

Technical Report

OpenStack Keystone Integration

An investigation into integrating StorageGRID Webscale with OpenStack Keystone

Matthew Yee, Technical Marketing Engineer, StorageGRID Software Group, NetApp  
December 2014

Abstract

OpenStack Keystone provides identity services for all OpenStack infrastructure. Of particular interest is the interplay between Keystone and OpenStack’s object storage service, Swift, its S3 interface and how S3 RESTful API calls are translated and authenticated. A stepwise analysis is documented and a prototype proxy for S3 API calls and Keystone authentication for StorageGRID Webscale is presented.

TABLE OF CONTENTS

[1 Introduction 3](#_Toc406512623)

[1.1 StorageGRID 3](#_Toc406512624)

[1.2 OpenStack Keystone 3](#_Toc406512625)

[1.3 Keystone Authentication for StorageGRID Webscale 3](#_Toc406512626)

[2 Amazon S3 RESTful API call 3](#_Toc406512627)

[2.1 S3 RESTful API call 4](#_Toc406512628)

[2.2 AWS v2 Signing 4](#_Toc406512629)

[3 Keystone Authentication 6](#_Toc406512630)

[3.1 WSGI pipeline 6](#_Toc406512631)

[3.2 Swift3 and s3\_token 7](#_Toc406512632)

[3.3 Keystone and /v2.0/s3tokens 7](#_Toc406512633)

[4 Prototype 7](#_Toc406512634)

[4.1 Operation 7](#_Toc406512635)

[4.2 Implementation 8](#_Toc406512636)

[5 Conclusions 9](#_Toc406512637)

[5.1 Topics for further investigation 9](#_Toc406512638)

[6 Appendixes 10](#_Toc406512639)

[6.1 POST request to Authenticate an S3 ‘token’ against Keystone 10](#_Toc406512640)

[7 Version History 10](#_Toc406512641)

LIST OF TABLES

No table of figures entries found.

LIST OF FIGURES

[Figure 1 Prototype Workflow 8](file:///C:\Users\MATTHEWY\Google%20Drive\StorageGRID\Docs\Keystone\Keystone%20Auth.docx#_Toc406512642)

[Figure 2 GET Request 8](file:///C:\Users\MATTHEWY\Google%20Drive\StorageGRID\Docs\Keystone\Keystone%20Auth.docx#_Toc406512643)

[Figure 3 POST Request 9](file:///C:\Users\MATTHEWY\Google%20Drive\StorageGRID\Docs\Keystone\Keystone%20Auth.docx#_Toc406512644)

# Introduction

The OpenStack project currently facilitates Amazon S3 object storage API requests to the Swift object storage service. This enables end users using Amazon S3 storage to build and use their own private on-premises object storage infrastructure.

This document explores a possible method of integrating OpenStack Keystone identity management with StorageGRID Webscale.

## StorageGRID

StorageGRID Webscale 10.0 supports the Amazon S3 API. In addition to supporting RESTful HTTP verbs specified by the Amazone S3 API, StorageGRID Webscale manages identity by associating resources with user accounts. It also issues S3 compatible Access Key IDs and Secret Keys.

## OpenStack Keystone

The OpenStack project includes an identity and authentication service called Keystone. This service provides a central repository from which component services of OpenStack can authenticate requests and users.

Swift, the object storage component of OpenStack provides an API translation service that converts Amazon S3 API calls to Swift API calls. This process authenticates S3 API calls against Keystone.

The Keystone service maintains user accounts and controls access to projects within the OpenStack. Amazon Elastic Compute Cloud credentials consisting of an Access Key ID and Access Secret are generated for each user. These credentials can be used in authenticating S3 API calls to Swift.

Keystone makes use of Amazon Web Services (AWS) version 2[[1]](#footnote-2) signatures to verify the authenticity of an S3 API call.

## Keystone Authentication for StorageGRID Webscale

One possible method of integration between StorageGRID Webscale and OpenStack Keystone is to map Keystone generated Access Key IDs and Secret Keys to StorageGRID Webscale Access Key IDs and Secret Keys.

In this paper, OpenStack Keystone and Swift authentication is documented and a prototype proxy for S3 API calls and Keystone authentication for StorageGRID Webscale is presented.

# Amazon S3 RESTful API call

Amazon’s S3 object storage service provides a RESTful API interface for customers to upload and download data (amongst other functions). The S3 API has been adopted as a defacto standard for third party applications and storage vendors (such as NetApp StorageGRID).

Authentication is achieved by signing the S3 request and is discussed below.

In an OpenStack environment, the S3 RESTful API call is sent to a Swift proxy service which translates the API call to a Swift API call and authenticates the request with Keystone before passing the Swift API call to the Swift service.

## S3 RESTful API call

The following is an example of a RESTful S3 API call that lists all buckets for a given endpoint. This was obtained directly from sniffing a TCP packet.

GET / HTTP/1.1

Host: 10.96.96.53:8080

Accept-Encoding: identity

Date: Wed, 10 Dec 2014 17:26:09 GMT

Content-Length: 0

Authorization: AWS 1b3e0523f37a4542aeb67edf378f1eaf:WlXcWTr3b5vBQU/A1vEwuAnxuuI=

User-Agent: Boto/2.34.0 Python/2.7.8 Windows/7

This particular request is formulated with an HTTP GET method via the Python Boto library.

This particular API provides the following output.

HTTP/1.1 200 OK

x-amz-id-2: tx5caf16447d0a4e3ba4ed8-0054888229

Content-Length: 315

x-amz-request-id: tx5caf16447d0a4e3ba4ed8-0054888229

Content-Type: application/xml

X-Trans-Id: tx5caf16447d0a4e3ba4ed8-0054888229

Date: Wed, 10 Dec 2014 17:26:01 GMT

<?xml version='1.0' encoding='UTF-8'?>

<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>admin:admin</ID><DisplayName>admin:admin</DisplayName></Owner><Buckets><Bucket><Name>test</Name><CreationDate>2009-02-03T16:45:09.000Z</CreationDate></Bucket></Buckets></ListAllMyBucketsResult>

## AWS v2 Signing

The above GET method request must be ‘signed’ so that the server can determine if the requestor can be trusted. If the requestor is trusted, then the server can decide to service the request. Amazon Web Services have several specifications for signing API calls. The method used by Keystone is AWS v2 signing.

The S3 request, signing process and authentication process is outlined in the following order:

1. The requestor formulates an S3 API request. E.g. GET <hostname> <resource>
2. The requestor signs the request with a Secret Key.
3. The requestor sends the S3 API request, including Access Key ID and signature.

The secret key is never transmitted.

In AWS v2, authentication is established by an endpoint server (like Keystone) by:

1. Obtaining the Secret Key based upon the Access Key ID in the request.
2. Calculating a signature based upon the body of the request.
3. Comparing the computed signature with the signature included in the request to see if it matches.

### The Authorization Header

Revisiting the S3 API request in the previous section, we have the following.

GET / HTTP/1.1

Host: 10.96.96.53:8080

Accept-Encoding: identity

Date: Wed, 10 Dec 2014 17:26:09 GMT

Content-Length: 0

Authorization: AWS 1b3e0523f37a4542aeb67edf378f1eaf:WlXcWTr3b5vBQU/A1vEwuAnxuuI=

User-Agent: Boto/2.34.0 Python/2.7.8 Windows/7

The above GET method request contains two major components. The first is the Authorization header (as highlighted in yellow below).

GET / HTTP/1.1

Host: 10.96.96.53:8080

Accept-Encoding: identity

Date: Wed, 10 Dec 2014 17:26:09 GMT

Content-Length: 0

Authorization: AWS 1b3e0523f37a4542aeb67edf378f1eaf:WlXcWTr3b5vBQU/A1vEwuAnxuuI=

User-Agent: Boto/2.34.0 Python/2.7.8 Windows/7

The Authorization header contains two components.

1. Access Key ID
2. Signature

In this particular request, the Access Key ID is 1b3e0523f37a4542aeb67edf378f1eaf.

The signature is WlXcWTr3b5vBQU/A1vEwuAnxuuI=.

### The Request Body

The second major component is the body (as highlighted in yellow below) of the request.

GET / HTTP/1.1

Host: 10.96.96.53:8080

Accept-Encoding: identity

Date: Wed, 10 Dec 2014 17:26:09 GMT

Content-Length: 0

Authorization: AWS 1b3e0523f37a4542aeb67edf378f1eaf:WlXcWTr3b5vBQU/A1vEwuAnxuuI=

User-Agent: Boto/2.34.0 Python/2.7.8 Windows/7

### Computing the AWS v2 Signature

The AWS v2 signature is composed of the following attributes that may or may not be contained in the request body.

1. HTTP method
2. Content-MD5
3. Content-Type
4. Date
5. URI

In the example request, only the following attributes are contained in the request body.

GET / HTTP/1.1

Host: 10.96.96.53:8080

Accept-Encoding: identity

Date: Wed, 10 Dec 2014 17:26:09 GMT

Content-Length: 0

1. HTTP Method - GET
2. Date - Wed, 10 Dec 2014 17:26:09 GMT
3. URI - /

Using Python, a ‘message’ is constructed out of these attributes.

message = 'GET\n\n\nWed, 10 Dec 2014 17:26:09 GMT\n/'

The ‘\n’ newline characters must be included in the message.

Using Python, the signature is then computed from the message, obtaining the following:

message = 'GET\n\n\nWed, 10 Dec 2014 17:26:09 GMT\n/'

signed\_message = base64.encodestring(hmac.new(secret\_key,

message,

hashlib.sha1).digest()).strip()

signed message: WlXcWTr3b5vBQU/A1vEwuAnxuuI=

# Keystone Authentication

Keystone is the OpenStack project’s identity (amongst other things) management service. After the S3 API call has been received by the Swift proxy service and translated into a Swift API call, the Access Key ID, signature and relevant request body attributes are passed to the Keystone server to authenticate the request. If the request is determined to be authentic, the Swift API call is passed on to the Swift service.

## WSGI pipeline

When the S3 API call is received by the Swift proxy service, the request passes through an extensible framework known as the WSGI pipeline.

Within the Swift proxy service, there is a framework for processing incoming requests. This is an implementation of a specification known as the Web Services Gateway Interface (WSGI). The WSGI is implemented in a ‘pipeline’ within the Swift Web Objects library (which is derived from the WebOb library) which parses incoming requests by assigning request elements to various variable within the WSGI environment.

From Wikipedia[[2]](#footnote-3):

“The WSGI has two sides: the "server" or "gateway" side, and the "application" or "framework" side. To process a WSGI request, the server side provides environment information and a callback function to the application side. The application processes the request, and returns the response to the server side using the callback function it was provided.

So-called WSGI middleware implements both sides of the API so that it can intermediate between a WSGI server and a WSGI application: the middleware acts as an application from some WSGI server's point of view and as a server from some WSGI application's point of view. A "middleware" component can perform such functions as:

Routing a request to different application objects based on the target URL, after changing the environment variables accordingly.

Allowing multiple applications or frameworks to run side-by-side in the same process

Load balancing and remote processing, by forwarding requests and responses over a network

Perform content postprocessing, such as applying XSLT stylesheets”

By default, a Swift installation is not configured to service S3 requests. That is, the required middleware is not configured to filter and route requests within the WSGI pipeline in the Swift proxy server. The swift3 and s3token apps must be inserted in the WSGI pipeline[[3]](#footnote-4).

## Swift3 and s3\_token

Swift3[[4]](#footnote-5) middleware translates S3 API requests into Swift API requests within the WSGI pipeline. S3token middleware authenticates S3 requests with Keystone within the WSGI pipeline. Swift3 passes Access Key ID, signature and relevant request body attributes to s3token. S3token passes this information to Keystone to authenticate the request.

## Keystone and /v2.0/s3tokens

OpenStack Keystone includes an API interface specifically for authenticating S3 requests to Swift object storage.

1. S3token makes a POST request containing the Access Key ID, signature, and a reversible hash of the relevant request body attributes of the original request to the Keystone server at the resource /v2.0/s3tokens[[5]](#footnote-6).
2. Keystone looks up the Secret Key that accompanies the Access Key ID.
3. Keystone then calculates a signature based upon the request body information and the Secret Key.
4. A comparison between the signatures is made. If the signatures match, a token is returned. If the signatures don’t match, an error message is returned.

If S3token receives a token from Keystone, it passes the Swift request down the pipeline. If an error is received, s3token returns an error.

# Prototype

A prototype proxy for S3 api calls made with Keystone generated credentials was created and demonstrated as a method of integrating the StorageGRID Webscale product with Keystone.

## Operation

The workflow for the processing of the S3 request via the proxy is:

1. Requestor obtains Access Key ID and Secret key (created in Keystone via EC2 credential generator) by logging into the Keystone server and downloading them.
2. Requestor formulates S3 request.
3. Authenticate ‘token’[[6]](#footnote-7) with Keystone.
4. If the ‘token’ is valid, return a token containing the Keystone service catalogue (amongst other things).
5. Pass S3 request to StorageGRID Webscale. This request will make use of S3 credentials created by StorageGRID Webscale.
6. Return result through the proxy, to the requestor.

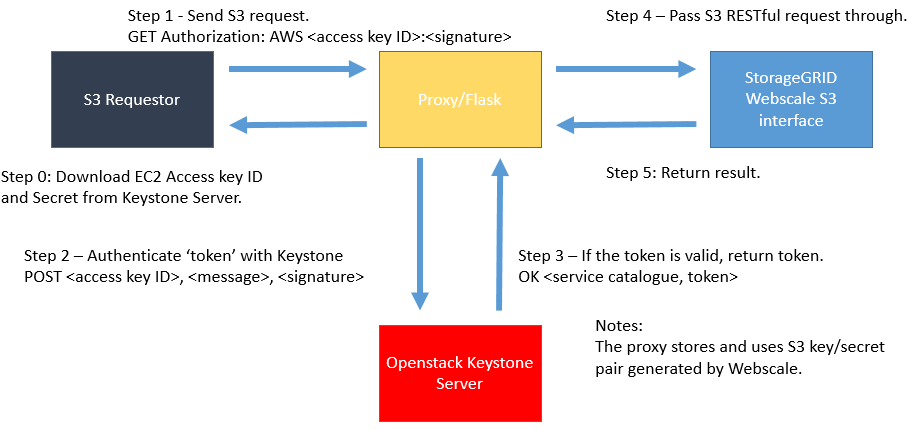


Figure 1 Prototype Workflow

## Implementation

The proxy was written as a Python Flask application.

The source is available here:

<http://git.vtc.eng.netapp.com/gitweb/?p=people/matthewy/keystone_prototype;a=blob_plain;f=keystone_prototype.py;hb=HEAD>

Using Advanced REST Client[[7]](#footnote-8), the following settings were used to construct fake S3 request.

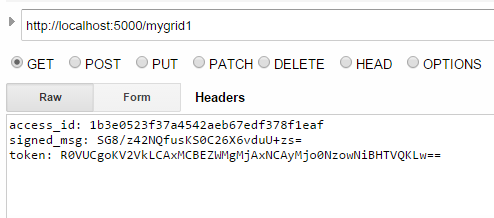
GET:

Figure 2 GET Request

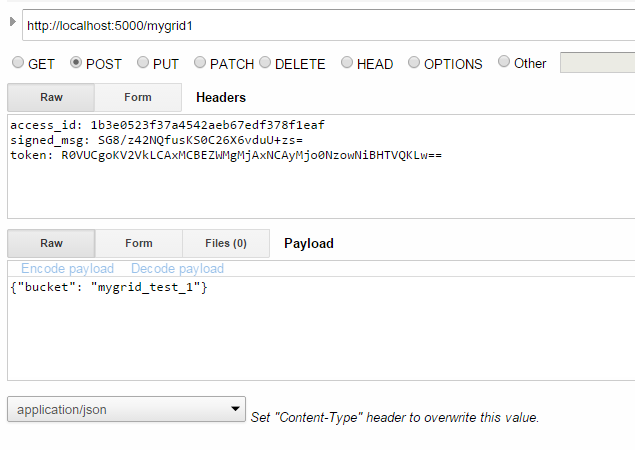
POST:

Figure 3 POST Request

# Conclusions

OpenStack documentation is extremely lacking but with a bit of persistence I think I learned two important facts:

1. To authenticate an AWS style request, a POST request with the Access Key ID, signature and S3 body must be sent to the Keystone server at the URI /v2.0/s3tokens
2. Keystone’s /v2.0/s3tokens API call uses AWS v2 signing.

## Topics for further investigation

1. Is it possible to inject StorageGRID Access Key/Secret pairs into Keystone?
2. If is possible to inject StorageGRID Access Key/Secret pairs into Keystone, is it possible to create a service catalogue entry for the StorageGRID Webscale system?

# Appendixes

## POST request to Authenticate an S3 ‘token’ against Keystone

Example code written in Python.

def check\_keystone\_token(access\_id, signed\_msg, token):

data = json.dumps({'credentials': {'access': access\_id, 'token': token, 'signature': signed\_msg}})

req = requests.post(

KEYSTONE\_SERVER\_S3,

headers={'Content-Type': 'application/json'},

data=data,

verify=None

)

if req.text:

return True

else:

return False

# Version History

As an option, use the NetApp Table style to create a Version History table. Do not add a table number or caption.

|  |  |  |
| --- | --- | --- |
| Version | Date | Document Version History |
| Version 0.1 | December 2014 | Initial draft. |

Refer to the [Interoperability Matrix Tool (IMT)](http://support.netapp.com/matrix/mtx/login.do) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.



NetApp provides no representations or warranties regarding the accuracy, reliability, or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information in this document is distributed AS IS, and the use of this information or the implementation of any recommendations or techniques herein is a customer’s responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. This document and   
the information contained herein may be used solely in connection with the NetApp products discussed   
in this document.

© 2014 NetApp, Inc. All rights reserved. No portions of this document may be reproduced without prior written consent of NetApp, Inc. Specifications are subject to change without notice. NetApp, the NetApp logo, xxx, and xxx are trademarks or registered trademarks of NetApp, Inc. in the United States and/or other countries. <<Insert third-party trademark notices here.>> All other brands or products are trademarks or registered trademarks of their respective holders and should be treated as such. TR-XXXXi-MMYR

1. <http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html> [↑](#footnote-ref-2)
2. [http://en.wikipedia.org/wiki/Web\_Server\_Gateway\_Interface 12/16/2014](http://en.wikipedia.org/wiki/Web_Server_Gateway_Interface%2012/16/2014), 11:44am PST [↑](#footnote-ref-3)
3. <http://docs.openstack.org/juno/config-reference/content/configuring-openstack-object-storage-with-s3_api.html> [↑](#footnote-ref-4)
4. <https://github.com/stackforge/swift3> [↑](#footnote-ref-5)
5. <https://wiki.openstack.org/wiki/Keystone-BP-S3Token> [↑](#footnote-ref-6)
6. The term ‘token’ is used within s3\_token.py code. The token variable contains the value of the hash of specific attributes contained in the S3 request body. [↑](#footnote-ref-7)
7. <https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo> [↑](#footnote-ref-8)