keystonemiddleware Documentation

Release 9.2.1.dev9

Openstack Developers

Apr 16, 2024

CONTENTS

This is the middleware provided for integrating with the OpenStack Identity API and handling authorization enforcement based upon the data within the OpenStack Identity tokens. Also included is middleware that provides the ability to create audit events based on API requests.

Contents:

CHAPTER

ONE

MIDDLEWARE ARCHITECTURE

1.1 Abstract

The keystonemiddleware architecture supports a common authentication protocol in use between the OpenStack projects. By using keystone as a common authentication and authorization mechanism, various OpenStack projects can leverage the existing authentication and authorization systems in use.

In this document, we describe the architecture and responsibilities of the authentication middleware which acts as the internal API mechanism for OpenStack projects based on the WSGI standard.

This documentation describes the implementation in *keystonemiddleware.auth_token*

1.2 Specification Overview

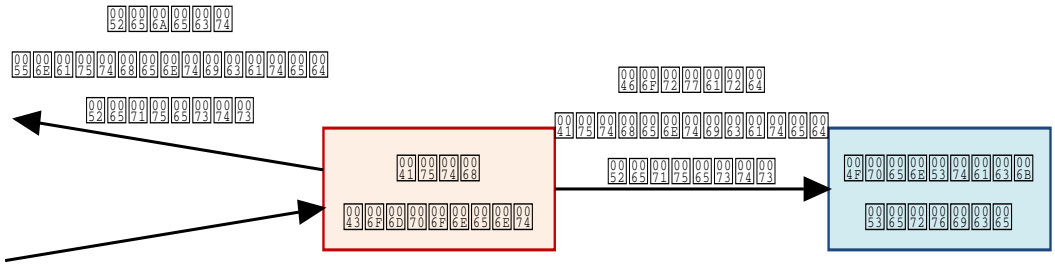
Authentication is the process of determining that users are who they say they are. Typically, authentication protocols such as HTTP Basic Auth, Digest Access, public key, token, etc, are used to verify a users identity. In this document, we define an authentication component as a software module that implements an authentication protocol for an OpenStack service. Bearer tokens are currently the most common authentication protocol used within OpenStack.

At a high level, an authentication middleware component is a proxy that intercepts HTTP calls from clients and populates HTTP headers in the request context for other WSGI middleware or applications to use. The general flow of the middleware processing is:

- clear any existing authorization headers to prevent forgery
- collect the token from the existing HTTP request headers
- validate the token
  - if valid, populate additional headers representing the identity that has been authenticated and authorized
  - if invalid, or no token present, reject the request (HTTPUnauthorized) or pass along a header indicating the request is unauthorized (configurable in the middleware)
  - if the keystone service is unavailable to validate the token, reject the request with HTTPServiceUnavailable.

1.2.1 Authentication Component

The following shows the default behavior of an Authentication Component deployed in front of an OpenStack service.

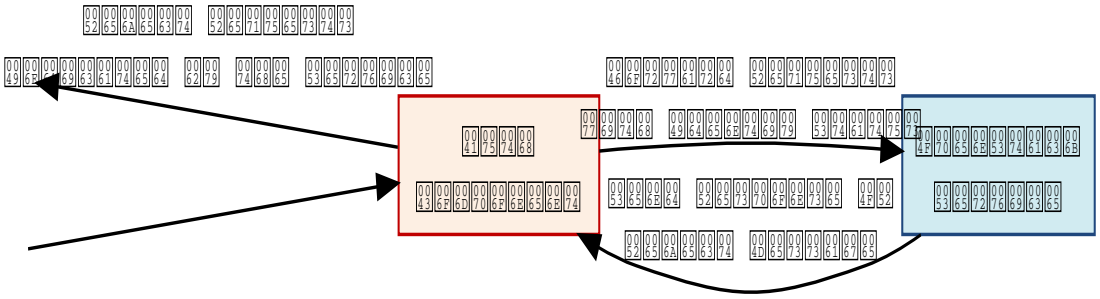


The Authentication Component, or middleware, will reject any unauthenticated requests, only allowing authenticated requests through to the OpenStack service.

1.2.2 Authentication Component (Delegated Mode)

The Authentication Component may be configured to operate in a delegated mode. In this mode, the decision to reject or accept an unauthenticated client is delegated to the OpenStack service.

Here, requests are forwarded to the OpenStack service with an identity status message that indicates whether the identity of the client has been confirmed or is indeterminate. The consuming OpenStack service decides whether or not a rejection message should be sent to the client.



1.3 Deployment Strategy

The middleware is intended to be used inline with OpenStack WSGI components, based on the Oslo WSGI middleware class. It is typically deployed as a configuration element in a paste configuration pipeline of other middleware components, with the pipeline terminating in the service application. The middleware conforms to the python WSGI standard [PEP-333]. In initializing the middleware, a configuration item (which acts like a python dictionary) is passed to the middleware with relevant configuration options.

1.3.1 Configuration

The middleware is configured within the config file of the main application as a WSGI component. Example for the auth_token middleware:

```
[app:myService]
paste.app_factory = myService:app_factory

[pipeline:main]
pipeline = authtoken myService

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

```
[DEFAULT]


[keystone_authtoken]

#
# From keystonemiddleware.auth_token
#

# 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. (string
# value)
# Deprecated group/name - [keystone_authtoken]/auth_uri
#www_authenticate_uri = <None>

# DEPRECATED: 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. (string value)
# 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_uri = <None>
```

(continues on next page)

(continued from previous page)

```
# API version of the Identity API endpoint. (string value)
#auth_version = <None>

# Interface to use for the Identity API endpoint. Valid values are "public",
# "internal" (default) or "admin". (string value)
#interface = internal

# Do not handle authorization requests within the middleware, but delegate the
# authorization decision to downstream WSGI components. (boolean value)
#delay_auth_decision = false

# Request timeout value for communicating with Identity API server. (integer
# value)
#http_connect_timeout = <None>

# How many times are we trying to reconnect when communicating with Identity
# API Server. (integer value)
#http_request_max_retries = 3

# 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. (string value)
#cache = <None>

# Required if identity server requires client certificate (string value)
#certfile = <None>

# Required if identity server requires client certificate (string value)
#keyfile = <None>

# A PEM encoded Certificate Authority to use when verifying HTTPs connections.
# Defaults to system CAs. (string value)
#cafile = <None>

# Verify HTTPS connections. (boolean value)
#insecure = false

# The region in which the identity server can be found. (string value)
#region_name = <None>

# Optionally specify a list of memcached server(s) to use for caching. If left
# undefined, tokens will instead be cached in-process. (list value)
# Deprecated group/name - [keystone_authtoken]/memcache_servers
#memcached_servers = <None>

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

(continues on next page)

(continued from previous page)

```
# to -1 to disable caching completely. (integer value)
#token_cache_time = 300

# (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. (string value)
# Possible values:
# None - <No description provided>
# MAC - <No description provided>
# ENCRYPT - <No description provided>
#memcache_security_strategy = None

# (Optional, mandatory if memcache_security_strategy is defined) This string␣
↪is
# used for key derivation. (string value)
#memcache_secret_key = <None>

# (Optional) Number of seconds memcached server is considered dead before it␣
↪is
# tried again. (integer value)
#memcache_pool_dead_retry = 300

# (Optional) Maximum total number of open connections to every memcached
# server. (integer value)
#memcache_pool_maxsize = 10

# (Optional) Socket timeout in seconds for communicating with a memcached
# server. (integer value)
#memcache_pool_socket_timeout = 3

# (Optional) Number of seconds a connection to memcached is held unused in the
# pool before it is closed. (integer value)
#memcache_pool_unused_timeout = 60

# (Optional) Number of seconds that an operation will wait to get a memcached
# client connection from the pool. (integer value)
#memcache_pool_conn_get_timeout = 10

# (Optional) Use the advanced (eventlet safe) memcached client pool. The
# advanced pool will only work under python 2.x. (boolean value)
#memcache_use_advanced_pool = false

# (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. (boolean value)
#include_service_catalog = true
```

(continues on next page)

(continued from previous page)

```
# 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.
# (string value)
#enforce_token_bind = permissive

# 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. (list value)
#service_token_roles = service

# For backwards compatibility reasons we must let valid service tokens pass
# that don't 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. (boolean value)
#service_token_roles_required = false

# 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. (string value)
#service_type = <None>

# Authentication type to load (string value)
# Deprecated group/name - [keystone_authtoken]/auth_plugin
#auth_type = <None>

# Config Section from which to load plugin specific options (string value)
#auth_section = <None>
```

If the `auth_type` configuration option is set, you may need to refer to the Authentication Plugins document for how to configure the auth_token middleware.

For services which have a separate paste-deploy ini file, auth_token middleware can be alternatively configured in [keystone_authtoken] section in the main config file. For example in nova, all middleware parameters can be removed from `api-paste.ini`:

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

and set in `nova.conf`:

```
[DEFAULT]
auth_strategy=keystone

[keystone_authtoken]
identity_uri = http://127.0.0.1:5000
admin_user = admin
admin_password = SuperSekretPassword
admin_tenant_name = service
# Any of the options that could be set in api-paste.ini can be set here.
```

Note: Middleware parameters in paste config take priority and must be removed to use options in the [keystone_authtoken] section.

The following is an example of a services auth_token middleware configuration when `auth_type` is set to `password`.

```
[keystone_authtoken]
auth_type = password
project_domain_name = Default
project_name = service
user_domain_name = Default
username = nova
password = ServicePassword
interface = public
auth_url = http://127.0.0.1:5000
# Any of the options that could be set in api-paste.ini can be set here.
```

If using an `auth_type`, connection to the Identity service will be established on the `interface` as registered in the service catalog. In the case where you are using an `auth_type` and have multiple regions, also specify the `region_name` option to fetch the correct endpoint.

If the service doesnt use the global oslo.config object (CONF), then the oslo config project name can be set it in paste config and keystonemiddleware will load the project configuration itself. Optionally the location of the configuration file can be set if oslo.config is not able to discover it.

```
[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
oslo_config_project = nova
# oslo_config_file = /not_discoverable_location/nova.conf
```

1.3.2 Improving response time

Validating the identity of every client on every request can impact performance for both the OpenStack service and the identity service. As a result, keystonemiddleware is configurable to cache authentication responses from the identity service in-memory. It is worth noting that tokens invalidated after theyve been stored in the cache may continue to work. Deployments using memcached may use the following keystonemiddleware configuration options instead of an in-memory cache.

- `memcached_servers`: (optional) if defined, the memcached server(s) to use for caching. It will be ignored if Swift MemcacheRing is used instead.
- `token_cache_time`: (optional, default 300 seconds) Set to -1 to disable caching completely.

When deploying auth_token middleware with Swift, user may elect to use Swift MemcacheRing instead of the local Keystone memcache. The Swift MemcacheRing object is passed in from the request environment and it defaults to swift.cache. However it could be different, depending on deployment. To use Swift MemcacheRing, you must provide the `cache` option.

- `cache`: (optional) if defined, the environment key where the Swift MemcacheRing object is stored.

1.4 Memcached dependencies

In order to use memcached it is necessary to install the python-memcached library. If data stored in memcached will need to be encrypted it is also necessary to install the pycrypto library. These libs are not listed in the requirements.txt file.

1.5 Memcache Protection

When using memcached, tokens and authentication responses are stored in the cache as raw data. In the event the cache is compromised, all token and authentication responses will be readable. To mitigate this risk, `auth_token` middleware provides an option to authenticate and optionally encrypt the token data stored in the cache.

- `memcache_security_strategy`: (optional) if defined, indicate whether token data should be authenticated or authenticated and encrypted. Acceptable values are `MAC` or `ENCRYPT`. 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`: (optional, mandatory if `memcache_security_strategy` is defined) this string is used for key derivation. If `memcache_security_strategy` is defined and `memcache_secret_key` is absent, `auth_token` will raise an exception on initialization.

1.6 Exchanging User Information

The middleware expects to find a token representing the user with the header `X-Auth-Token` or `X-Storage-Token`. *X-Storage-Token* is supported for swift/cloud files and for legacy Rackspace use. If the token isnt present and the middleware is configured to not delegate auth responsibility, it will respond to the HTTP request with HTTPUnauthorized, returning the header `WWW-Authenticate` with the value *Keystone uri=* to indicate where to request a token. The URI returned is configured with the `www_authenticate_uri` option.

The authentication middleware extends the HTTP request with the header `X-Identity-Status`. If a request is successfully authenticated, the value is set to *Confirmed*. If the middleware is delegating the auth decision to the service, then the status is set to *Invalid* if the auth request was unsuccessful.

An `X-Service-Token` header may also be included with a request. If present, and the value of `X-Auth-Token` or `X-Storage-Token` has not caused the request to be denied, then the middleware will attempt to validate the value of `X-Service-Token`. If valid, the authentication middleware extends the HTTP request with the header `X-Service-Identity-Status` having value *Confirmed* and also extends the request with additional headers representing the identity authenticated and authorised by the token.

If `X-Service-Token` is present and its value is invalid and the `delay_auth_decision` option is True then the value of `X-Service-Identity-Status` is set to *Invalid* and no further headers are added. Otherwise if `X-Service-Token` is present and its value is invalid then the middleware will respond to the HTTP request with HTTPUnauthorized, regardless of the validity of the `X-Auth-Token` or `X-Storage-Token` values.

1.6.1 Extended the request with additional User Information

keystonemiddleware.auth_token.AuthProtocol extends the request with additional information if the user has been authenticated. See the What we add to the request for use by the OpenStack service section in *keystonemiddleware.auth_token* for the list of fields set by the auth_token middleware.

1.7 References

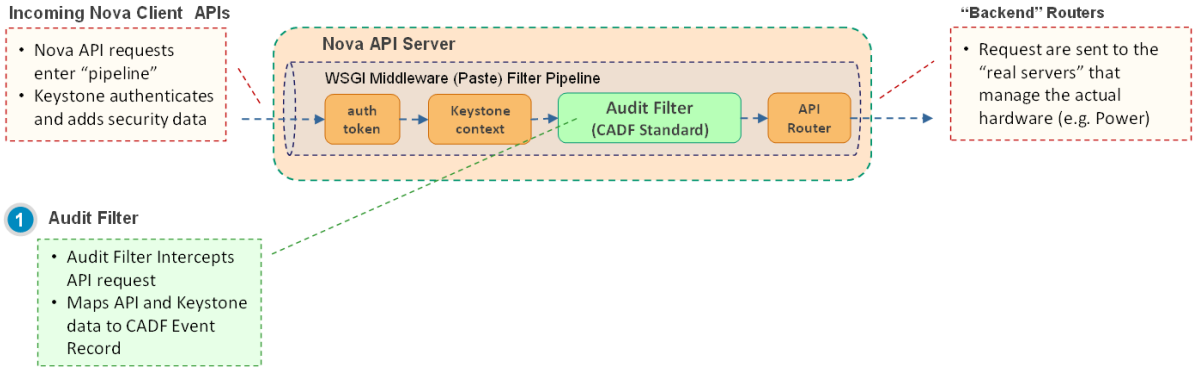
CHAPTER

TWO

AUDIT MIDDLEWARE

The Keystone middleware library provides an optional WSGI middleware filter which allows the ability to audit API requests for each component of OpenStack.

The audit middleware filter utilises environment variables to build the CADF event.



The figure above shows the middleware in Novas pipeline.

2.1 Enabling audit middleware

To enable auditing, oslo.messaging should be installed. If not, the middleware will log the audit event instead. Auditing can be enabled for a specific project by editing the projects api-paste.ini file to include the following filter definition:

```
[filter:audit]
paste.filter_factory = keystonemiddleware.audit:filter_factory
audit_map_file = /etc/nova/api_audit_map.conf
```

The filter should be included after Keystone middlewares auth_token middleware so it can utilise environment variables set by auth_token. Below is an example using Novas WSGI pipeline:

```
[composite:openstack_compute_api_v2]
use = call:nova.api.auth:pipeline_factory
noauth = faultwrap sizelimit noauth ratelimit osapi_compute_app_v2
keystone = faultwrap sizelimit authtoken keystonecontext ratelimit audit␣
↪osapi_compute_app_v2
```

(continues on next page)

(continued from previous page)

```
keystone_nolimit = faultwrap sizelimit authtoken keystonecontext audit osapi_
↪compute_app_v2
```

2.2 Configure audit middleware

To properly audit api requests, the audit middleware requires an api_audit_map.conf to be defined. The projects corresponding api_audit_map.conf file is included in the pyCADF library.

The location of the mapping file should be specified explicitly by adding the path to the audit_map_file option of the filter definition:

```
[filter:audit]
paste.filter_factory = keystonemiddleware.audit:filter_factory
audit_map_file = /etc/nova/api_audit_map.conf
```

Additional options can be set:

```
[filter:audit]
paste.filter_factory = pycadf.middleware.audit:filter_factory
audit_map_file = /etc/nova/api_audit_map.conf
service_name = test # opt to set HTTP_X_SERVICE_NAME environ variable
ignore_req_list = GET,POST # opt to ignore specific requests
```

Audit middleware can be configured to use its own exclusive notification driver and topic(s) value. This can be useful when the service is already using oslo messaging notifications and wants to use a different driver for auditing e.g. service has existing notifications sent to queue via messagingv2 and wants to send audit notifications to a log file via log driver. Example shown below:

```
[audit_middleware_notifications]
driver = log
```

When audit events are sent via messagingv2 or messaging, middleware can specify a transport URL if its transport URL needs to be different from the services own messaging transport setting. Other Transport related settings are read from oslo messaging sections defined in service configuration e.g. oslo_messaging_rabbit. Example shown below:

```
[audit_middleware_notifications]
driver = messaging
transport_url = rabbit://user2:passwd@host:5672/another_virtual_host
```

CHAPTER

THREE

INSTALLATION

3.1 Install using pip

At the command line:

```
$ pip install keystonemiddleware
```

Or, if you want to use it in a virtualenvwrapper:

```
$ mkvirtualenv keystonemiddleware
$ pip install keystonemiddleware
```

3.2 Install optional dependencies

Certain keystonemiddleware features are only available if specific libraries are available. These libraries can be installed using pip as well.

To install support for audit notifications:

```
$ pip install keystonemiddleware[audit_notifications]
```

CHAPTER

FOUR

RELATED IDENTITY PROJECTS

In addition to creating the Python Middleware for OpenStack Identity API, the Keystone team also provides Identity Service, as well as Python Client Library.

CHAPTER

FIVE

RELEASE NOTES

CHAPTER

SIX

CONTRIBUTING

Code is hosted on GitHub. Submit bugs to the Keystone project on Launchpad. Submit code to the `openstack/keystonemiddleware` project using Gerrit.

Run tests with `tox -e py27` if running with python 2.7. See the `tox.ini` file for other test environment options.

CHAPTER

SEVEN

CODE DOCUMENTATION

7.1 keystonemiddleware

7.1.1 keystonemiddleware package

Subpackages

keystonemiddleware.audit package

Module contents

Build open standard audit information based on incoming requests.

AuditMiddleware filter should be placed after keystonemiddleware.auth_token in the pipeline so that it can utilise the information the Identity server provides.

class keystonemiddleware.audit.**AuditMiddleware**(*app*, ***conf*)

Bases: object

Create an audit event based on request/response.

The audit middleware takes in various configuration options such as the ability to skip audit of certain requests. The full list of options can be discovered here: https://docs.openstack.org/keystonemiddleware/latest/audit.html

keystonemiddleware.audit.**filter_factory**(*global_conf*, ***local_conf*)

Return a WSGI filter app for use with paste.deploy.

keystonemiddleware.auth_token package

Module contents

Token-based Authentication Middleware.

This WSGI component:

- Verifies that incoming client requests have valid tokens by validating tokens with the auth service.
- Rejects unauthenticated requests unless the auth_token middleware is in delay_auth_decision mode, which means the final decision is delegated to the downstream WSGI component (usually the OpenStack service).

- Collects and forwards identity information based on a valid token such as user name, domain, project, etc.

Refer to: https://docs.openstack.org/keystonemiddleware/latest/middlewarearchitecture.html

Headers

The auth_token middleware uses headers sent in by the client on the request and sets headers and environment variables for the downstream WSGI component.

Coming in from initial call from client or customer

HTTP_X_AUTH_TOKEN
The client token being passed in.

HTTP_X_SERVICE_TOKEN
A service token being passed in.

Used for communication between components

WWW-Authenticate
HTTP header returned to a user indicating which endpoint to use to retrieve a new token.

What auth_token adds to the request for use by the OpenStack service

When using composite authentication (a user and service token are present) additional service headers relating to the service user will be added. They take the same form as the standard headers but add `_SERVICE_`. These headers will not exist in the environment if no service token is present.

HTTP_X_IDENTITY_STATUS, HTTP_X_SERVICE_IDENTITY_STATUS
Will be set to either `Confirmed` or `Invalid`.

The underlying service will only see a value of Invalid if the middleware is configured to run in `delay_auth_decision` mode. As with all such headers, `HTTP_X_SERVICE_IDENTITY_STATUS` will only exist in the environment if a service token is presented. This is different than `HTTP_X_IDENTITY_STATUS` which is always set even if no user token is presented. This allows the underlying service to determine if a denial should use `401 Unauthenticated` or `403 Forbidden`.

HTTP_OPENSTACK_SYSTEM_SCOPE
A string relaying system information about the tokens scope. This attribute is only present if the token is system-scoped. The string `all` means the token is scoped to the entire deployment system.

HTTP_X_DOMAIN_ID, HTTP_X_SERVICE_DOMAIN_ID
Identity service managed unique identifier, string. Only present if this is a domain-scoped token.

HTTP_X_DOMAIN_NAME, HTTP_X_SERVICE_DOMAIN_NAME
Unique domain name, string. Only present if this is a domain-scoped token.

HTTP_X_PROJECT_ID, HTTP_X_SERVICE_PROJECT_ID
Identity service managed unique identifier, string. Only present if this is a project-scoped token.

HTTP_X_PROJECT_NAME, HTTP_X_SERVICE_PROJECT_NAME
Project name, unique within owning domain, string. Only present if this is a project-scoped token.

HTTP_X_PROJECT_DOMAIN_ID, HTTP_X_SERVICE_PROJECT_DOMAIN_ID
Identity service managed unique identifier of owning domain of project, string. Only present if this is a project-scoped v3 token. If this variable is set, this indicates that the PROJECT_NAME can only be assumed to be unique within this domain.

HTTP_X_PROJECT_DOMAIN_NAME, HTTP_X_SERVICE_PROJECT_DOMAIN_NAME
Name of owning domain of project, string. Only present if this is a project-scoped v3 token. If this variable is set, this indicates that the PROJECT_NAME can only be assumed to be unique within this domain.

HTTP_X_USER_ID, HTTP_X_SERVICE_USER_ID
Identity-service managed unique identifier, string.

HTTP_X_USER_NAME, HTTP_X_SERVICE_USER_NAME
User identifier, unique within owning domain, string.

HTTP_X_USER_DOMAIN_ID, HTTP_X_SERVICE_USER_DOMAIN_ID
Identity service managed unique identifier of owning domain of user, string. If this variable is set, this indicates that the USER_NAME can only be assumed to be unique within this domain.

HTTP_X_USER_DOMAIN_NAME, HTTP_X_SERVICE_USER_DOMAIN_NAME
Name of owning domain of user, string. If this variable is set, this indicates that the USER_NAME can only be assumed to be unique within this domain.

HTTP_X_ROLES, HTTP_X_SERVICE_ROLES
Comma delimited list of case-sensitive role names.

HTTP_X_IS_ADMIN_PROJECT
The string value True or False representing whether the users token is scoped to the admin project. As historically there was no admin project this will default to True for tokens without this information to be backwards with existing policy files.

HTTP_X_SERVICE_CATALOG
service catalog (optional, JSON string).

For compatibility reasons this catalog will always be in the V2 catalog format even if it is a v3 token.

Note: This is an exception in that it contains SERVICE but relates to a user token, not a service token. The existing users catalog can be very large; it was decided not to present a catalog relating to the service token to avoid using more HTTP header space.

HTTP_X_TENANT_ID
Deprecated in favor of HTTP_X_PROJECT_ID.

Identity service managed unique identifier, string. For v3 tokens, this will be set to the same value as HTTP_X_PROJECT_ID.

HTTP_X_TENANT_NAME
Deprecated in favor of HTTP_X_PROJECT_NAME.

Project identifier, unique within owning domain, string. For v3 tokens, this will be set to the same value as HTTP_X_PROJECT_NAME.

HTTP_X_TENANT
Deprecated in favor of HTTP_X_TENANT_ID and HTTP_X_TENANT_NAME.

Identity server-assigned unique identifier, string. For v3 tokens, this will be set to the same value as HTTP_X_PROJECT_ID.

HTTP_X_USER
Deprecated in favor of HTTP_X_USER_ID and HTTP_X_USER_NAME.

User name, unique within owning domain, string.

HTTP_X_ROLE
Deprecated in favor of HTTP_X_ROLES.

Will contain the same values as HTTP_X_ROLES.

Environment Variables

These variables are set in the request environment for use by the downstream WSGI component.

keystone.token_info
Information about the token discovered in the process of validation. This may include extended information returned by the token validation call, as well as basic information about the project and user.

keystone.token_auth
A keystoneauth1 auth plugin that may be used with a `keystoneauth1.session.Session`. This plugin will load the authentication data provided to auth_token middleware.

Configuration

auth_token middleware configuration can be in the main applications configuration file, e.g. in `nova.conf`:

```
[keystone_authtoken]
auth_plugin = password
auth_url = http://keystone:5000/
username = nova
user_domain_id = default
password = whyarewestillusingpasswords
project_name = service
project_domain_id = default
```

Configuration can also be in the `api-paste.ini` file with the same options, but this is discouraged.

Swift

When deploy auth_token middleware with Swift, user may elect to use Swift memcache instead of the local auth_token memcache. Swift memcache is passed in from the request environment and its identified by the `swift.cache` key. However it could be different, depending on deployment. To use Swift memcache, you must set the `cache` option to the environment key where the Swift cache object is stored.

class keystonemiddleware.auth_token.**AuthProtocol**(*app*, *conf*)

Bases: *BaseAuthProtocol*

Middleware that handles authenticating client calls.

fetch_token(*token*, *allow_expired=False*)

Retrieve a token from either a PKI bundle or the identity server.

Parameters
token (*str*) token id

Raises
exc.InvalidToken if token is rejected

process_request(*request*)

Process request.

Evaluate the headers in a request and attempt to authenticate the request. If authenticated then additional headers are added to the request for use by applications. If not authenticated the request will be rejected or marked unauthenticated depending on configuration.

process_response(*response*)

Process Response.

Add `WWW-Authenticate` headers to requests that failed with `401 Unauthenticated` so users know where to authenticate for future requests.

class keystonemiddleware.auth_token.**BaseAuthProtocol**(*app*, *log=<KeywordArgumentAdapter keystonemiddleware.auth_token (WARNING)>*, *enforce_token_bind='permissive'*, *service_token_roles=None*, *service_token_roles_required=False*, *service_type=None*)

Bases: `object`

A base class for AuthProtocol token checking implementations.

Parameters

- **app** (*Callable*) The next application to call after middleware.
- **log** (*logging.Logger*) The logging object to use for output. By default it will use a logger in the keystonemiddleware.auth_token namespace.
- **enforce_token_bind** (*str*) The style of token binding enforcement to perform.

fetch_token(*token*, ***kwargs*)

Fetch the token data based on the value in the header.

Retrieve the data associated with the token value that was in the header. This can be from PKI, contacting the identity server or whatever is required.

Parameters

- **token** (*str*) The token present in the request header.
- **kwargs** (*dict*) Additional keyword arguments may be passed through here to support new features. If an implementation is not aware of how to use these arguments it should ignore them.

Raises
exc.InvalidToken if token is invalid.

Returns
The token data

Return type
dict

process_request(*request*)

Process request.

If this method returns a value then that value will be used as the response. The next application down the stack will not be executed and process_response will not be called.

Otherwise, the next application down the stack will be executed and process_response will be called with the generated response.

By default this method does not return a value.

Parameters
request (*_request.AuthTokenRequest*) Incoming request

process_response(*response*)

Do whatever youd like to the response.

By default the response is returned unmodified.

Parameters
response (*._request._AuthTokenResponse*) Response object

validate_allowed_request(*request*, *token*)

keystonemiddleware.auth_token.**app_factory**(*global_conf*, ***local_conf*)

keystonemiddleware.auth_token.**filter_factory**(*global_conf*, ***local_conf*)

Return a WSGI filter app for use with paste.deploy.

keystonemiddleware.auth_token.**list_opts**()

Return a list of oslo_config options available in auth_token middleware.

The returned list includes all oslo_config options which may be registered at runtime by the project.

Each element of the list is a tuple. The first element is the name of the group under which the list of elements in the second element will be registered. A group name of None corresponds to the [DEFAULT] group in config files.

NOTE: This function is no longer used for oslo_config sample generation. Some services rely on this function for listing ALL (including deprecated) options and registering them into their own config objects which we do not want for sample config files.

See: `keystonemiddleware.auth_token._opts.list_opts()` for sample config files.

Returns
a list of (group_name, opts) tuples

keystonemiddleware.echo package

Submodules

keystonemiddleware.echo.service module

Run the echo service.

The echo service can be run on port 8000 by executing the following:

```
$ python -m keystonemiddleware.echo
```

When the `auth_token` module authenticates a request, the echo service will respond with all the environment variables presented to it by this module.

class keystonemiddleware.echo.service.**EchoService**

Bases: `object`

Runs an instance of the echo app on init.

keystonemiddleware.echo.service.**echo_app**(*environ*, *start_response*)

A WSGI application that echoes the CGI environment back to the user.

Module contents

Submodules

keystonemiddleware.ec2_token module

Starting point for routing EC2 requests.

class keystonemiddleware.ec2_token.**EC2Token**(*application*, *conf*)

Bases: `object`

Authenticate an EC2 request with keystone and convert to token.

keystonemiddleware.ec2_token.**app_factory**(*global_conf*, ***local_conf*)

keystonemiddleware.ec2_token.**filter_factory**(*global_conf*, ***local_conf*)

Return a WSGI filter app for use with paste.deploy.

keystonemiddleware.exceptions module

exception keystonemiddleware.exceptions.**ConfigurationError**

Bases: *KeystoneMiddlewareException*

exception keystonemiddleware.exceptions.**KeystoneMiddlewareException**

Bases: Exception

keystonemiddleware.fixture module

class keystonemiddleware.fixture.**AuthTokenFixture**

Bases: Fixture

Overrides what keystonemiddleware will return to the app behind it.

add_token(*token_data*, *token_id=None*)

Add an existing token to the middleware.

Parameters

- **token_data** (*dict*) token data to add to the fixture
- **token_id** (*str*) the token ID to add this token as

Returns
The token_id that the token was added as.

Return type
str

add_token_data(*token_id=None*, *expires=None*, *user_id=None*, *user_name=None*, *user_domain_id=None*, *user_domain_name=None*, *project_id=None*, *project_name=None*, *project_domain_id=None*, *project_domain_name=None*, *role_list=None*, *is_v2=False*)

Add token data to the auth_token fixture.

fetch_token(*token*, ***kwargs*)

Low level replacement of fetch_token for AuthProtocol.

setUp()

Prepare the Fixture for use.

This should not be overridden. Concrete fixtures should implement _setUp. Overriding of setUp is still supported, just not recommended.

After setUp has completed, the fixture will have one or more attributes which can be used (these depend totally on the concrete subclass).

Raises
MultipleExceptions if _setUp fails. The last exception captured within the MultipleExceptions will be a SetupError exception.

Returns
None.

Changed in 1.3
The recommendation to override setUp has been reversed - before 1.3, setUp() should be overridden, now it should not be.

Changed in 1.3.1
BaseException is now caught, and only subclasses of Exception are wrapped in MultipleExceptions.

property tokens

keystonemiddleware.i18n module

oslo.i18n integration module.

See https://docs.openstack.org/oslo.i18n/latest/user/usage.html .

keystonemiddleware.opts module

keystonemiddleware.opts.**list_auth_token_opts**()

keystonemiddleware.s3_token module

S3 Token Middleware.

This WSGI component:

- Gets a request from the swift3 middleware with an S3 Authorization access key.
- Validates s3 token in Keystone.
- Transforms the account name to AUTH_%(tenant_name).

class keystonemiddleware.s3_token.**S3Token**(*app*, *conf*)

Bases: object

Middleware that handles S3 authentication.

exception keystonemiddleware.s3_token.**ServiceError**

Bases: Exception

keystonemiddleware.s3_token.**filter_factory**(*global_conf*, ***local_conf*)

Return a WSGI filter app for use with paste.deploy.

Module contents

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[PEP-333] pep0333 Phillip J Eby. Python Web Server Gateway Interface v1.0. http://www.python.org/dev/peps/pep-0333/.