1/containers/{uuid}

Deletes a container

Request:

```
DELETE /v1/containers/{container_uuid}
Headers:
  X-Auth-Token: <token>
```

Response:

```
204 No Content
```

HTTP Status Codes

Code	Description
204	Successful deletion of a container
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Container not found or unavailable

POST /v1/containers/{container_uuid}/secrets

Add a secret to an existing container. This is only supported on generic containers.

Request Attributes

Name	Type	Description
name	string	(optional) Human readable name for identifying your secret within the container.
secret_ref	uri	(required) Full URI reference to an existing secret.

Request:

```
POST /v1/containers/{container_uuid}/secrets
Headers:
  X-Project-Id: {project_id}

Content:
{
  "name": "private_key",
  "secret_ref": "https://{barbican_host}/v1/secrets/{secret_uuid}"
}
```

Response:

```
{
  "container_ref": "https://{barbican_host}/v1/containers/{container_
  →uuid}"
}
```

Note that the requesting container_uuid is the same as that provided in the response.

HTTP Status Codes

In general, error codes produced by the containers POST call pertain here as well, especially in regards to the secret references that can be provided.

Code	Description
201	Successful update of the container
400	Missing secret_ref
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
403	Forbidden. The user has been authenticated, but is not authorized to add the secret to the specified container. This can be based on the users role or the projects quota.

DELETE /v1/containers/{container_uuid}/secrets

Remove a secret from a container. This is only supported on generic containers.

Request Attributes

Name	Type	Description
name	string	(optional) Human readable name for identifying your secret within the container.
secret_ref	uri	(required) Full URI reference to an existing secret.

Request:

```
DELETE /v1/containers/{container_uuid}/secrets
Headers:
  X-Project-Id: {project_id}

Content:
{
  "name": "private key",
  "secret_ref": "https://{barbican_host}/v1/secrets/{secret_uuid}"
}
```


Response:

```
204 No Content
```

HTTP Status Codes

Code	Description
204	Successful removal of the secret from the container.
400	Missing secret_ref
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
403	Forbidden. The user has been authenticated, but is not authorized to remove the secret from the specified container. This can be based on the users role or the projects quota.
404	Specified secret_ref is not found in the container.

ACL API - Reference

Note: This feature is applicable only when Barbican is used in an authenticated pipeline i.e. integrated with Keystone.

Note: Currently the access control list (ACL) settings defined for a container are not propagated down to associated secrets.

Warning: This ACL documentation is work in progress and may change in near future.

Secret ACL API**GET /v1/secrets/{uuid}/acl**

Retrieve the ACL settings for a given secret.

If no ACL is defined for that secret, then [Default ACL](#) is returned.

Request/Response (With ACL defined):

```
Request:
GET /v1/secrets/{uuid}/acl
Headers:
  X-Auth-Token: {token_id}
Response:
```

(continues on next page)

(continued from previous page)

```
HTTP/1.1 200 OK
{
  "read":{
    "updated":"2015-05-12T20:08:47.644264",
    "created":"2015-05-12T19:23:44.019168",
    "users":[
      {user_id1},
      {user_id2},
      .....
    ],
    "project-access":{project-access-flag}
  }
}
```

Request/Response (With no ACL defined):

```
Request:

GET /v1/secrets/{uuid}/acl
Headers:
  X-Auth-Token: {token_id}

Response:

HTTP/1.1 200 OK
{
  "read":{
    "project-access": true
  }
}
```

HTTP Status Codes

Code	Description
200	Successful request.
401	Missing or Invalid X-Auth-Token. Authentication required.
403	User does not have permission to access this resource.
404	Secret not found for the given UUID.

PUT /v1/secrets/{uuid}/acl

Create new or replaces existing ACL for a given secret.

This call is used to add new ACL for a secret. If the ACL is already set on a secret, this method will replace it with the requested ACL settings. In case of create (first new explicit ACL) or replace existing ACL, 200 is returned in both cases. To delete existing users from an ACL definition, pass empty list [] for *users*.

Returns an ACL reference in success case.

Attributes

The ACL resource detailed in this page allows access to individual secrets to be controlled. This access is configured via operations on those secrets. Currently only the read operation (which includes GET REST actions) is supported.

Attribute Name	Type	Description	Default
read	parent element	ACL data for read operation.	None
users	[string]	(optional) List of user ids. This needs to be a user id as returned by Keystone.	[]
project-access	boolean	(optional) Flag to mark a secret private so that the user who created the secret and <i>users</i> specified in above list can only access the secret. Pass <i>false</i> to mark the secret private.	<i>true</i>

Request/Response (Set or Replace ACL):

```
Request:

PUT /v1/secrets/{uuid}/acl
Headers:
  Content-Type: application/json
  X-Auth-Token: {token_id}

Body:
{
  "read":{
    "users":[
      {user_id1},
      {user_id2},
      .....
    ],
    "project-access":{project-access-flag}
  }
}

Response:
```

(continues on next page)

(continued from previous page)

```
HTTP/1.1 200 OK
{"acl_ref": "https://{barbican_host}/v1/secrets/{uuid}/acl"}
```

HTTP Status Codes

Code	Description
200	Successfully set/replaced secret ACL.
400	Bad Request.
401	Missing or Invalid X-Auth-Token. Authentication required.
403	User does not have permission to access this resource.
404	Secret not found for the given UUID.
415	Unsupported Media Type.

PATCH /v1/secrets/{uuid}/acl

Updates existing ACL for a given secret. This method can be used to apply partial changes on existing ACL settings. Client can update the *users* list and enable or disable *project-access* flag for existing ACL. List of provided users replaces existing users if any. For an existing list of provided users from an ACL definition, pass empty list [] for *users*.

Returns an ACL reference in success case.

Note: PATCH API support will be changing in near future.

Attributes

At-tribute Name	Type	Description	De-fault
read	parent element	ACL data for read operation.	None
users	[string]	(optional) List of user ids. This needs to be a user id as returned by Keystone.	None
project-access	boolean	(optional) Flag to mark a secret private so that the user who created the secret and <i>users</i> specified in above list can only access the secret. Pass <i>false</i> to mark the secret private.	None

Request/Response (Updating project-access flag):

```

PATCH /v1/secrets/{uuid}/acl
Headers:
  Content-Type: application/json
  X-Auth-Token: {token_id}

Body:
{
  "read":
  {
    "project-access": false
  }
}

Response:
HTTP/1.1 200 OK
{"acl_ref": "https://{barbican_host}/v1/secrets/{uuid}/acl"}

```

Request/Response (Removing all users from ACL):

```

PATCH /v1/secrets/{uuid}/acl
Headers:
  Content-Type: application/json
  X-Auth-Token: {token_id}

Body:
{
  "read":
  {
    "users": []
  }
}

Response:
HTTP/1.1 200 OK
{"acl_ref": "https://{barbican_host}/v1/secrets/{uuid}/acl"}

```

HTTP Status Codes

Code	Description
200	Successfully updated secret ACL.
400	Bad Request.
401	Missing or Invalid X-Auth-Token. Authentication required.
403	User does not have permission to access this resource.
404	Secret not found for the given UUID.
415	Unsupported Media Type.

DELETE /v1/secrets/{uuid}/acl

Delete ACL for a given secret. No content is returned in the case of successful deletion.

Request/Response:

```
DELETE /v1/secrets/{uuid}/acl
Headers:
  X-Auth-Token: {token_id}

Response:
HTTP/1.1 200 OK
```

HTTP Status Codes

Code	Description
200	Successfully deleted secret ACL.
401	Missing or Invalid X-Auth-Token. Authentication required.
403	User does not have permission to access this resource.
404	Secret not found for the given UUID.

Container ACL API

GET /v1/containers/{uuid}/acl

Retrieve the ACL settings for a given container.

If no ACL is defined for that container, then [Default ACL](#) is returned.

Request/Response (With ACL defined):

```
Request:

GET /v1/containers/{uuid}/acl
Headers:
  X-Auth-Token: {token_id}

Response:
HTTP/1.1 200 OK
{
  "read": {
    "updated": "2015-05-12T20:08:47.644264",
    "created": "2015-05-12T19:23:44.019168",
    "users": [
      {user_id1},
      {user_id2},
```

(continues on next page)

(continued from previous page)

```

    .....,
    ],
    "project-access": {project-access-flag}
  }
}

```

Request/Response (With no ACL defined):

```

Request:

GET /v1/containers/{uuid}/acl
Headers:
  X-Auth-Token: {token_id}

Response:

HTTP/1.1 200 OK
{
  "read": {
    "project-access": true
  }
}

```

HTTP Status Codes

Code	Description
200	Successful request.
401	Missing or Invalid X-Auth-Token. Authentication required.
403	User does not have permission to access this resource.
404	Container not found for the given UUID.

PUT /v1/containers/{uuid}/acl

Create new or replaces existing ACL for a given container.

This call is used to add new ACL for an container. If the ACL is already set on a container, this method will replace it with the requested ACL settings. In case of create (first new explicit ACL) or replace existing ACL, 200 is returned in both cases. To delete existing users from an ACL definition, pass empty list [] for *users*.

Returns an ACL reference in success case.

Attributes

The ACL resource detailed in this page allows access to individual containers to be controlled. This access is configured via operations on those containers. Currently only the read operation (which includes GET REST actions) is supported.

Attribute Name	Type	Description	Default
read	parent element	ACL data for read operation.	None
users	[string]	(optional) List of user ids. This needs to be a user id as returned by Keystone.	[]
project-access	boolean	(optional) Flag to mark a container private so that the user who created the container and users specified in above list can only access the container. Pass <i>false</i> to mark the container private.	<i>true</i>

Request/Response (Set or Replace ACL):

```

PUT /v1/containers/{uuid}/acl
Headers:
  Content-Type: application/json
  X-Auth-Token: {token_id}

Body:
{
  "read":{
    "users":[
      {user_id1},
      {user_id2},
      .....
    ],
    "project-access":{project-access-flag}
  }
}

Response:
HTTP/1.1 200 OK
{"acl_ref": "https://{barbican_host}/v1/containers/{uuid}/acl"}

```


HTTP Status Codes

Code	Description
200	Successfully set/replaced container ACL.
400	Bad Request.
401	Missing or Invalid X-Auth-Token. Authentication required.
403	User does not have permission to access this resource.
404	Container not found for the given UUID.
415	Unsupported Media Type.

PATCH /v1/containers/{uuid}/acl

Update existing ACL for a given container. This method can be used to apply partial changes on existing ACL settings. Client can update *users* list and enable or disable *project-access* flag for existing ACL. List of provided users replaces existing users if any. For an existing list of provided users from an ACL definition, pass empty list [] for *users*.

Returns an ACL reference in success case.

Note: PATCH API support will be changing in near future.

Attributes

Attribute Name	Type	Description	Default
read	parent element	ACL data for read operation.	None
users	[string]	(optional) List of user ids. This needs to be a user id as returned by Keystone.	None
project-access	boolean	(optional) Flag to mark a container private so that the user who created the container and <i>users</i> specified in above list can only access the container. Pass <i>false</i> to mark the container private.	None

Request/Response (Updating project-access flag):

```
PATCH /v1/containers/{uuid}/acl
Headers:
  Content-Type: application/json
  X-Auth-Token: {token_id}

Body:
{
  "read":
```

(continues on next page)

(continued from previous page)

```
{
  "project-access": false
}

Response:
HTTP/1.1 200 OK
{"acl_ref": "https://{barbican_host}/v1/containers/{uuid}/acl"}
```

Request/Response (Removing all users from ACL):

```
PATCH /v1/containers/{uuid}/acl
Headers:
  Content-Type: application/json
  X-Auth-Token: {token_id}

Body:
{
  "read":
  {
    "users": []
  }
}

Response:
HTTP/1.1 200 OK
{"acl_ref": "https://{barbican_host}/v1/containers/{uuid}/acl"}
```

HTTP Status Codes

Code	Description
200	Successfully updated container ACL.
400	Bad Request.
401	Missing or Invalid X-Auth-Token. Authentication required.
403	User does not have permission to access this resource.
404	Container not found for the given UUID.
415	Unsupported Media Type.

DELETE /v1/containers/{uuid}/acl

Delete ACL for a given container. No content is returned in the case of successful deletion.

Request/Response:

```
DELETE /v1/containers/{uuid}/acl
Headers:
  X-Auth-Token: {token_id}

Response:
HTTP/1.1 200 OK
```

HTTP Status Codes

Code	Description
200	Successfully deleted container ACL.
401	Missing or Invalid X-Auth-Token. Authentication required.
403	User does not have permission to access this resource.
404	Container not found for the given UUID.

Quotas API - Reference**GET /v1/quotas**

Get the effective quotas for the project of the requester. The project id of the requester is derived from the authentication token provided in the X-Auth-Token header.

Request/Response:

```
Request:

GET /v1/quotas
Headers:
  X-Auth-Token: <token>
  Accept: application/json

Response:

HTTP/1.1 200 OK
Content-Type: application/json

{
  "quotas": {
    "secrets": 10,
    "orders": 20,
    "containers": 10,
    "consumers": -1,
    "cas": 5
  }
}
```

Response Attributes

Name	Type	Description
quotas	dict	Contains a dictionary with quota information
secrets	integer	Contains the effective quota value of the current project for the secret resource.
orders	integer	Contains the effective quota value of the current project for the orders resource.
containers	integer	Contains the effective quota value of the current project for the containers resource.
consumers	integer	Contains the effective quota value of the current project for the consumers resource.
cas	integer	Contains the effective quota value of the current project for the CAs resource.

Effective quota values are interpreted as follows:

Value	Description
-1	A negative value indicates the resource is unconstrained by a quota.
0	A zero value indicates that the resource is disabled.
int	A positive value indicates the maximum number of that resource that can be created for the current project.

HTTP Status Codes

Code	Description
200	Successful Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource

GET /v1/project-quotas

Gets a list of configured project quota records. Paging is supported using the optional parameters offset and limit.

Request/Response:

<pre>Request: GET /v1/project-quotas Headers: X-Auth-Token:<token> Accept: application/json Response:</pre>

(continues on next page)

(continued from previous page)

```
200 OK
Content-Type: application/json

{
  "project_quotas": [
    {
      "project_id": "1234",
      "project_quotas": {
        "secrets": 2000,
        "orders": 0,
        "containers": -1,
        "consumers": null,
        "cas": null
      }
    },
    {
      "project_id": "5678",
      "project_quotas": {
        "secrets": 200,
        "orders": 100,
        "containers": -1,
        "consumers": null,
        "cas": null
      }
    }
  ],
  "total" : 30,
}
```

Parameters

Name	Type	Description
offset	integer	The starting index within the total list of the project quotas that you would like to receive.
limit	integer	The maximum number of records to return.

Response Attributes

Name	Type	Description
project-id	string	The UUID of a project with configured quota information.
project-quotas	dict	Contains a dictionary with project quota information.
secrets	integer	Contains the effective quota value of the current project for the secret resource.
orders	integer	Contains the effective quota value of the current project for the orders resource.
containers	integer	Contains the effective quota value of the current project for the containers resource.
consumers	integer	Contains the effective quota value of the current project for the consumers resource.
cas	integer	Contains the effective quota value of the current project for the CAs resource.
total	integer	The total number of configured project quotas records.
next	string	A HATEOAS URL to retrieve the next set of quotas based on the offset and limit parameters. This attribute is only available when the total number of secrets is greater than offset and limit parameter combined.
previous	string	A HATEOAS URL to retrieve the previous set of quotas based on the offset and limit parameters. This attribute is only available when the request offset is greater than 0.

Configured project quota values are interpreted as follows:

Value	Description
-1	A negative value indicates the resource is unconstrained by a quota.
0	A zero value indicates that the resource is disabled.
int	A positive value indicates the maximum number of that resource that can be created for the current project.
null	A null value indicates that the default quota value for the resource will be used as the quota for this resource in the current project.

HTTP Status Codes

Code	Description
200	Successful Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource

GET /v1/project-quotas/{uuid}

Retrieves a projects configured project quota information.

Request/Response:

Request:

```
GET /v1/project-quotas/{uuid}
Headers:
  X-Auth-Token: <token>
  Accept: application/json
```

Response:

```
200 OK

Content-Type: application/json

{
  "project_quotas": {
    "secrets": 10,
    "orders": 20,
    "containers": -1,
    "consumers": 10,
    "cas": 5
  }
}
```

Response Attributes

Name	Type	Description
project-quotas	dict	Contains a dictionary with project quota information.
secrets	integer	Contains the configured quota value of the requested project for the secret resource.
orders	integer	Contains the configured quota value of the requested project for the orders resource.
containers	integer	Contains the configured quota value of the requested project for the containers resource.
consumers	integer	Contains the configured quota value of the requested project for the consumers resource.
cas	integer	Contains the configured quota value of the requested project for the CAs resource.

HTTP Status Codes

Code	Description
200	Successful request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Not Found. The requested project does not have any configured quotas.

PUT /v1/project-quotas/{uuid}

Create or update the configured project quotas for the project with the specified UUID.

Request/Response:

Request:

```

PUT /v1/project-quotas/{uuid}
Headers:
  X-Auth-Token:<token>
  Content-Type: application/json

Body::

{
  "project_quotas": {
    "secrets": 50,
    "orders": 10,
    "containers": 20
  }
}
    
```

(continues on next page)

(continued from previous page)

Response :

204 OK

Request Attributes

Attribute Name	Type	Description
project-quotas	dict	A dictionary with project quota information.
secrets	integer	The value to set for this projects secret quota.
orders	integer	The value to set for this projects order quota.
containers	integer	The value to set for this projects container quota.
consumers	integer	The value to set for this projects consumer quota.
cas	integer	The value to set for this projects CA quota.

Configured project quota values are specified as follows:

Value	Description
-1	A negative value indicates the resource is unconstrained by a quota.
0	A zero value indicates that the resource is disabled.
int	A positive value indicates the maximum number of that resource that can be created for the specified project.
	If a value is not given for a resource, this indicates that the default quota should be used for that resource for the specified project.

HTTP Status Codes

Code	Description
204	Successful request
400	Bad Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource

DELETE /v1/project-quotas/{uuid}

Delete the project quotas configuration for the project with the requested UUID. When the project quota configuration is deleted, then the default quotas will be used for the specified project.

Request/Response:

Request :

```
DELETE v1/project-quotas/{uuid}
Headers:
  X-Auth-Token: <token>
```

Response:

```
204 No Content
```

HTTP Status Codes

Code	Description
204	Successful request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Not Found

Consumers API - Reference

GET {container_ref}/consumers

Lists a containers consumers.

The list of consumers can be filtered by the parameters passed in via the URL.

Parameters

Name	Type	Description
offset	integer	The starting index within the total list of the consumers that you would like to retrieve.
limit	integer	The maximum number of records to return (up to 100). The default limit is 10.

Request:

```
GET {container_ref}/consumers
Headers:
  X-Auth-Token: <token>
```

Response:

```

200 OK
{
  "total": 3,
  "consumers": [
    {
      "status": "ACTIVE",
      "URL": "consumerurl",
      "updated": "2015-10-15T21:06:33.123878",
      "name": "consumername",
      "created": "2015-10-15T21:06:33.123872"
    },
    {
      "status": "ACTIVE",
      "URL": "consumerURL2",
      "updated": "2015-10-15T21:17:08.092416",
      "name": "consumername2",
      "created": "2015-10-15T21:17:08.092408"
    },
    {
      "status": "ACTIVE",
      "URL": "consumerURL3",
      "updated": "2015-10-15T21:21:29.970370",
      "name": "consumername3",
      "created": "2015-10-15T21:21:29.970365"
    }
  ]
}

```

Request:

```

GET {container_ref}/consumers?limit=1\&offset=1
Headers:
  X-Auth-Token: <token>

```

```

{
  "total": 3,
  "next": "http://localhost:9311/v1/containers/{container_ref}/consumers?
↪limit=1&offset=2",
  "consumers": [
    {
      "status": "ACTIVE",
      "URL": "consumerURL2",
      "updated": "2015-10-15T21:17:08.092416",
      "name": "consumername2",
      "created": "2015-10-15T21:17:08.092408"
    }
  ],
  "previous": "http://localhost:9311/v1/containers/{container_ref}/
↪consumers?limit=1&offset=0"
}

```

Response Attributes

Name	Type	Description
consumers	list	Contains a list of dictionaries filled with consumer metadata.
total	integer	The total number of consumers available to the user.
next	string	A HATEOAS URL to retrieve the next set of consumers based on the offset and limit parameters. This attribute is only available when the total number of consumers is greater than offset and limit parameter combined.
previous	string	A HATEOAS URL to retrieve the previous set of consumers based on the offset and limit parameters. This attribute is only available when the request offset is greater than 0.

HTTP Status Codes

Code	Description
200	OK.
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource.
403	Forbidden. The user has been authenticated, but is not authorized to delete a consumer. This can be based on the users role.

POST {container_ref}/consumers

Creates a consumer

Attributes

Attribute Name	Type	Description	Default
name	string	The name of the consumer set by the user.	None
url	string	The URL for the user or service using the container.	None

Request:

```
POST {container_ref}/consumers
Headers:
  X-Auth-Token: <token>
  Content-Type: application/json

Content:
{
  "name": "ConsumerName",
  "url": "ConsumerURL"
}
```

Response:

```

200 OK

{
  "status": "ACTIVE",
  "updated": "2015-10-15T17:56:18.626724",
  "name": "container name",
  "consumers": [
    {
      "URL": "consumerURL",
      "name": "consumername"
    }
  ],
  "created": "2015-10-15T17:55:44.380002",
  "container_ref": "http://localhost:9311/v1/containers/74bbd3fd-9ba8-
↪42ee-b87e-2eecf10e47b9",
  "creator_id": "b17c815d80f946ea8505c34347a2aeba",
  "secret_refs": [
    {
      "secret_ref": "http://localhost:9311/v1/secrets/b61613fc-be53-
↪4696-ac01-c3a789e87973",
      "name": "private_key"
    }
  ],
  "type": "generic"
}

```

HTTP Status Codes

Code	Description
200	OK.
400	Bad Request.
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource.
403	Forbidden. The user has been authenticated, but is not authorized to create a consumer. This can be based on the users role or the projects quota.

DELETE {container_ref}/consumers

Delete a consumer.

Attributes

Attribute Name	Type	Description	Default
name	string	The name of the consumer set by the user.	None
URL	string	The URL for the user or service using the container.	None

Request:

```
DELETE {container_ref}/consumers
Headers:
  X-Auth-Token: <token>
  Content-Type: application/json

Content:
{
  "name": "ConsumerName",
  "URL": "ConsumerURL"
}
```

Response:

```
200 OK

{
  "status": "ACTIVE",
  "updated": "2015-10-15T17:56:18.626724",
  "name": "container name",
  "consumers": [],
  "created": "2015-10-15T17:55:44.380002",
  "container_ref": "http://localhost:9311/v1/containers/74bbd3fd-9ba8-
→42ee-b87e-2eecf10e47b9",
  "creator_id": "b17c815d80f946ea8505c34347a2aeba",
  "secret_refs": [
    {
      "secret_ref": "http://localhost:9311/v1/secrets/b61613fc-be53-
→4696-ac01-c3a789e87973",
      "name": "private_key"
    }
  ],
  "type": "generic"
}
```

HTTP Status Codes

Code	Description
200	OK.
400	Bad Request.
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource.
403	Forbidden. The user has been authenticated, but is not authorized to delete a consumer. This can be based on the users role.
404	Consumer Not Found.

Orders API - Reference

Warning: DEPRECATION WARNING: The Certificates Order resource has been deprecated and will be removed in the P release.

GET /v1/orders

Lists a projects orders.

The list of orders can be filtered by the parameters passed in via the URL.

Parameters

Name	Type	Description
offset	integer	The starting index within the total list of the orders that you would like to retrieve. (Default is 0)
limit	integer	The maximum number of records to return (up to 100). (Default is 10)

Request:

```
GET /v1/orders
Headers:
  Content-Type: application/json
  X-Auth-Token: {token}
```

Response:

```
200 Success

{
  "orders": [
    {
      "created": "2015-10-20T18:38:44",
      "creator_id": "40540f978fbd45c1af18910e3e02b63f",
      "meta": {
        "algorithm": "AES",
        "bit_length": 256,
        "expiration": null,
        "mode": "cbc",
        "name": "secretname",
        "payload_content_type": "application/octet-stream"
      },
      "order_ref": "http://localhost:9311/v1/orders/2284ba6f-f964-4de7-
↪b61e-c413df5d1e47",
      "secret_ref": "http://localhost:9311/v1/secrets/15dcf8e4-3138-4360-
↪be9f-fc4bc2e64a19",
      "status": "ACTIVE",
      "sub_status": "Unknown",
      "sub_status_message": "Unknown",
      "type": "key",
      "updated": "2015-10-20T18:38:44"
    },
    {
      "created": "2015-10-20T18:38:47",
      "creator_id": "40540f978fbd45c1af18910e3e02b63f",
      "meta": {
        "algorithm": "AES",
        "bit_length": 256,
        "expiration": null,
        "mode": "cbc",
        "name": "secretname",
        "payload_content_type": "application/octet-stream"
      },
      "order_ref": "http://localhost:9311/v1/orders/87b7169e-3aa2-4cb1-
↪8800-b5aadf6babd1",
      "secret_ref": "http://localhost:9311/v1/secrets/80183f4b-c0de-4a94-
↪91ad-6d55251acee2",
      "status": "ACTIVE",
      "sub_status": "Unknown",
      "sub_status_message": "Unknown",
      "type": "key",
      "updated": "2015-10-20T18:38:47"
    }
  ],
  "total": 2
}
```


Response Attributes

Name	Type	Description
orders	list	Contains a list of dictionaries filled with order metadata.
total	integer	The total number of orders available to the user.
next	string	A HATEOS URL to retrieve the next set of objects based on the offset and limit parameters. This attribute is only available when the total number of objects is greater than offset and limit parameter combined.
previous	string	A HATEOS URL to retrieve the previous set of objects based on the offset and limit parameters. This attribute is only available when the request offset is greater than 0.

HTTP Status Codes

Code	Description
200	Successful Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource

POST /v1/orders

Creates an order

Parameters

Attribute Name	Type	Description	Default
type	string	The type of key to be generated. Valid types are key, asymmetric, and certificate	None
meta	dict	Dictionary containing the secret metadata used to generate the secret.	None

Request:

```
POST /v1/orders
Headers:
  Content-Type: application/json
  X-Auth-Token: {token}

Content:
{
  "type": "key",
```

(continues on next page)

(continued from previous page)

```
"meta":
  {
    "name": "secretname",
    "algorithm": "AES",
    "bit_length": 256,
    "mode": "cbc",
    "payload_content_type": "application/octet-stream"
  }
}
```

Response:

```
202 Created

{
  "order_ref": "http://{barbican_host}/v1/orders/{order_uuid}"
}
```

Response Attributes

Name	Type	Description
order_ref	string	Order reference

HTTP Status Codes

Code	Description
202	Successfully created an order
400	Bad Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
415	Unsupported media-type

GET /v1/orders/{uuid}

Retrieves an orders metadata

Request:

```
GET /v1/orders/{order_uuid}
Headers:
  Accept: application/json
  X-Auth-Token: {token}
```

Parameters

None

Response:

```
200 Success

{
  "created": "2015-10-20T18:49:02",
  "creator_id": "40540f978fbd45c1af18910e3e02b63f",
  "meta": {
    "algorithm": "AES",
    "bit_length": 256,
    "expiration": null,
    "mode": "cbc",
    "name": "secretname",
    "payload_content_type": "application/octet-stream"
  },
  "order_ref": "http://localhost:9311/v1/orders/5443d349-fe0c-4bfd-bd9d-
→99c4a9770638",
  "secret_ref": "http://localhost:9311/v1/secrets/16f8d4f3-d3dd-4160-
→a5bd-8e5095a42613",
  "status": "ACTIVE",
  "sub_status": "Unknown",
  "sub_status_message": "Unknown",
  "type": "key",
  "updated": "2015-10-20T18:49:02"
}
```

Response Attributes

Name	Type	Description
created	string	Timestamp in ISO8601 format of when the order was created
creator_id	string	Keystone Id of the user who created the order
meta	dict	Secret metadata used for informational purposes
order_ref	string	Order href associated with the order
secret_ref	string	Secret href associated with the order
status	string	Current status of the order
sub_status	string	Metadata associated with the order
sub_status_message	string	Metadata associated with the order
type	string	Indicates the type of order
updated	string	Timestamp in ISO8601 format of the last time the order was updated.

HTTP Status Codes

Code	Description
200	Successfully retrieved the order
400	Bad Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Not Found

DELETE /v1/orders/{uuid}

Delete an order

Request:

```
DELETE /v1/orders/{order_uuid}
Headers:
  X-Auth-Token: {token}
```

Parameters

None

Response:

204 Success

HTTP Status Codes

Code	Description
204	Successfully deleted the order
400	Bad Request
401	Invalid X-Auth-Token or the token doesnt have permissions to this resource
404	Not Found

SAMPLE FILES

3.1 Barbican Sample Configuration File

Use the `barbican.conf` file to configure most Key Manager service options:

3.2 Barbican Sample Policy

The following is a sample Barbican policy file that has been auto-generated from default policy values in code. If you're using the default policies, then the maintenance of this file is not necessary, and it should not be copied into a deployment. Doing so will result in duplicate policy definitions. It is here to help explain which policy operations protect specific Barbican APIs, but it is not suggested to copy and paste into a deployment unless you're planning on providing a different policy for an operation that is not the default.

The sample policy file can also be viewed in [file form](#).

```
#"admin": "role:admin"

#"observer": "role:observer"

#"creator": "role:creator"

#"audit": "role:audit"

#"service_admin": "role:key-manager:service-admin"

#"admin_or_creator": "rule:admin or rule:creator"

#"all_but_audit": "rule:admin or rule:observer or rule:creator"

#"all_users": "rule:admin or rule:observer or rule:creator or rule:audit_
↳or rule:service_admin"

#"secret_project_match": "project_id:%(target.secret.project_id)s"

#"secret_acl_read": "'read':%(target.secret.read)s"

#"secret_private_read": "'False':%(target.secret.read_project_access)s"

#"secret_creator_user": "user_id:%(target.secret.creator_id)s"
```

(continues on next page)

(continued from previous page)

```

#"container_project_match": "project_id:%(target.container.project_id)s"

#"container_acl_read": "'read':%(target.container.read)s"

#"container_private_read": "'False':%(target.container.read_project_
↪access)s"

#"container_creator_user": "user_id:%(target.container.creator_id)s"

#"secret_non_private_read": "rule:all_users and rule:secret_project_match_
↪and not rule:secret_private_read"

#"secret_decrypt_non_private_read": "rule:all_but_audit and rule:secret_
↪project_match and not rule:secret_private_read"

#"container_non_private_read": "rule:all_users and rule:container_project_
↪match and not rule:container_private_read"

#"secret_project_admin": "rule:admin and rule:secret_project_match"

#"secret_project_creator": "rule:creator and rule:secret_project_match and_
↪rule:secret_creator_user"

#"secret_project_creator_role": "rule:creator and rule:secret_project_match
↪"

#"container_project_admin": "rule:admin and rule:container_project_match"

#"container_project_creator": "rule:creator and rule:container_project_
↪match and rule:container_creator_user"

#"container_project_creator_role": "rule:creator and rule:container_
↪project_match"

# Retrieve the ACL settings for a given secret.If no ACL is defined
# for that secret, then Default ACL is returned.
# GET /v1/secrets/{secret-id}/acl
# Intended scope(s):
#"secret_acls:get": "rule:all_but_audit and rule:secret_project_match"

# Delete the ACL settings for a given secret.
# DELETE /v1/secrets/{secret-id}/acl
# Intended scope(s):
#"secret_acls:delete": "rule:secret_project_admin or rule:secret_project_
↪creator or (rule:secret_project_creator_role and rule:secret_non_private_
↪read) "

# Create new, replaces, or updates existing ACL for a given secret.
# PUT /v1/secrets/{secret-id}/acl
# PATCH /v1/secrets/{secret-id}/acl
# Intended scope(s):
#"secret_acls:put_patch": "rule:secret_project_admin or rule:secret_
↪project_creator or (rule:secret_project_creator_role and rule:secret_non_
↪private_read) "

# Retrieve the ACL settings for a given container.

```

(continues on next page)

(continued from previous page)

```

# GET /v1/containers/{container-id}/acl
# Intended scope(s):
#"container_acls:get": "rule:all_but_audit and rule:container_project_match
↳"

# Delete ACL for a given container. No content is returned in the case
# of successful deletion.
# DELETE /v1/containers/{container-id}/acl
# Intended scope(s):
#"container_acls:delete": "rule:container_project_admin or rule:container_
↳project_creator or (rule:container_project_creator_role and_
↳rule:container_non_private_read) "

# Create new or replaces existing ACL for a given container.
# PUT /v1/containers/{container-id}/acl
# PATCH /v1/containers/{container-id}/acl
# Intended scope(s):
#"container_acls:put_patch": "rule:container_project_admin or_
↳rule:container_project_creator or (rule:container_project_creator_role_
↳and rule:container_non_private_read) "

# List a specific consumer for a given container.
# GET /v1/containers/{container-id}/consumers/{consumer-id}
# Intended scope(s):
#"consumer:get": "rule:admin or rule:observer or rule:creator or_
↳rule:audit or rule:container_non_private_read or rule:container_project_
↳creator or rule:container_project_admin or rule:container_acl_read"

# List a containers consumers.
# GET /v1/containers/{container-id}/consumers
# Intended scope(s):
#"consumers:get": "rule:admin or rule:observer or rule:creator or_
↳rule:audit or rule:container_non_private_read or rule:container_project_
↳creator or rule:container_project_admin or rule:container_acl_read"

# Creates a consumer.
# POST /v1/containers/{container-id}/consumers
# Intended scope(s):
#"consumers:post": "rule:admin or rule:container_non_private_read or_
↳rule:container_project_creator or rule:container_project_admin or_
↳rule:container_acl_read"

# Deletes a consumer.
# DELETE /v1/containers/{container-id}/consumers/{consumer-id}
# Intended scope(s):
#"consumers:delete": "rule:admin or rule:container_non_private_read or_
↳rule:container_project_creator or rule:container_project_admin or_
↳rule:container_acl_read"

# Creates a container.
# POST /v1/containers
# Intended scope(s):
#"containers:post": "rule:admin_or_creator"

# Lists a projects containers.
# GET /v1/containers

```

(continues on next page)

(continued from previous page)

```
# Intended scope(s):
#"containers:get": "rule:all_but_audit"

# Retrieves a single container.
# GET /v1/containers/{container-id}
# Intended scope(s):
#"container:get": "rule:container_non_private_read or rule:container_
↳project_creator or rule:container_project_admin or rule:container_acl_
↳read"

# Deletes a container.
# DELETE /v1/containers/{uuid}
# Intended scope(s):
#"container:delete": "rule:container_project_admin or rule:container_
↳project_creator"

# Add a secret to an existing container.
# POST /v1/containers/{container-id}/secrets
# Intended scope(s):
#"container_secret:post": "rule:container_project_admin or rule:container_
↳project_creator or rule:container_project_creator_role and_
↳rule:container_non_private_read"

# Remove a secret from a container.
# DELETE /v1/containers/{container-id}/secrets/{secret-id}
# Intended scope(s):
#"container_secret:delete": "rule:container_project_admin or_
↳rule:container_project_creator or rule:container_project_creator_role_
↳and rule:container_non_private_read"

# Gets list of all orders associated with a project.
# GET /v1/orders
# Intended scope(s):
#"orders:get": "rule:all_but_audit"

# Creates an order.
# POST /v1/orders
# Intended scope(s):
#"orders:post": "rule:admin_or_creator"

# Unsupported method for the orders API.
# PUT /v1/orders
# Intended scope(s):
#"orders:put": "rule:admin_or_creator"

# Retrieves an orders metadata.
# GET /v1/orders/{order-id}
# Intended scope(s):
#"order:get": "rule:all_users and project_id:%(target.order.project_id)s"

# Deletes an order.
# DELETE /v1/orders/{order-id}
# Intended scope(s):
#"order:delete": "rule:admin and project_id:%(target.order.project_id)s"

# List quotas for the project the user belongs to.
```

(continues on next page)

(continued from previous page)

```

# GET /v1/quotas
# Intended scope(s):
#"quotas:get": "rule:all_users"

# List quotas for the specified project.
# GET /v1/project-quotas
# GET /v1/project-quotas/{uuid}
# Intended scope(s):
#"project_quotas:get": "rule:service_admin"

# Create or update the configured project quotas for the project with
# the specified UUID.
# PUT /v1/project-quotas/{uuid}
# Intended scope(s):
#"project_quotas:put": "rule:service_admin"

# Delete the project quotas configuration for the project with the
# requested UUID.
# DELETE /v1/quotas}
# Intended scope(s):
#"project_quotas:delete": "rule:service_admin"

# metadata/: Lists a secrets user-defined metadata. || metadata/{key}:
# Retrieves a secrets user-added metadata.
# GET /v1/secrets/{secret-id}/metadata
# GET /v1/secrets/{secret-id}/metadata/{meta-key}
# Intended scope(s):
#"secret_meta:get": "rule:secret_non_private_read or rule:secret_project_
↪creator or rule:secret_project_admin or rule:secret_acl_read"

# Adds a new key/value pair to the secrets user-defined metadata.
# POST /v1/secrets/{secret-id}/metadata/{meta-key}
# Intended scope(s):
#"secret_meta:post": "rule:secret_project_admin or rule:secret_project_
↪creator or (rule:secret_project_creator_role and rule:secret_non_private_
↪read) "

# metadata/: Sets the user-defined metadata for a secret ||
# metadata/{key}: Updates an existing key/value pair in the secrets
# user-defined metadata.
# PUT /v1/secrets/{secret-id}/metadata
# PUT /v1/secrets/{secret-id}/metadata/{meta-key}
# Intended scope(s):
#"secret_meta:put": "rule:secret_project_admin or rule:secret_project_
↪creator or (rule:secret_project_creator_role and rule:secret_non_private_
↪read) "

# Delete secret user-defined metadata by key.
# DELETE /v1/secrets/{secret-id}/metadata/{meta-key}
# Intended scope(s):
#"secret_meta:delete": "rule:secret_project_admin or rule:secret_project_
↪creator or (rule:secret_project_creator_role and rule:secret_non_private_
↪read) "

# Retrieve a secrets payload.
# GET /v1/secrets/{uuid}/payload

```

(continues on next page)

(continued from previous page)

```
# Intended scope(s):
#"secret:decrypt": "rule:secret_decrypt_non_private_read or rule:secret_
↳project_creator or rule:secret_project_admin or rule:secret_acl_read"

# Retrieves a secrets metadata.
# GET /v1/secrets/{secret-id}
# Intended scope(s):
#"secret:get": "rule:secret_non_private_read or rule:secret_project_
↳creator or rule:secret_project_admin or rule:secret_acl_read"

# Add the payload to an existing metadata-only secret.
# PUT /v1/secrets/{secret-id}
# Intended scope(s):
#"secret:put": "rule:admin_or_creator and rule:secret_project_match"

# Delete a secret by uuid.
# DELETE /v1/secrets/{secret-id}
# Intended scope(s):
#"secret:delete": "rule:secret_project_admin or rule:secret_project_
↳creator or (rule:secret_project_creator_role and not rule:secret_private_
↳read) "

# Creates a Secret entity.
# POST /v1/secrets
# Intended scope(s):
#"secrets:post": "rule:admin_or_creator"

# Lists a projects secrets.
# GET /v1/secrets
# Intended scope(s):
#"secrets:get": "rule:all_but_audit"

# Get list of available secret store backends.
# GET /v1/secret-stores
# Intended scope(s):
#"secretstores:get": "rule:admin"

# Get a reference to the secret store that is used as default secret
# store backend for the deployment.
# GET /v1/secret-stores/global-default
# Intended scope(s):
#"secretstores:get_global_default": "rule:admin"

# Get a reference to the preferred secret store if assigned
# previously.
# GET /v1/secret-stores/preferred
# Intended scope(s):
#"secretstores:get_preferred": "rule:admin"

# Set a secret store backend to be preferred store backend for their
# project.
# POST /v1/secret-stores/{ss-id}/preferred
# Intended scope(s):
#"secretstore_preferred:post": "rule:admin"

# Remove preferred secret store backend setting for their project.
```

(continues on next page)

(continued from previous page)

```
# DELETE /v1/secret-stores/{ss-id}/preferred
# Intended scope(s):
#"secretstore_preferred:delete": "rule:admin"

# Get details of secret store by its ID.
# GET /v1/secret-stores/{ss-id}
# Intended scope(s):
#"secretstore:get": "rule:admin"

# Get a specific transport key.
# GET /v1/transport_keys/{key-id}
# Intended scope(s):
#"transport_key:get": "rule:all_users"

# Delete a specific transport key.
# DELETE /v1/transport_keys/{key-id}
# Intended scope(s):
#"transport_key:delete": "rule:admin"

# Get a list of all transport keys.
# GET /v1/transport_keys
# Intended scope(s):
#"transport_keys:get": "rule:all_users"

# Create a new transport key.
# POST /v1/transport_keys
# Intended scope(s):
#"transport_keys:post": "rule:admin"
```


INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

`barbican.plugin.crypto.base`, [127](#)

`barbican.plugin.interface.secret_store`,
[122](#)

A

algorithm (barbican.plugin.crypto.base.GenerateDTO attribute), 129

algorithm (barbican.plugin.crypto.base.KEKMetaDTO attribute), 128

AsymmetricKeyMetadataDTO (class in barbican.plugin.interface.secret_store), 123

B

barbican.plugin.crypto.base module, 127

barbican.plugin.interface.secret_store module, 122

bind_kek_metadata() (barbican.plugin.crypto.base.CryptoPluginBase method), 129

bit_length (barbican.plugin.crypto.base.GenerateDTO attribute), 129

bit_length (barbican.plugin.crypto.base.KEKMetaDTO attribute), 128

C

CryptoPluginBase (class in barbican.plugin.crypto.base), 129

D

decrypt() (barbican.plugin.crypto.base.CryptoPluginBase method), 130

DecryptDTO (class in barbican.plugin.crypto.base), 128

delete_secret() (barbican.plugin.interface.secret_store.SecretStoreBase method), 124

E

encrypt() (barbican.plugin.crypto.base.CryptoPluginBase

method), 130

EncryptDTO (class in barbican.plugin.crypto.base), 128

encrypted (barbican.plugin.crypto.base.DecryptDTO attribute), 128

G

generate_asymmetric() (barbican.plugin.crypto.base.CryptoPluginBase method), 130

generate_asymmetric_key() (barbican.plugin.interface.secret_store.SecretStoreBase method), 124

generate_supports() (barbican.plugin.interface.secret_store.SecretStoreBase method), 124

generate_symmetric() (barbican.plugin.crypto.base.CryptoPluginBase method), 131

generate_symmetric_key() (barbican.plugin.interface.secret_store.SecretStoreBase method), 124

GenerateDTO (class in barbican.plugin.crypto.base), 128

generation_type (barbican.plugin.crypto.base.GenerateDTO attribute), 129

get_plugin_name() (barbican.plugin.crypto.base.CryptoPluginBase method), 131

get_plugin_name() (barbican.plugin.interface.secret_store.SecretStoreBase method), 124

get_secret() (barbican.plugin.interface.secret_store.SecretStoreBase method), 125

get_transport_key() (barbican.plugin.interface.secret_store.SecretStoreBase method), 125

I

`is_transport_key_current()` (*barbican.plugin.interface.secret_store.SecretStoreBase* method), 125

`store_secret()` (*barbican.plugin.interface.secret_store.SecretStoreBase* method), 125

K

`kek_label` (*barbican.plugin.crypto.base.KEKMetaDTO* attribute), 128

`store_secret_supports()` (*barbican.plugin.interface.secret_store.SecretStoreBase* method), 125

`supports()` (*barbican.plugin.crypto.base.CryptoPluginBase* method), 131

U

`KEKMetaDTO` (class in *barbican.plugin.crypto.base*), 127

`KeyAlgorithm` (class in *barbican.plugin.interface.secret_store*), 123

`unencrypted` (*barbican.plugin.crypto.base.EncryptDTO* attribute), 128

`KeySpec` (class in *barbican.plugin.interface.secret_store*), 123

M

`mode` (*barbican.plugin.crypto.base.GenerateDTO* attribute), 129

`mode` (*barbican.plugin.crypto.base.KEKMetaDTO* attribute), 128

`module`

- `barbican.plugin.crypto.base`, 127
- `barbican.plugin.interface.secret_store`, 122

P

`plugin_meta` (*barbican.plugin.crypto.base.KEKMetaDTO* attribute), 128

`plugin_name` (*barbican.plugin.crypto.base.KEKMetaDTO* attribute), 127

R

`ResponseDTO` (class in *barbican.plugin.crypto.base*), 129

S

`SecretDTO` (class in *barbican.plugin.interface.secret_store*), 123

`SecretStoreBase` (class in *barbican.plugin.interface.secret_store*), 124

`SecretType` (class in *barbican.plugin.interface.secret_store*), 123