# OpenStack Swift as Object Storage Service

## Bug Reference

CLOUDSTACK-6230

## Branch

Here is a patch against the CS Master branch as of 2014/04/12...

Implements-OpenStack-Swift-as-an-Object-Storage-Service.patch

## Introduction

This implementation allows OpenStack Swift to be integrated directly into the CloudStack UI as an Object Storage Service to be used by the CloudStack end users.

## Purpose

*Everything in this document reflects functionality which has already been developed unless it is marked with 'TODO'. If you have comments or questions please let me know and I will update this document.*

This document lays out the integration of the OpenStack Swift Object Store as an object store service made available through the CloudStack UI. This integration allows CloudStack users to authenticate against Swift using their CloudStack credentials as well as exposing a UI in CloudStack for users to manage their documents in Swift.

A key enabling component for this integration is a Swift Auth middleware that I wrote called 'mauth'. This middleware supports CloudStack by default, but it is also extensible and open source. The middlware implementation uses the CloudStack username and api key as the Swift user credentials. After authentication, Swift will return a token to be used for the subsequent API calls.

### How 'mauth' works:

To get thing started I will show a curl example using mauth to authenticate a CloudStack user.
In this example, http://127.0.0.1:8080/v1.0 is pointing at the Swift cluster.
The $username and $apikey reflect the CloudStack user and his api key respectively.

```
Request for authentication
curl -v -H "X-Auth-User: $username" -H "X-Auth-Key: $apikey" http://127.0.0.1:8080/v1.0
returns: $auth_token and $swift_storage_url

Request container list
curl -v -X GET -H "X-Auth-Token: $auth_token" $swift_storage_url
```

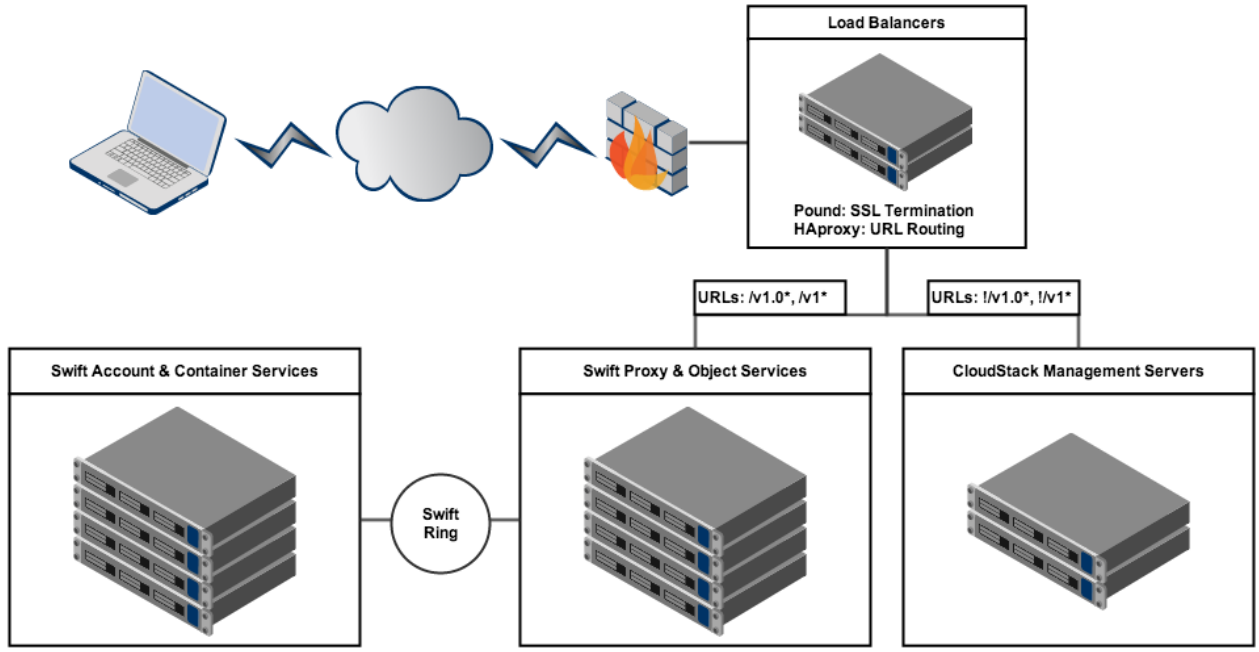
In this example, the user requests authentication and they are given back a token and their storage url for where their objects are stored. They can then use that token and url for subsequent requests.

### How the integration works:

A UI for Object Store is added to CloudStack in the left hand services menu. When a user clicks on the Object Store menu option, they are authenticated with the Swift backend (assuming they have api keys generated, if they don't, it will notify them that they need to generate an api key to use the service). Once authenticated, the user's containers and objects are listed in the main area using the treeView mechanism. Selecting a container or object gives details about it as well as shows actions that can be taken on it.

### A technical overview of the integration:

There are a few challenges with this integration. I will give a basic overview of the components in play and touch on some of the reasons for the decisions, but more detail will be added later in the document.




Because the object storage data is often large and very network intensive, we do not want the data to actually flow through the CloudStack box as it would introduce a huge bottleneck. Because of this, there is no Java component of this implementation in CloudStack. Instead, the traffic is routed to Swift or CloudStack, based on the url structure, by a load balancer which sits in front of the two services. The technology used for load balancing (and ssl termination) is not important, but to illustrate the concept, I used HAproxy and Pound for these tasks.

## References

- *mauth - An extensible middleware for OpenStack Swift which enables Swift to authenticate CloudStack users. (Developed by Will Stevens @ CloudOps)*
- *HAproxy configuration example*
- The patch against master from 2014/03/12

## Document History

1. The initial writing of the document (2014/03/11)
2. Added some details and added a patch file (2014/03/12)

## Glossary

CS - CloudStack
Swift - OpenStack Swift Object Storage

## Feature Specifications

- **'Object Store' added to the main services menu on the left**
- **When 'Object Store' is selected, the user is seamlessly authenticated and shown a UI for Swift**
- **User can List all containers and objects in their Swift account**
- **User can Add/Delete Swift Containers**
  - Supports cascade delete, so it will first delete all the objects in the container if there are any
- **User can Add/Delete Swift Objects**
- **User can Add/Delete Folders in containers to group objects (unique to this implementation)**
  - Supports cascade delete of folders, so it will first delete all the objects in the folder if there are any
- **User can create containers as Public or Private**
  - Public - Each object in the container can be accessed without needing authentication
  - Private - Each object in the container requires the user to authenticate to access the object
- **User can modify the Public/Private setting for existing containers**
- **A public URL is shown in the details for each object in a public container**
- **Supports sharing a Swift account with all users in a specific CS account (global setting)**
- **Supports giving each user in a CS account their own Swift account (global setting)**

- **When a user logs in, a change had to be made to add the 'api key' to the CS user object in the browser**
- **This Swift object storage service implementation is only available to users who have an API Key generated in CS**

    - It gracefully handles when a user does not have an api key generated and tells the user what to do in order for the functionality to work
    - Another option would be to hide the Object Store service menu unless the user has an API key, but that did not feel like a constructive approach
- **The load balancer exposes the URL to be used by both CloudStack and Swift calls**
    - One reason for this is to make sure the Swift traffic does not go through the CS server which would create a bottleneck
    - Another reason is to be able to make AJAX calls from the client to Swift without having issues with the Same Origin Policy
- **All Swift calls are done using AJAX and the load balancer handles routing the calls to Swift**
- **This implementation is entirely on the client side, without any Java backend in CloudStack**
- **Swift uses the HTTP PUT verb, but browsers do not support the PUT verb, so the client sends the PUT requests as POST and the LB translates them to PUT**
    - This means that the current implementation does not support any of the Swift API calls that require POST (this has not been a limiting factor yet)
    - More details on this can be found in the HAproxy config attached at the end of this document
        - Line: *reqirep ^POST\s+(.*)$ PUT\ \1*
- **This functionality requires the 'mauth' middleware to be installed and configured in the Swift cluster**
- **The mauth configuration points to CS (the load balancer actually) and requires the 'admin' api keys**
    - mauth needs access to the 'listUsers' CS API call for all accounts
- **It is VERY strongly recommended to use CS on HTTPS (with SSL)**
- **Swift Usage Reporting is not included in this integration (see the note at the end of this file)**
- **TODO - Important - A global variable needs to be added to CS to enable/disable the Object Store functionality**
- **TODO - Medium - Clean up the couple places where text is not translatable**
- **TODO - Nice-to-have - Find a cleaner solution when someone tries to download a folder or container**
    - currently downloading a container will result in an XML list of all of the objects in the container
    - currently downloading a folder will just result in an empty file
    - maybe I should just hide the download button for those two
- **TODO - Nice-to-have - Make the public URL for a public object clickable (it is just text right now)**
- **TODO - Nice-to-have - Reflect the full object path in the breadcrumb when an object is in detail view**
- **TODO - Nice-to-have - Remember what containers and folders were open and reopen them after a new container is added (enhancement)**
- **TODO - Nice-to-have - When a container or folder is clicked, expand/collapse it rather than forcing the arrow to be clicked (CS treeView enhancement)**
    - I do not like the current implementation in CS and I would like to change it
    - Currently: If you click on a parent item, it will just show it in details, but it will not expand to show the children and it feels very un-natural
    - I Want: When a parent item is clicked, it shows the details, but it ALSO expands/collapses the tree structure (so you don't have to click the tiny arrow as well)
- **TODO - Medium - A user guide to explain the setup required to get Swift setup with mauth as well as setting the load balancer in front of CS and Swift**
    - What is the best way to make this doc available?

## Use cases

For now, refer to the Feature Specifications or the UI Flow sections.

## Architecture and Design description

Most of this has already been covered.  I will add to this section as questions arise or if there is a request for more detailed descriptions of the different pieces.

- When a user logs in, a change was made to CS to ALSO pass the 'api key' to the client side, it is then saved in the user object.
    - This was added because mauth (the Swift middleware) authenticates to Swift with the CS 'username' and 'api key' (see curl example above).
    - It is highly recommended that you setup CS with SSL in front of it so the API requests are not in clear text (not safe against sniffers)
- It is required that a user has an API Key generated in order to use this service.
- A load balancer is put in front of CS (and Swift).  I have included details for getting this to work with HAproxy.
- mauth is required to enable authorization/authentication of CS users in Swift.

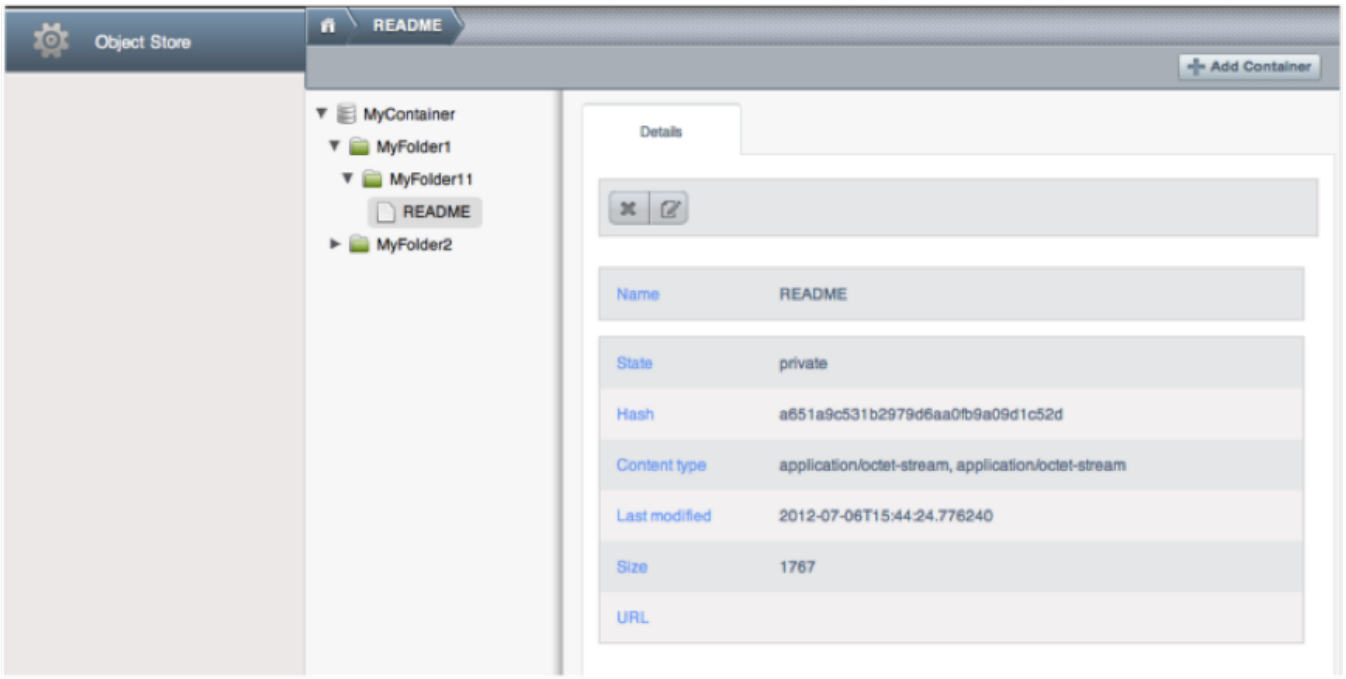
## Web Services APIs

No changes to the CloudStack APIs

## UI Flow

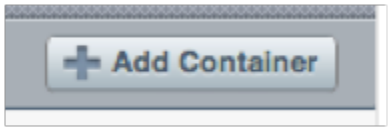
I will continue to add screenshots to this section as I have time...

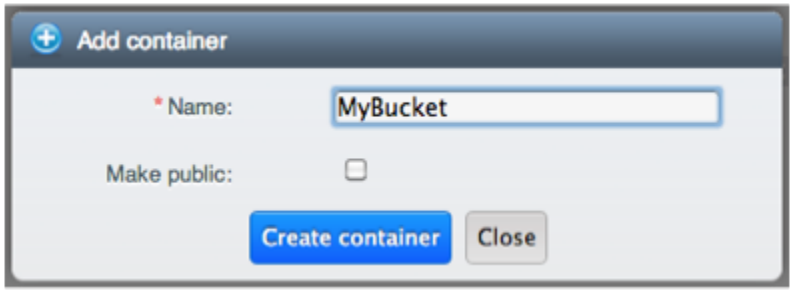
### Basic page view

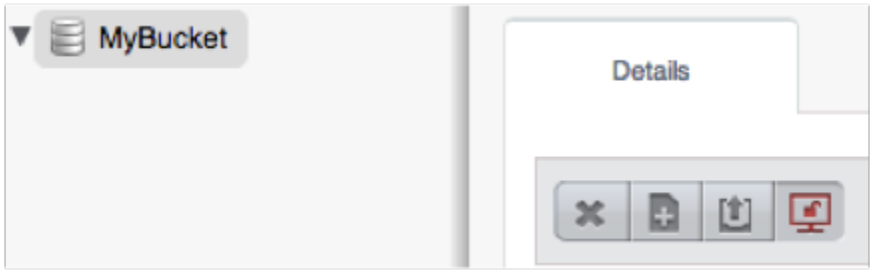


(Old Screenshot:  The Object Store menu item is actually added as the last item in the services menu, it is not the only item in the menu)
(Note: URL is empty because MyContainer is private in this case)

### Add Container

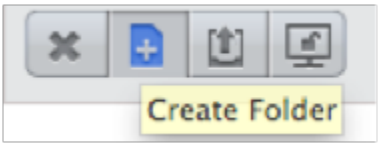


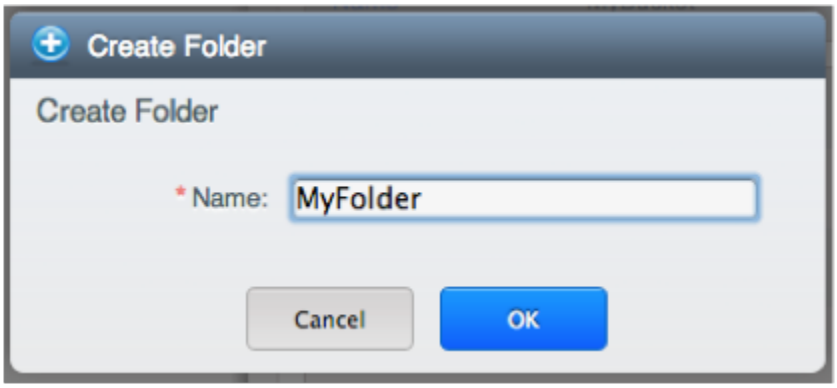



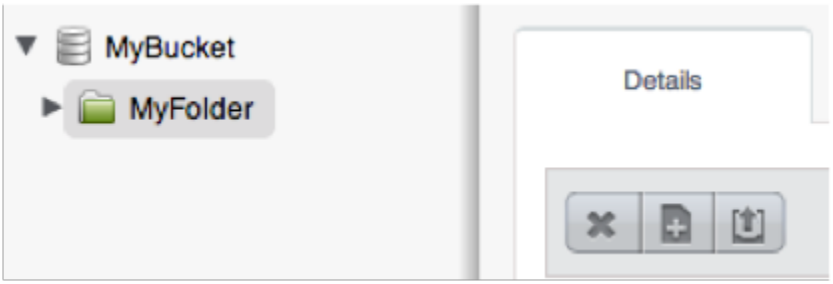



(options are:  `delete`, `add folder`, `upload object` and `make container (and all its contents) public`)

### Add Folder

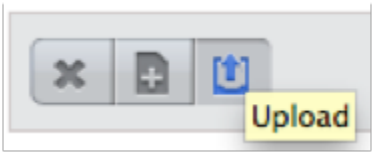


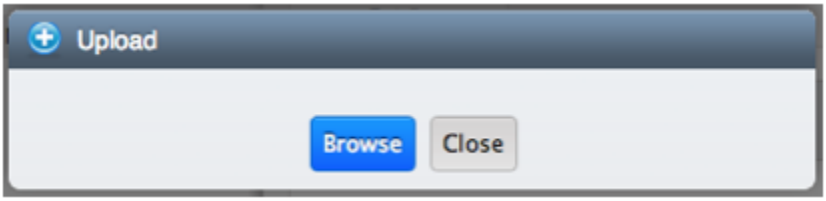



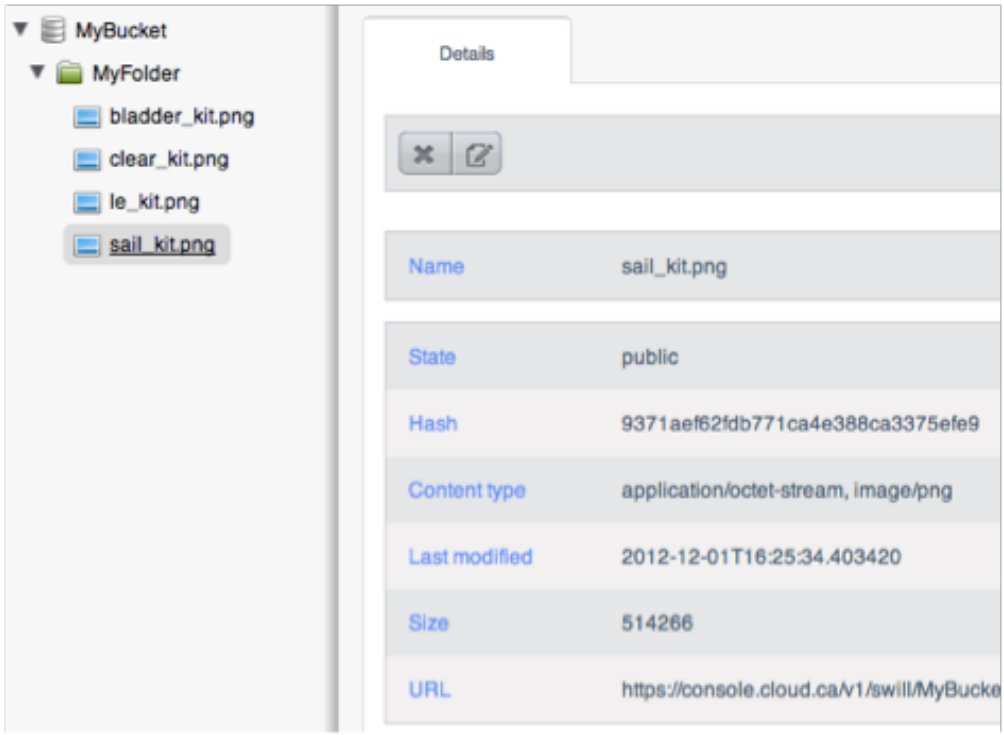




(options are: `delete`, `add sub-folder`, `upload object`)

**Add Object**




***NOTE:*** *Undocumented Step - Before this last screenshot the `make container public` action was applied to show the public URL.*

Additional details can be found in this presentation: swift_ui_with_cloudstack.pptx

## IP Clearance

- This implementation depends on both OpenStack Swift and the mauth middleware for Swift.
  - Both are available under the Apache 2 License.
- This implementation also adds PLupload as a client side file upload manager.

- PLupload is licenced as GPLv2.

## Appendix

- mauth middleware (git repository for the Swift auth middleware)
- swift_setup.png (overview image)
- haproxy_conf.txt (haproxy config)
- swift_ui_with_cloudstack.pptx (additional details from a presentation I did on the topic)
- Implements-OpenStack-Swift-as-an-Object-Storage-Service.patch (the patch against master on 2014/03/12)

**Note:** If you have followed this implementation at all (from previous conferences), you may have heard of 'cs_auth' being used as the Swift auth middleware. 'cs_auth' was the first Swift auth middleware I wrote to authenticate CS users in Swift and was the predecessor of 'mauth'. 'mauth' will continue to be supported, but 'cs_auth' will not and should not be used for future projects.

**Note: Regarding Swift Usage Reporting**

The Swift implementation is made up of two parts, the auth middleware in Swift and the AJAX CloudStack UI in the users browser. Since we are not integrating with the Java backend of CS at all, I have not implemented any Swift usage reporting in CS.

However, I have developed a middleware for Swift called 'swift_usage' (git repo) that exposes the Swift usage for accounts as a REST API. This was developed to be used with my auth middlewares, so that has been tested. I have not touched this project in quite a while and it was only used by a small group of us, so I need to do a once over on the project to make it 'open source' ready. I need to do a bit of a code cleanup and provide much better documentation, but the core is a working product already. We will be using this project internally, so expect some updates soon.

'swift_usage' depends on the 'slogging' Swift middleware. 'swift_logging' just exposes the usage data collected by 'slogging' as a REST API. 'slogging' is not an entirely obvious piece of software to work with and it creates many different logs. Here is a summary of some of my notes on 'slogging'.