Tutorial encswift

Contents

Chapter 1

Introduction

This tutorial explains how to correctly install and use the tool. The tool allows to protect all the objects included in several containers, with different options. It is possible to encrypt them only on the client side, protecting them against a honest but curious server, or to protect them with a double encryption, one layer applied on the client side and the other one on the server side.

1.1 Organization

The tool is organized in two main parts:

- **Client** - It is organized in:
  - encswift_client - The client including the API interface and all the core functions necessary to make available the encryption layers. It is available on gitHub at *https://github.com/danieleguttadoro/openstack_encswift/tree/master/proxy_server/enc_swift*.
  - SwiftBrowser - The graphical user interface which can be easily used to send the requests to the Swift server and interact with the encswiftclient tool. It is available on gitHub at *https://github.com/danieleguttadoro/openstack_encswift/tree/master/swiftbrowser_client*.
- **Server** - It is organized in:
  - devstack - The Vagrant virtual machine running the OpenStack environment, containing all the necessary modules. It is available on

gitHub at *https://github.com/danieleguttadoro/openstack_encswift/tree/master/devstack-vagrant-enc*.

- encswift_server - The Swift Storage Service conveniently modified to support the new protection. In particular, it includes two additional middleware using Paste Framework. It is available on gitHub at *https://github.com/danieleguttadoro/openstack_encswift/tree/master/encswift_server*.

- **Proxy server**:

  The proxy is used to map all the requests from the client to the server. This way, the client can be unchanged and the encryption tool is totally transparent. It is available on gitHub at *https://github.com/danieleguttadoro/openstack_encswift/tree/master/proxy_server*.

Chapter 2

Setup

This chapter describes how to start an OpenStack environment instance including all the new features.

2.1 Configuration

To start and correctly interact with the new Swift version, it is important to execute all the following steps:

- Copy locally the devstack folder and configure its environment:
  - Install vagrant & virtual box.
  - Configure a base /.vagrant.d/Vagrantfile to set your VM size. If you have enough horsepower you should make the file something like:

```
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do
   |config|
  config.vm.provider :virtualbox do |vb|
    # Use VBoxManage to customize the VM. For
       example to change memory:
    vb.customize ["modifyvm", :id, "--memory",
        "8192"]
    vb.customize ["modifyvm", :id, "--cpus",
       "4"]
  end
end
```

    The Vagrantfile contains also the network configuration, included the forwarded ports to change in case of some conflict.

  - Install the vagrant-hostmanager and vagrant-cachier plugins:

```
vagrant plugin install vagrant-hostmanager
vagrant plugin install vagrant-cachier
```

  - Check the configuration defined in the config.yaml file. You have to change at least the bridge int value, setting the interface name on which you want the devstack nodes to get bridged (the one you are connected to; use ifconfig and use exactly the name showed there).

- Start the virtual machine through the vagrant up manager com- mand (it may require a few minutes).
- Connect to the machine through the vagrant ssh manager com- mand.
- Change the access policy to obtain the user information.
  - Set a new rule in /etc/keystone/policy.json, changing it as follows:

```
{...
"any":"role:admin or role:Member",
...
"identity:get_user": "rule:any",
"identity:list_users": "rule:any",
...}
```

  - Delete the assert_admin() in /opt/stack/keystone/keystone/identity/controllers.py, in get_user, get_users and get_user_by_name functions. This way, each user can access the id of other ones.
- Restart the apache server, as follows:
  - sudo su root
  - service apache2 restart
  - exit

- Change the proxy server [pipeline:main] in /etc/swift/proxy-server.conf as follows, to include encrypt and key_master filter:

```
[pipeline:main]
pipeline = catch_errors gatekeeper healthcheck
   proxy-logging cache container_sync bulk tempurl
    ratelimit crossdomain authtoken keystoneauth
   tempauth formpost staticweb container-quotas
   account-quotas slo dlo proxy-logging encrypt
   key_master proxy-server

[filter:encrypt]
paste.filter_factory = swift.common.middleware.
   encrypt:filter_factory

[filter:key_master]
paste.filter_factory = swift.common.middleware.
   key_master:filter_factory
```

- Restart Swift and Keystone services running the following commands:
  - sudo su stack
  - script /dev/null
  - screen -r stack
  - Press Ctrl+A and number 2, to see the Key service.
  - Press Ctrl+C and re-execute the last command to restart the service.
  - Press Ctrl+A and number 3, to see the Key-Access service.
  - Press Ctrl+C and re-execute the last command to restart the service.
  - Press Ctrl+A and number 4, to see the Proxy-Server service.
  - Press Ctrl+C and re-execute the last command to restart the Proxy-Server service.

On the server side, there is now a complete version of Swift, including the new features. To complete the configuration, a few steps for the proxy server and the client have to be set.

- **Proxy Server**:

  1. Copy the Proxy server folder.
  2. Install flask, ecdsa, python-swiftclient and python-barbicanclient packages.
  3. Set the correct IP address, where the Swift server runs, changing AUTH_URL , BARBICAN_URL and STORAGE_URL in enc_swift/config.py
  4. Execute the proxy.py file to start the proxy

- **Client**:

  1. Copy locally the swift_browser client folder.
  2. Set the correct IP address, where the Proxy server runs, changing SWIFT_AUTH_URL , STORAGE_URL in swiftbrowser/settings.py and in swiftbrowser/config.py.
  3. Execute the make run command in swiftbrowser directory. The interface is reachable at address 127.0.0.1:8000.

2.2 Settings

2.2.1 Over-Encryption Settings

The configuration file (proxy_server/enc_swift/config.py) contains a variable (OVER_ENCRYPTION) set to True only when the user wants to perform Over-Encryption to manage a policy evolution. Any other value considers only a client side protection, downloading, re-encrypting and uploading all the files in case of policy evolution (in particular after a user revocation).

2.2.2 Users

To create an example user set, necessary to use the new encryption layers, you could follow these steps:

- Change the PATH variable in proxy_server/enc_swift/config.py inserting "."

- Execute the proxy_server/enc_swift/create_demo_users.py to create the users
- In Over_Encryption case, update the user swift, adding own keys on the server, at path /opt/stack/swift/swift/common/middleware, and adding those keys references into swift user description
- Change the PATH variable in proxy_server/enc_swift/config.py, inserting "proxy_server/enc_swift"

The user set creation, executed how described in the previous steps, makes available a list of users to interact with the Swift service. In particular they have all the necessary elements (public/private, master and signature keys) for creating containers, uploading/downloading objects and managing the ACL. The users available to test the functionalities, in project demo, are:

Username	**Password**
enctest1	enctest1
enctest2	enctest2
enctest3	enctest3
enctest4	enctest4
enctest5	enctest5

Table 2.1: Users

Chapter 3

How to use the tool

This chapter describes how to use the tool and, in particular, which steps are necessary to send the requests to the server. The images are referred to the graphical user interface Swiftbrowser.

3.1 Login

The login page makes possible to access the Swift service. The username, in project_name:username style to indicate which tenant connect to, and the password are required (Figure 3.1).

3.2 Container

Once a user has logged in, she can create a new container, through the dedicated form, reachable by the red button (Figure 3.2).

SwiftBrowser displays a list of containers included in the tenant. For each container is indicated the number of objects included in it and the total dimension (Figure 3.3).

For each container, it is possible to manage the Access Control List and to delete the container (Figure 3.4).

Sharing

The tool displays the list of authorized users, differentiating the authorizations between read-only and read/write (Figure 3.5).

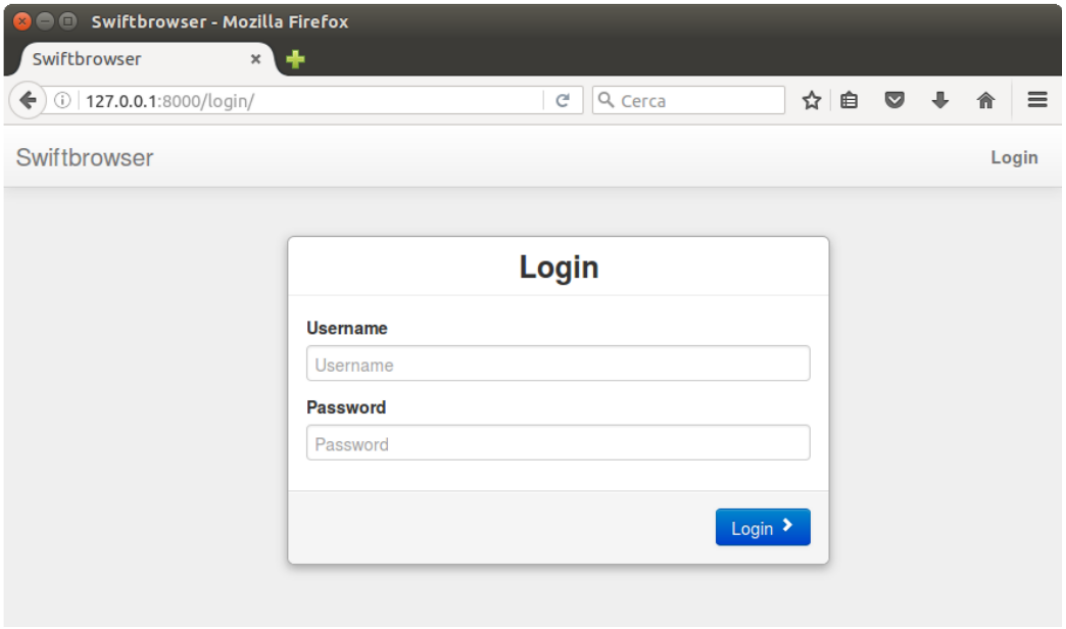

Figure 3.1: Login Page

Through the dedicated button it is possible to add a new user to the ACL or to make a container private/public (if an ACL has been defined or not) (Figure 3.6).

The add to acl button permits to add a new user, through a specific form, specifying the authorization type (read or read/write) to assign (Figure 3.7).

The second operation available is to make a container private/public.

- If a container is public - i.e., it is not protected by any new encryption layer, it is possible to make it private . From now, only the owner has the possibility to see and upload/download the objects. Obviously, after this operation, it is possible to enlarge the ACL to include other users.

- If a container is private - i.e., it is protected by a BEL key and, if necessary, a SEL key, it is possible to make it public, in order to delete any reference to the ACL since all the users have the possibility to access it. In particular all the objects are downloaded and re-uploaded in plain-text.

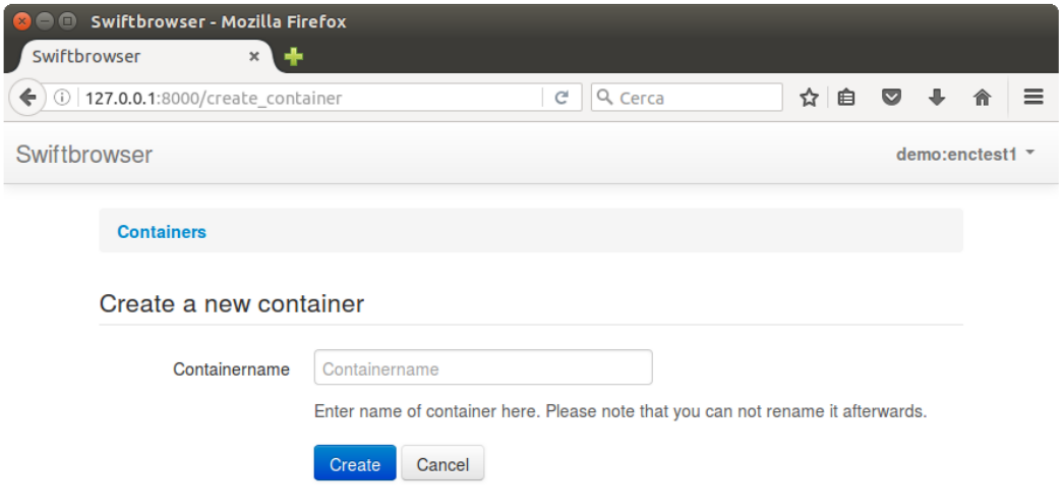

Figure 3.2: Create container form

3.3 Objects

Clicking on a specific container, SwiftBrowser displays its object list(Figure 3.8).

To download an object it is sufficient to click on the name of the object. The tool starts a procedure to download the clear version of the file, on which the it has applied the decryption functions to return the object requested by the client.

The upload of a new object can be performed clicking the red button. It is possible to select a new file among the files stored on the hard-disk of the client and upload it applying the Base Encryption Layer (Figure 3.9).

The red button permits also to create a pseudofolder inside the container through the dedicated form. It is shown in Figure 3.10.

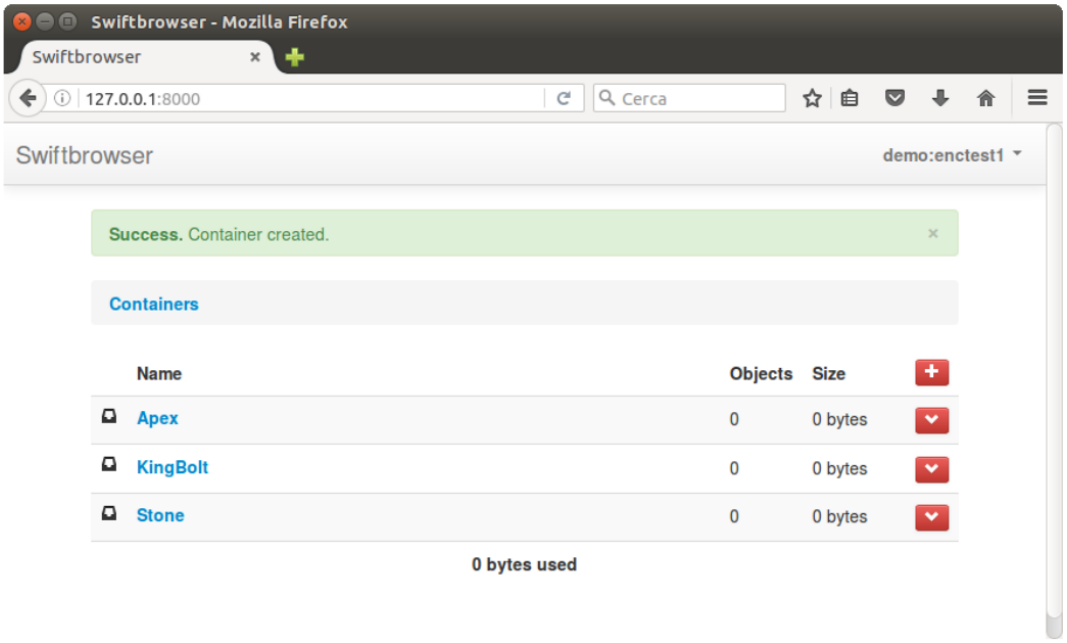

Figure 3.3: Container list

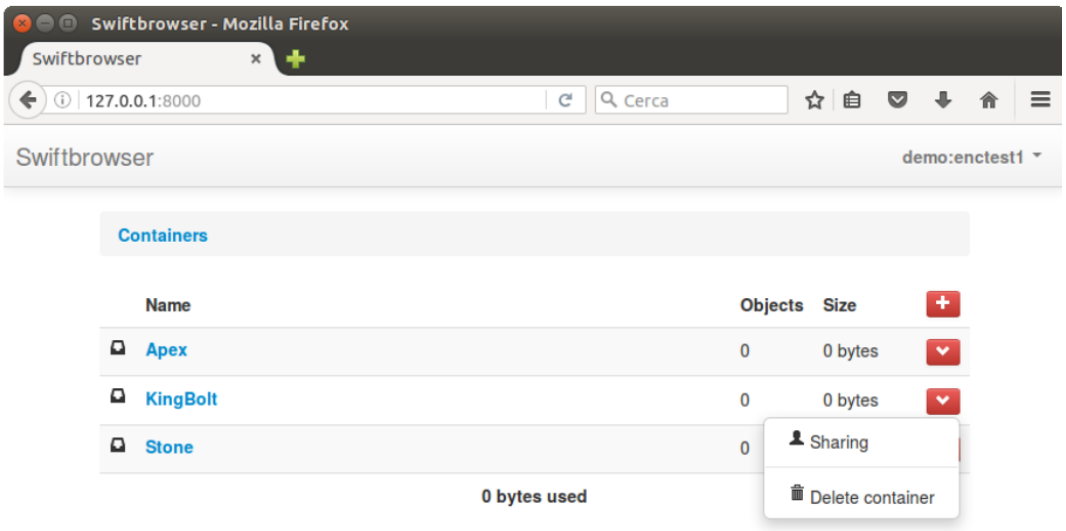

Figure 3.4: Container list (2)

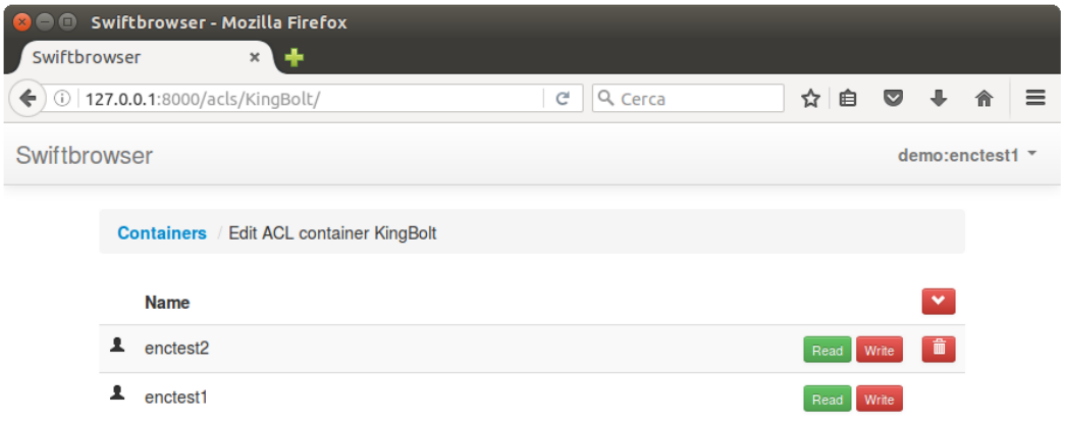

Figure 3.5: Access Control List associated to a container

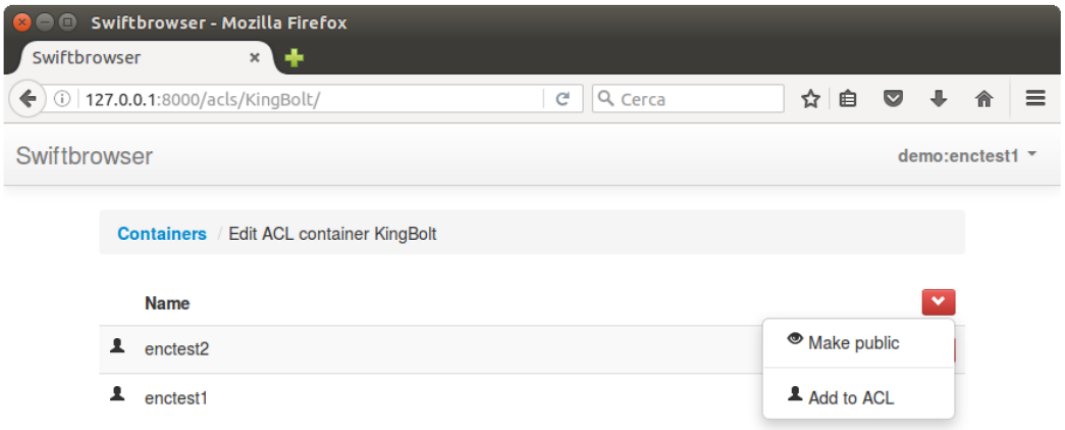

Figure 3.6: Access Control List associated to a container (2)

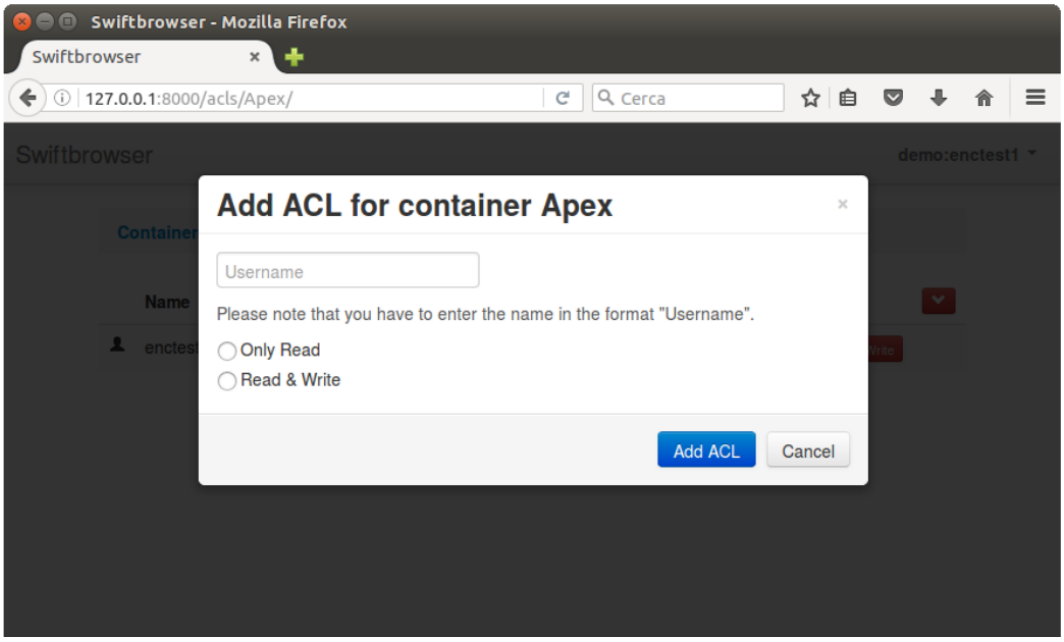

Figure 3.7: Form to introduce a new user in the ACL

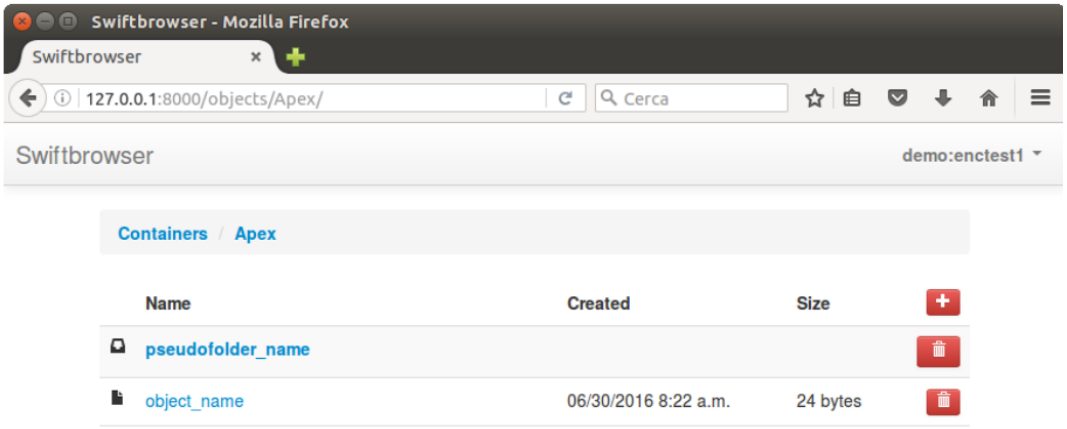

Figure 3.8: Objects included in a container

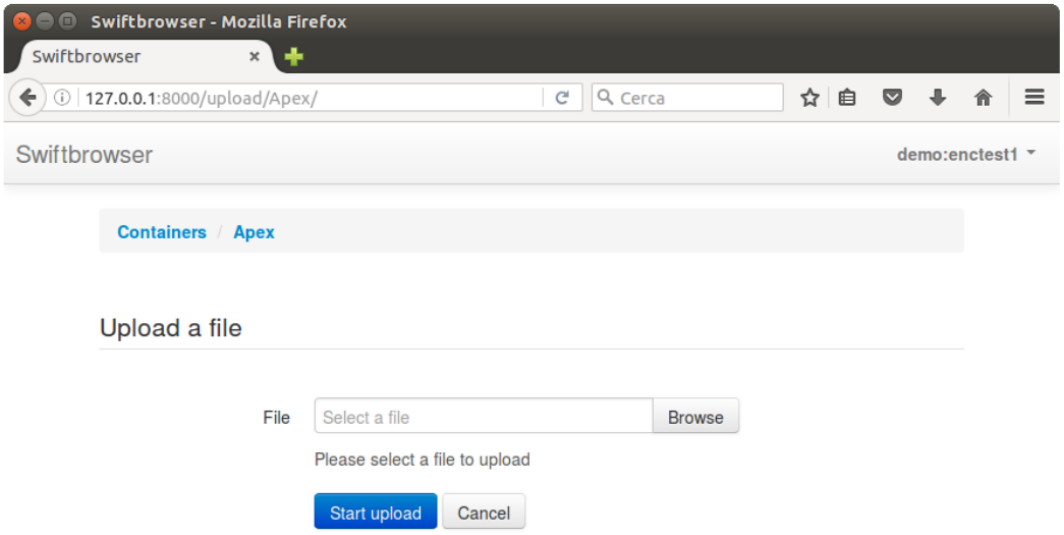

Figure 3.9: Upload form

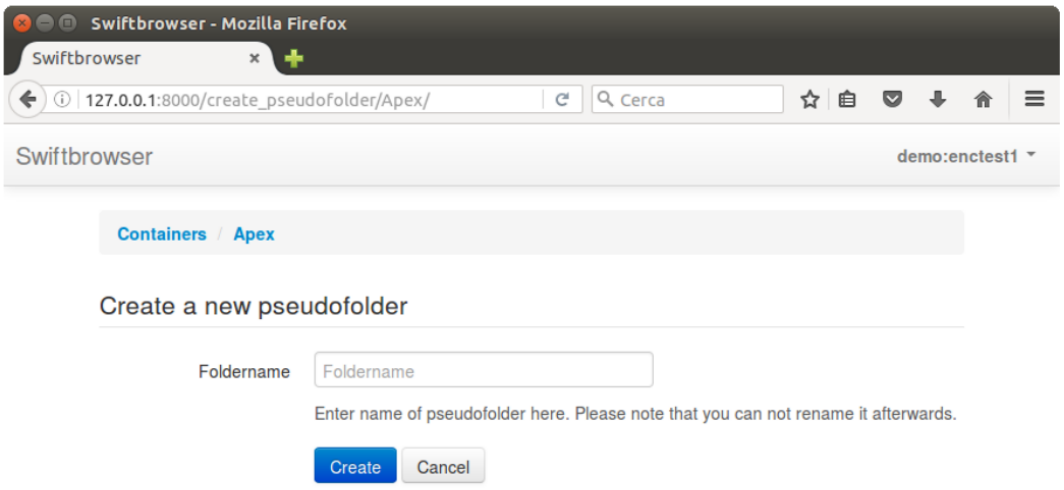

Figure 3.10: Create a new pseudo folder