Technical white paper

Programming to the StoreAll OpenStack Object Storage Interface

Table of contents

Introduction

With StoreAll OS v6.5, HP introduces an OpenStack based convergent storage platform incorporating both file and object interfaces. StoreAll OS v6.5 contains the Swift and Keystone projects from the OpenStack Grizzly release. These projects provide object storage and authentication services for the platform.

One of the important features of object storage is the simplified programmatic interface they present. While the interface is less complex than traditional file system interfaces, it is different. That difference may be perceived as complexity. This document seeks to alleviate the perceived complexity by providing step by step examples to developing a fully functional application able to interface directly with the StoreAll OS OpenStack object store.

This document is designed to be read from beginning to end. Each example builds on the information preceding it.

Finally, while the target audience is largely aimed at the software development community, this document may be used by managers and business decision makers to better understand what it takes to interact with an object store.

Languages

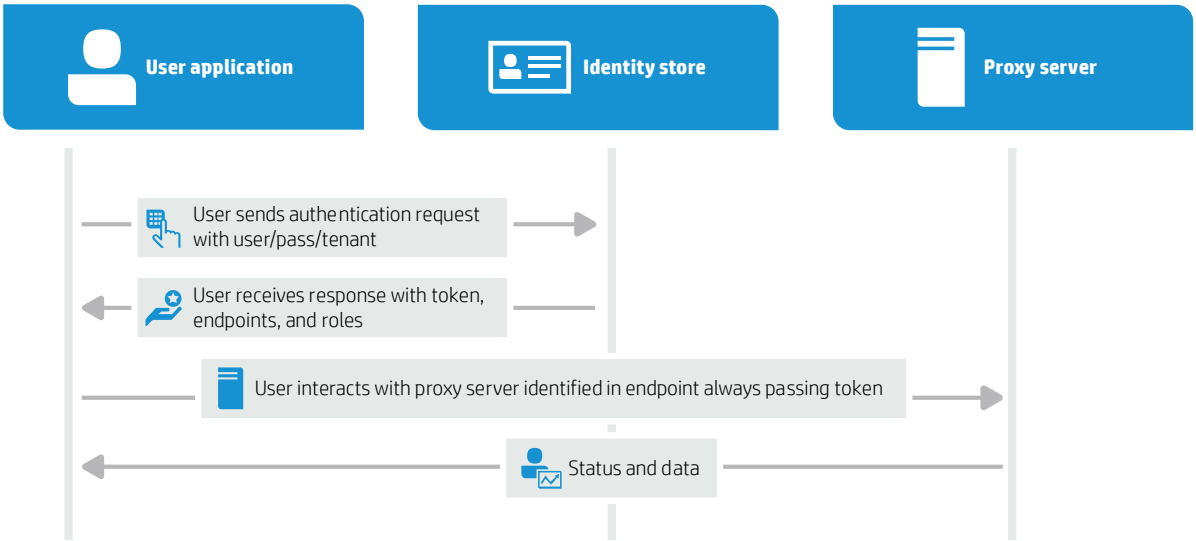
The code presented in the main body of text is written using C#. Each of the examples is explained in concept so that a pre-requisite knowledge of C# is not necessary to understand the programming concepts to communicate with the StoreAll OpenStack object store.

In addition, complete code examples in Java and PHP are presented in appendices at the end of the document. These examples closely follow the C# code examples in the main document.

Interacting with the object store

Before attempting your first object store application it is important to understand the basic flow that any application will follow. Prior to interacting with the object store, the application must first communicate with the identity store that is included in the StoreAll OS. As shown in figure 1, the application first sends user credentials, username, password, and the tenant that the user is associated with to the identity store.

Figure 1. Basic Program Flow



For a valid user, the identity store returns an access token containing, among other things, an authentication token and a list of endpoints that the user may access. With StoreAll OS these endpoints are the proxy servers for the object store. These endpoints will be used for all further communication with the object store.

The returned authentication token is sent with each subsequent command to the object store. This token is placed in the HTTP request header as the X-auth-token value. The authentication token is valid for 24 hours after which time the application must re-authenticate the user and receive a new token. Applications, such as the example in this paper, that are short-lived may authenticate the user each time the application is executed, a new token is issued at each authentication request.

The code

The examples in this document are designed to focus on the interaction with the object store and not on nuances in the code. As a result the code is written in such a way as to illustrate the interaction rather than to show optimal coding techniques.

The code example is made up of a main program that creates and uses a few objects (programming objects). The main program is really just a vehicle for accessing, primarily, two objects, an authentication object and an HTTP proxy. The classes are actually quite similar in the respect that each sends a request to a resource in the object store and each returns a response. Because of some unique treatment of the authentication process, they are written separately.

Authentication class

The authentication class show below builds an HTTP request object consisting of the URL of the identity store and the user credentials to be authenticated.

Items of note within the class include the authorization string that is sent to the identity store and data returned.

The authorization string is POSTed as data to the identity store and is not written as part of the headers as frequently is done with HTTP requests. The string takes the following parameters and may be formatted as Json or XML: the tenantName and the passwordCredentials made up of the username and password. In our example we kept things simple with the username “user1”, the password “pass1” and the tenant “tenant1”. This example uses Json formatting.

Note:
The user and tenant were configured in the StoreAll prior to developing this example code.

In our example the authorization string is as follows:

```
{
   “auth”:{
      “passwordCredentials”:{
         “username”:”user1”,
         “password”:”pass1”
      },
      “tenantName”:”tenant1”
   }
}
```

This data is written as part of the data stream associated with the POST command.

```
    class authentication
    {
        public authResponse authenticate(string user, string password, string tenant, string
url)
        {
            // Your system administrator should be able to provide you with the URL
            // of the identity service
            // By default this services listens on port 5000.
            System.Net.HttpWebRequest request = ((HttpWebRequest)WebRequest.Create(url));

            // this example is working with Json formatted data
            request.ContentType = “application/json”;
            request.Method = “POST”;

            // To authenticate the user a string containing the username, password
            // and tenant is sent to the identity service
            string authString = “{\”auth\”:{“
                + “\”passwordCredentials\”:{“
                + “\”username\”:\””+ user + “\”,”
                + “\”password\”:\”“ + password+”\””
                + “},\”tenantName\”:\””+tenant+”\”}}”;

            // The string is converted to a byte array in order to be sent using the
```

```
            // C# Stream.write() method
            ASCIIEncoding encoding = new ASCIIEncoding();
            byte[] authBytes = encoding.GetBytes(authString);
            using (Stream stream = request.GetRequestStream())
            {
                stream.Write(authBytes, 0, authBytes.Length);
            }

            authToken authenticationToken = null;

            // A POST command containing the string
            //
{“auth”:{“passwordCredentials”:{“username”:”user1”,”password”:”pass1”},”tenantName”:”tenant1”
}}
            // is sent to the resource http://10.32.19.165:5000/v2.0/tokens

            WebResponse response = request.GetResponse();
            if (response != null)
            {
                string jsonResponse = null;

                // The server will respond, to a properly authenticated user, with a string
                // containing Json formatted data representing the access token for the user
                using (var sr = new StreamReader(response.GetResponseStream()))
                    jsonResponse = sr.ReadToEnd();

                if (jsonResponse != null)
                {
                    // This application uses Newtonsoft Json.NET to deserialize the
                    // json string into a C# object.
                    authenticationToken =
JsonConvert.DeserializeObject<authToken>(jsonResponse);
                }
            }

            if (authenticationToken == null)
            {
                Console.WriteLine(“Unable to authenticate user”);
                return null;;
            }

            // The access token contains, among other items, two fields necessary for
            // all further access to the object store, the X-auth-token and URL
            // of the proxy server where all other requests will be made
            authResponse authResponseObj = new authResponse();
            authResponseObj.X_auth_token = authenticationToken.access.token.id;
            foreach (authToken.ServiceCatalog sc in
authenticationToken.access.serviceCatalog)
            {
                foreach (authToken.Endpoint ep in sc.endpoints)
                {
                    if (ep.publicURL.Contains(“:8888”))
                    {
                        authResponseObj.ObjectStoreURL = ep.publicURL;
                    }
                }
            }
            return authResponseObj;
        }
    }
    class authResponse
    {
        string x auth token;
        string objectStoreURL;

        public string X_auth_token { get { return x_auth_token; } set { x_auth_token = value;
} }
        public string ObjectStoreURL { get { return objectStoreURL; } set { objectStoreURL =
value;}}
    }
```

In response to the authorization string the server returns an XML or Json formatted access token containing a number of values including information about the user and their roles. It also contains a token that will be used for all subsequent requests in the HTTP header X-auth-token.

The following access token is in Json format. The parameters that are of interest in this document are highlighted in blue.

The value stored at access.token.id is used with the X-auth-token.

Endpoints for the object store are stored in access.serviceCatalog.endPoints. By default StoreAll OS assigns the port value of 8888 to the object store endpoints.


```
{
   "access":{
      "token":{
         "expires":"2014-01-04T22:55:57Z",
         "id":"951584bbc13247cba9eb501e5f4868e1",
         "tenant":{
            "enabled":true,
            "id":"7b9a902423a582c9eda266dcf3ad697473909b181af608c0497bda5fa37ce321",
            "name":"tenant1"
         }
      },
      "serviceCatalog":[
         {
            "endpoints":[
               {
                  "adminURL":"http://10.32.19.10:8888/",
                  "region":"RegionOne",
                  "internalURL":
"http://10.32.19.10:8888/v1/AUTH_7b9a902423a582c9eda266dcf3ad697473909b181af608c0497bda5fa37ce321",
                  "id":"31e7fb8cf4284b228d55171abd43409c",
                  "publicURL":
"http://10.32.19.10:8888/v1/AUTH_7b9a902423a582c9eda266dcf3ad697473909b181af608c0497bda5fa37ce321"
               }
            ],
            "endpoints_links":[

            ],
            "type":"object-store",
            "name":"swift"
         }
      ],
      "user":{
         "username":"user1",
         "roles_links":[
```

```
            ],
            "id":"324c2ae86fff69d22629320cdf589f417b9a902423a582c9eda266dcf3ad6974",
            "roles":[
               {
                  "name":"admin"
               }
            ],
            "name":"user1"
         },
         "metadata":{
            "is_admin":true,
            "roles":[
               "7b9a902423a582c9eda266dcf3ad69740455f3e86019620d40d7a3adcc3b2dc4"
            ]
         }
      }
}
```

This example uses a popular third-party library from Newtonsoft called Json.NET[1] to deserialize the Json string to a C# object that is likely to be more useful within an application that the string itself. The class used to deserialize the object is shown later in the document.

HTTP proxy class

Because all of the operations associated with the object store are done to the proxy server a class named httpProxy is shown providing the basic HTTP interaction for the application to the server. This class is used to perform all of the HTTP interactions with the object store in this document. The class has a single method called AccessResource that used by the application to access the various object store resources. The AccessResource method takes a proxyResource object as input. The proxyResource class is just below the httpProxy class.

As mentioned earlier each command sent to a resource in the object store must have the X-Auth-Token header present. Commands sent without a valid X-Auth-Token return an "unauthorized" error.

The accept header may take "text/plain", "application/xml" and "application/json" as values. The examples in this document use "application/json".

The method varies by the operation desired. For example to get something the "GET" verb is used and to create something the "PUT" verb is used.

In a few cases some special handling is inserted to make sure the examples work. For example when creating an object we have inserted code to write a string to the object. Special cases have also been inserted to handle the creation and reading of object metadata.

[1] newtonsoft.com/json

The basic construct of the method is to build the HTTP request, send the request and then get the response.

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.IO;


namespace example

{
    class httpProxy

    {
        public string AccessResource(proxyResource resourceObj)

        {
            string responseString = "";


            System.Net.HttpWebRequest request =
((HttpWebRequest)WebRequest.Create(resourceObj.Url));

            request.Headers.Add("X-Auth-Token:" + resourceObj.XauthToken);

            request.Accept = resourceObj.Accept;

            request.Method = resourceObj.Verb;


            // check to see if this is to upload object

            if(resourceObj.Verb.Equals("PUT"))

            {
                // if so, write the data

                UTF8Encoding encoding = new UTF8Encoding();

                byte [] buffer = encoding.GetBytes("This is the data in my object");

                Stream dataStream = request.GetRequestStream();

                dataStream.Write(buffer, 0, buffer.Length);

                dataStream.Close();

            }


            // check to see if this is writing metadata

            if (resourceObj.Verb.Equals("POST") && !resourceObj.Metadata.Equals(""))

            {
                string [] nameValuePairs = resourceObj.Metadata.Split(new char[] { ',' });

                foreach (string pair in nameValuePairs)

                {
                    request.Headers.Add("X-Object-Meta-" + pair);

                }
            }
```

```
                WebResponse response = null;
                response = request.GetResponse();

                if (response != null)
                {
                    Stream dataFromSite = response.GetResponseStream();

                    var sr = new StreamReader(dataFromSite);
                    responseString = sr.ReadToEnd();
                }

                // check to see if this is a request to read metadata
                if (resourceObj.Verb.Equals("HEAD"))
                {
                    responseString += Environment.NewLine;
                    foreach (string key in response.Headers.AllKeys)
                    {
                        string[] headerValues = response.Headers.GetValues(key);
                        foreach(string headerValue in headerValues)
                        {
                            responseString += key + ":" + headerValue + Environment.NewLine;
                        }
                    }
                }
                return responseString;
            }
        }
        class proxyResource
        {
            string verb;
            string url;
            string XauthToken;
            string accept;
            string metadata;

            public string Verb { get { return verb; } set { verb = value; } }
            public string Url { get { return url; } set { url = value; } }
            public string XauthToken { get { return XauthToken; } set { XauthToken = value; } }
            public string Accept { get { return accept; } set { accept = value; } }
            public string Metadata { get { return metadata; } set { metadata = value; } }
        }
    }
```

Authenticating the user

The remainder of the code examples are based on a very basic “main” program that is really nothing more than a vehicle to call the authentication and httpProxy methods.

Before doing operations on any object store resource, the application must first authenticate with the identity store. Your system administrator should be able to provide you with the address of the identity store and a set of valid user credentials to be used.

By default the StoreAll OS assigns the port value of 5000 to the public address of the identity store.

```
using System;

using System.Collections.Generic;

using System.Text;

using System.Net;

using Newtonsoft.Json;

using System.IO;


namespace example

{

    class Program

    {

        static void Main(string[] args)

        {

            // authenticate user

            authentication authenticationObj = new authentication();

            authResponse authenticationResponseObj = authenticationObj.authenticate(“user1”,
“pass1”, “tenant1”, “http://10.32.19.165:5000/v2.0/tokens”);

            Console.WriteLine(“X-Auth-Token to use is ” +
authenticationResponseObj.X_auth_token);

            Console.WriteLine(“Selected proxy server is ” +
authenticationResponseObj.ObjectStoreURL);


            if (authenticationResponseObj.X_auth_token.Equals(“”))

            {

                // user failed to authenticate

                return;

            }


        }

    }

}
```

The code creates an authentication object, described earlier, and calls the authenticate method passing the user credentials, the tenant and the address of the identity store.

In response to a valid user request the identity store returns an access token. The X-Auth_Token and object store endpoint are saved for easy reuse in later examples.

The program displays

X-Auth-Token to use is 3e375509f6d44dd1b96cdda18440b6bc

Selected proxy server is
http://10.32.19.10:8888/v1/AUTH_7b9a902423a582c9eda266dcf3ad697473909b181af608c0497bda5fa37ce321

Creating containers

The remaining code fragments may be used in conjunction with the authentication code from the previous example. Complete program listings are shown later in the document.

Each of the subsequent examples use an httpProxy object called proxyObj calling the httpProxy.AccessResource method with a proxyResource called resourceObj. The resourceObj contains a collection of values used by the AccessResource method to process the request.

To create a container the container name with a leading forward slash "/" is appended to the endpoint returned by the identity store. The PUT verb is used to communicate with the object store that the container is being created.

In this case the URL looks like
http://10.32.19.10:8888/v1/AUTH_7b9a902423a582c9eda266dcf3ad697473909b181af608c0497bda5fa37ce321/myContainer

```
// create container

resourceObj.Url = authenticationResponseObj.ObjectStoreURL + "/myContainer";

resourceObj.XauthToken = authenticationResponseObj.X_auth_token;

resourceObj.Verb = "PUT";

resourceObj.Accept = "application/json";

resourceObj.Metadata = "";

response = proxyObj.AccessResource(resourceObj);

Console.WriteLine("Create container: " + response);
```

Listing containers

Containers are associated with the tenant. In order to list the containers associated with the tenant the application performs a "GET" using the endpoint returned by the identity store. The endpoint represents the tenant resource.

```
// list containers

httpProxy proxyObj = new httpProxy();

proxyResource resourceObj = new proxyResource();

resourceObj.Url = authenticationResponseObj.ObjectStoreURL;

resourceObj.XauthToken = authenticationResponseObj.X_auth_token;

resourceObj.Verb = "GET";

resourceObj.Accept = "application/json";

resourceObj.Metadata = "";

string response = proxyObj.AccessResource(resourceObj);

Console.WriteLine("List containers: " + response);
```

As mentioned earlier data may be formatted three ways, as XML, Json, or plain text. What follows are examples of the same data represented in each of these formats.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<account name="AUTH_7b9a902423a582c9eda266dcf3ad697473909b181af608c0497bda5fa37c
e321">
   <container>
      <name>myContainer</name>
      <count>0</count>
      <bytes>0</bytes>
   </container>
</account>
```

Json

```
[
  {
    "count": 0,
    "bytes": 0,
    "name": "myContainer"
  }
]
```

Plain text

```
myContainer
```

Creating objects

Objects must be created within a container. Attempting to create an object in the endpoint URL will result in a container being created with the intended object name used as the container name. Object resources are identified by appending the container name and the name of the object to the endpoint returned by the identity store. The container and object name are each preceded with a forward slash "/". In this example "/myContainer/object1" are appended to the endpoint to create "object1" within "myContainer".

The PUT verb is used to create the resource.

In the httpProxy class additional code is added to populate the object with the string "This is the data in my object."

```
            // create an object
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "PUT";
            resourceObj.Accept = "application/json";
            resourceObj.Size = 10;
            resourceObj.Metadata = "";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("Create object: " + response);
```

List the contents of a container

To list the contents of a container simply "GET" the container resource. Just as when the container was created, the container name with a leading forward slash "/" is appended to the endpoint.

```
            // list contents of a container

            resourceObj.Url = authenticationResponseObj.ObjectStoreURL + "/myContainer/";

            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;

            resourceObj.Verb = "GET";

            resourceObj.Accept = "application/json";

            resourceObj.Size = -1;

            resourceObj.Metadata = "";

            response = proxyObj.AccessResource(resourceObj);

            Console.WriteLine("List objects in container: " + response);
```

The object store returns information on each object stored in the container. The Json formatted text is as follows:

```
[
   {
      "hash":"7b0a002987665da5f2a4c480ae865e33",
      "last_modified":"2014-01-07T23:36:51.980940",
      "bytes":29,
      "name":"object1",
      "content_type":"application/octet-stream"
   }
]
```

An array of objects is returned. Each object has associated with it an MD5 hash of the object contents, the date last modified, the size of the object in bytes, the name of the object and the type of data as best determined by the object store.

Third-party libraries such as Json.NET may be used to convert the Json string to a program usable object.

Reading an object

Much like reading the contents of the container, reading an object uses the "GET" verb on the object resource. The object resource is identified just as it was when it was created by appending the container and object names to the endpoint provided by the identity store. As always the container and object names are preceded with a leading forward slash. In our example the object resource is identified as "/myContainer/object1"

```
            // read an object

            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";

            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;

            resourceObj.Verb = "GET";

            resourceObj.Accept = "application/json";

            resourceObj.Metadata = "";

            response = proxyObj.AccessResource(resourceObj);

            Console.WriteLine("Read object: " + response);
```

The contents of the object are read and presented showing

```
Read object: This is the data in my object
```

Writing object metadata

The object store allows applications to store custom metadata with container and object resources. Metadata is stored as name:value pairs and is accessed through the HTTP headers. In this example two pairs of metadata are written to the object resource "/myContainer/object1", "name1:value1" and "name2:value2". The "POST" verb is used to associate metadata with the resource.

```
            // write object metadata

            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";

            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;

            resourceObj.Verb = "POST";

            resourceObj.Accept = "application/json";

            resourceObj.Size = -1;

            resourceObj.Metadata = "name1:value1,name2:value2";

            response = proxyObj.AccessResource(resourceObj);

            Console.WriteLine("Write object metadata: " + response);
```

The httpProxy class contains code to add the metadata to the HTTP headers.

```
            // check to see if this is writing metadata

            if (resourceObj.Verb.Equals("POST") && !resourceObj.Metadata.Equals(""))

            {

                string [] nameValuePairs = resourceObj.Metadata.Split(new char[] { ',' });

                foreach (string pair in nameValuePairs)

                {

                    request.Headers.Add("X-Object-Meta-" + pair);

                }

            }
```

Each custom metadata pair is identified by the leading "X-Object-Meta-" string, so that the name1:value one pair is actually stored as "X-Object-Meta-name1:value1"

POSTing new metadata values to a resource replaces previously written metadata. If the application intends to write "additional" metadata, the existing metadata must first be read and then rewritten along with the new metadata.

Listing object metadata

Retrieving previously stored is accomplished by using the HEAD verb on the resource. In this case the example is reading the metadata stored with the resource "/myContainer/object1"

```
            // list object metadata

            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";

            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;

            resourceObj.Verb = "HEAD";

            resourceObj.Accept = "application/json";

            resourceObj.Size = -1;

            resourceObj.Metadata = "";

            response = proxyObj.AccessResource(resourceObj);

            Console.WriteLine("List object metadata: " + response);
```

Metadata is returned in the HTTP headers just as it was created. The httpProxy class has the following code to extract data from the HTTP headers

```
// check to see if this is a request to read metadata
if (resourceObj.Verb.Equals("HEAD"))
{
    responseString += Environment.NewLine;
    foreach (string key in response.Headers.AllKeys)
    {
        string[] headerValues = response.Headers.GetValues(key);
        foreach(string headerValue in headerValues)
        {
            responseString += key + ":" + headerValue + Environment.NewLine;
        }
    }
}
```

Resulting in program output that is as follows:

```
List object metadata:
X-Object-Meta-Name1:value1
X-Object-Meta-Name2:value2
X-Timestamp:1389137998.97341
Accept-Ranges:bytes
Content-Length:29
Content-Type:application/octet-stream
Date:Tue, 07 Jan 2014 23:40:06 GMT
ETag:7b0a002987665da5f2a4c480ae865e33
Last-Modified:Tue, 07 Jan 2014 23:39:58 GMT
```

Custom metadata is identified by the string "X-Object-Meta-" preceding the metadata values.

Deleting an object

Objects are deleted using the "DELETE" verb against the object resource, in this case "/myContainer/object1."

```
// delete object
resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
resourceObj.Verb = "DELETE";
resourceObj.Accept = "application/json";
resourceObj.Size = -1;
resourceObj.Metadata = "";
response = proxyObj.AccessResource(resourceObj);
Console.WriteLine("Delete object: " + response);
```

Deleting a container

Containers are deleted using the "DELETE" verb against the container resource, in this case "/myContainer."

```
// delete container

resourceObj.Url = authenticationResponseObj.ObjectStoreURL + "/myContainer";

resourceObj.XauthToken = authenticationResponseObj.X_auth_token;

resourceObj.Verb = "DELETE";

resourceObj.Accept = "application/json";

resourceObj.Size = -1;

resourceObj.Metadata = "";

response = proxyObj.AccessResource(resourceObj);

Console.WriteLine("Delete container: " + response);
```

Supporting classes

Program

```
using System;

using System.Collections.Generic;

using System.Text;

using System.Net;

using Newtonsoft.Json;

using System.IO;


namespace example

{

    class Program

    {

        static void Main(string[] args)

        {

            // authenticate user

            authentication authenticationObj = new authentication();

            authResponse authenticationResponseObj = authenticationObj.authenticate("user1",
"pass1", "tenant1", "http://10.32.19.165:5000/v2.0/tokens");

            Console.WriteLine("X-Auth-Token to use is " +
authenticationResponseObj.X_auth_token);

            Console.WriteLine("Selected proxy server is " +
authenticationResponseObj.ObjectStoreURL);


            if (authenticationResponseObj.X_auth_token.Equals(""))

            {

                // user failed to authenticate

                return;

            }
```

```
            // list containers
            httpProxy proxyObj = new httpProxy();
            proxyResource resourceObj = new proxyResource();
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL;
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "GET";
            resourceObj.Accept = "application/json";
            //resourceObj.Accept = "application/xml";
            //resourceObj.Accept = "text/plain";
            resourceObj.Metadata = "";
            string response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("List containers: " + response);

            // create container
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL + "/myContainer";
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "PUT";
            resourceObj.Accept = "application/json";
            resourceObj.Metadata = "";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("Create container: " + response);

            // create an object
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "PUT";
            resourceObj.Accept = "application/json";
            resourceObj.Metadata = "";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("Create object: " + response);

            // read an object
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "GET";
            resourceObj.Accept = "application/json";
            resourceObj.Metadata = "";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("Read object: " + response);
```

```
            // list contents of a container
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL + "/myContainer/";
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "GET";
            resourceObj.Accept = "application/json";
            resourceObj.Metadata = "";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("List objects in container: " + response);

            // write object metadata
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "POST";
            resourceObj.Accept = "application/json";
            resourceObj.Metadata = "name1:value1,name2:value2";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("Write object metadata: " + response);

            // list object metadata
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "HEAD";
            resourceObj.Accept = "application/json";
            resourceObj.Metadata = "";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("List object metadata: " + response);

            // delete object
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL +
"/myContainer/object1";
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "DELETE";
            resourceObj.Accept = "application/json";
            resourceObj.Metadata = "";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("Delete object: " + response);

            // delete container
            resourceObj.Url = authenticationResponseObj.ObjectStoreURL + "/myContainer/";
```

```
            resourceObj.XauthToken = authenticationResponseObj.X_auth_token;
            resourceObj.Verb = "DELETE";
            resourceObj.Accept = "application/json";
            resourceObj.Metadata = "";
            response = proxyObj.AccessResource(resourceObj);
            Console.WriteLine("Delete container: " + response);
        }
    }
}
```

Program output

```
X-Auth-Token to use is f50387beaca44b4783298022fae65c78

Selected proxy server is
http://10.32.19.10:8888/v1/AUTH_7b9a902423a582c9eda266dcf3ad697473909b181af608c0497bda5fa37ce321

List containers: [{"count": 0, "bytes": 0, "name": "myContainer"}]

Create container: <html><h1>Accepted</h1><p>The request is accepted for
processing.</p></html>

Create object:

Read object: This is the data in my object

List objects in container: [{"hash": "7b0a002987665da5f2a4c480ae865e33", "last_modified":
"2014-01-07T23:36:51.980940", "bytes": 29, "name": "object1", "content_type":
"application/octet-stream"}]

Write object metadata: <html><h1>Accepted</h1><p>The request is accepted for
processing.</p></html>

List object metadata:

X-Object-Meta-Name1:value1

X-Object-Meta-Name2:value2

X-Timestamp:1389137998.97341

Accept-Ranges:bytes

Content-Length:29

Content-Type:application/octet-stream

Date:Tue, 07 Jan 2014 23:40:06 GMT

ETag:7b0a002987665da5f2a4c480ae865e33

Last-Modified:Tue, 07 Jan 2014 23:39:58 GMT


Delete object:

Delete container:
```

authToken

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace example
{
    class authToken
    {
        // class representing the access token returned from the identity service
        public Access access { get; set; }

        public class Tenant
        {
            public bool enabled { get; set; }
            public string id { get; set; }
            public string name { get; set; }
        }

        public class Token
        {
            public string expires { get; set; }
            public string id { get; set; }
            public Tenant tenant { get; set; }
        }

        public class Endpoint
        {
            public string adminURL { get; set; }
            public string region { get; set; }
            public string internalURL { get; set; }
            public string id { get; set; }
            public string publicURL { get; set; }
        }

        public class ServiceCatalog
        {
            public List<Endpoint> endpoints { get; set; }
```

```
            public List<object> endpoints_links { get; set; }
            public string type { get; set; }
            public string name { get; set; }
        }

        public class Role
        {
            public string name { get; set; }
        }

        public class User
        {
            public string username { get; set; }
            public List<object> roles_links { get; set; }
            public string id { get; set; }
            public List<Role> roles { get; set; }
            public string name { get; set; }
        }

        public class Metadata
        {
            public bool is_admin { get; set; }
            public List<string> roles { get; set; }
        }

        public class Access
        {
            public Token token { get; set; }
            public List<ServiceCatalog> serviceCatalog { get; set; }
            public User user { get; set; }
            public Metadata metadata { get; set; }
        }
    }
}
```

Authentication

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using Newtonsoft.Json;
using System.IO;

namespace example
{
    class authentication
    {
        public authentication()
        {
        }
        public authResponse authenticate(string user, string password, string tenant, string
url)
        {
            // Your system administrator should be able to provide you with the URL of the
identity service
            // By default this services listens on port 5000.
            System.Net.HttpWebRequest request = ((HttpWebRequest)WebRequest.Create(url));

            // this example is working with Json formatted data
            request.ContentType = "application/json";
            request.Method = "POST";

            // To authenticate the user a string containing the username, password
            // and tenant is sent to the identity service
            string authString = "{\"auth\":{"
                + "\"passwordCredentials\":{"
                + "\"username\":\""+ user + "\","
                + "\"password\":\"" + password+"\""
                + "},\"tenantName\":\""+tenant+"\"}}";

            // The string is converted to a byte array in order to be sent using the
            // C# Stream.write() method
            ASCIIEncoding encoding = new ASCIIEncoding();
            byte[] authBytes = encoding.GetBytes(authString);
```

```
            using (Stream stream = request.GetRequestStream())
            {
                stream.Write(authBytes, 0, authBytes.Length);
            }

            authToken authenticationToken = null;

            // A POST command containing the string
            //
{“auth”:{“passwordCredentials”:{“username”:”user1”,”password”:”pass1”},”tenantName”:”tenant1”
}}
            // is sent to the resource http://10.32.19.165:5000/v2.0/tokens

            WebResponse response = request.GetResponse();
            if (response != null)
            {
                string jsonResponse = null;

                // The server will respond, to a properly authenticated user, with a string
                // containing Json formatted data representing the access token for the user
                using (var sr = new StreamReader(response.GetResponseStream()))
                    jsonResponse = sr.ReadToEnd();

                if (jsonResponse != null)
                {
                    // This application uses Newtonsoft Json.NET to deserialize the
                    // json string into a C# object.
                    authenticationToken =
JsonConvert.DeserializeObject<authToken>(jsonResponse);
                }
            }

            if (authenticationToken == null)
            {
                Console.WriteLine(“Unable to authenticate user”);
                return null;;
            }

            // The access token contains, among other items, two fields necessary for
            // all further access to the object store, the X-auth-token and URL
            // of the proxy server where all other requests will be made
```

```
            authResponse authResponseObj = new authResponse();
            authResponseObj.X_auth_token = authenticationToken.access.token.id;
            foreach (authToken.ServiceCatalog sc in
authenticationToken.access.serviceCatalog)
            {
                foreach (authToken.Endpoint ep in sc.endpoints)
                {
                    if (ep.publicURL.Contains(":8888"))
                    {
                        authResponseObj.ObjectStoreURL = ep.publicURL;
                    }
                }
            }
            return authResponseObj;
        }
    }
}
```

authResponse

```
    class authResponse
    {
        string x_auth_token;
        string objectStoreURL;

        public string X_auth_token { get { return x_auth_token; } set { x_auth_token = value;
} }
        public string ObjectStoreURL { get { return objectStoreURL; } set { objectStoreURL =
value;}}
    }
```

httpProxy

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.IO;
```

```
namespace example
{
    class httpProxy
    {
        public string AccessResource(proxyResource resourceObj)
        {
            string responseString = "";

            System.Net.HttpWebRequest request =
((HttpWebRequest)WebRequest.Create(resourceObj.Url));
            request.Headers.Add("X-Auth-Token:" + resourceObj.XauthToken);
            request.Accept = resourceObj.Accept;
            request.Method = resourceObj.Verb;

            // check to see if this is to upload object
            if(resourceObj.Verb.Equals("PUT"))
            {
                // if so, write the data
                UTF8Encoding encoding = new UTF8Encoding();
                byte [] buffer = encoding.GetBytes("This is the data in my object");
                Stream dataStream = request.GetRequestStream();
                dataStream.Write(buffer, 0, buffer.Length);
                dataStream.Close();
            }

            // check to see if this is writing metadata
            if (resourceObj.Verb.Equals("POST") && !resourceObj.Metadata.Equals(""))
            {
                string [] nameValuePairs = resourceObj.Metadata.Split(new char[] { ',' });
                foreach (string pair in nameValuePairs)
                {
                    request.Headers.Add("X-Object-Meta-" + pair);
                }
            }

            WebResponse response = null;
            response = request.GetResponse();

            if (response != null)
            {
```

```
                    Stream dataFromSite = response.GetResponseStream();

                    var sr = new StreamReader(dataFromSite);
                    responseString = sr.ReadToEnd();
                }

                // check to see if this is a request to read metadata
                if (resourceObj.Verb.Equals("HEAD"))
                {
                    responseString += Environment.NewLine;
                    foreach (string key in response.Headers.AllKeys)
                    {
                        string[] headerValues = response.Headers.GetValues(key);
                        foreach(string headerValue in headerValues)
                        {
                            responseString += key + ":" + headerValue + Environment.NewLine;
                        }
                    }
                }
                return responseString;
            }
        }
    }
```

proxyResource

```
    class proxyResource
    {
        string verb;
        string url;
        string XauthToken;
        string accept;
        string metadata;

        public string Verb { get { return verb; } set { verb = value; } }
        public string Url { get { return url; } set { url = value; } }
        public string XauthToken { get { return XauthToken; } set { XauthToken = value; } }
        public string Accept { get { return accept; } set { accept = value; } }
        public string Metadata { get { return metadata; } set { metadata = value; } }
    }
```

Appendix A Java code

With Java and C# being syntactically and functionally similar, the Java examples closely resemble the C# examples given in the main body of the text. For the most part the reader should be able to relate the comments made in the text surrounding the C# code. In instances where there is a significant difference the classes and code below are annotated to reflect those differences.

Authenticating the user

```
package example;


public class program {


  public static void main(String[] args) throws Exception {


        // authenticate user

        authentication authenticationObj = new authentication();

        authResponse authenticationResponseObj = authenticationObj.authenticate("user1",
"pass1", "tenant1", "http://10.32.19.165:5000/v2.0/tokens");

        System.out.println("X-Auth-Token to use is " +
authenticationResponseObj.x_auth_token);

        System.out.println("Selected proxy server is " +
authenticationResponseObj.objectStoreURL);


        if (authenticationResponseObj.x_auth_token.equals(""))

        {

            // user failed to authenticate

            return;

        }

}
```

Listing containers

```
        // list containers

        httpProxy proxyObj = new httpProxy();

        proxyResource resourceObj = new proxyResource();

        resourceObj.url = authenticationResponseObj.objectStoreURL;

        resourceObj.XauthToken = authenticationResponseObj.x_auth_token;

        resourceObj.verb = "GET";

        resourceObj.accept = "application/json";

        //resourceObj.accept = "application/xml";

        //resourceObj.accept = "text/plain";

        resourceObj.metadata = "";

        String response = proxyObj.AccessResource(resourceObj);

        System.out.println("List containers: " + response);
```

Creating containers

```
// create container
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "PUT";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Create container: " + response);
```

Creating objects

```
// create an object
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "PUT";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Create object: " + response);
```

List the contents of a container

```
// list contents of a container
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "GET";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("List objects in container: " + response);
```

Reading an object

```
// read an object
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "GET";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Read object: " + response);
```

Writing object metadata

```
// write object metadata
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "POST";
resourceObj.accept = "application/json";
resourceObj.metadata = "name1:value1,name2:value2";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Write object metadata: " + response);
```

Listing object metadata

```
// list object metadata
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "HEAD";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("List object metadata:\n" + response);
```

Deleting an object

```
// delete object
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "DELETE";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Delete object: " + response);
```

Deleting a container

```
// delete container
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "DELETE";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Delete container: " + response);
```

Supporting classes

Program

```
package example;


public class program {


  public static void main(String[] args) throws Exception {


        // authenticate user

        authentication authenticationObj = new authentication();

        authResponse authenticationResponseObj = authenticationObj.authenticate("user1",
"pass1", "tenant1", "http://10.32.19.165:5000/v2.0/tokens");

        System.out.println("X-Auth-Token to use is " +
authenticationResponseObj.x_auth_token);

        System.out.println("Selected proxy server is " +
authenticationResponseObj.objectStoreURL);


        if (authenticationResponseObj.x_auth_token.equals(""))

        {

            // user failed to authenticate

            return;

        }


        // list containers

        httpProxy proxyObj = new httpProxy();

        proxyResource resourceObj = new proxyResource();

        resourceObj.url = authenticationResponseObj.objectStoreURL;

        resourceObj.XauthToken = authenticationResponseObj.x_auth_token;

        resourceObj.verb = "GET";

        resourceObj.accept = "application/json";

        //resourceObj.accept = "application/xml";

        //resourceObj.accept = "text/plain";

        resourceObj.metadata = "";

        String response = proxyObj.AccessResource(resourceObj);

        System.out.println("List containers: " + response);


        // create container

        resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer";

        resourceObj.XauthToken = authenticationResponseObj.x_auth_token;

        resourceObj.verb = "PUT";

        resourceObj.accept = "application/json";
```

```
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Create container: " + response);

// create an object
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "PUT";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Create object: " + response);

// read an object
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "GET";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Read object: " + response);

// list contents of a container
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "GET";
resourceObj.accept = "application/json";
resourceObj.metadata = "";
response = proxyObj.AccessResource(resourceObj);
System.out.println("List objects in container: " + response);

// write object metadata
resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
resourceObj.verb = "POST";
resourceObj.accept = "application/json";
resourceObj.metadata = "name1:value1,name2:value2";
response = proxyObj.AccessResource(resourceObj);
System.out.println("Write object metadata: " + response);
```

```
        // list object metadata
        resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
        resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
        resourceObj.verb = "HEAD";
        resourceObj.accept = "application/json";
        resourceObj.metadata = "";
        response = proxyObj.AccessResource(resourceObj);
        System.out.println("List object metadata:\n" + response);

        // delete object
        resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/object1";
        resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
        resourceObj.verb = "DELETE";
        resourceObj.accept = "application/json";
        resourceObj.metadata = "";
        response = proxyObj.AccessResource(resourceObj);
        System.out.println("Delete object: " + response);

        // delete container
        resourceObj.url = authenticationResponseObj.objectStoreURL + "/myContainer/";
        resourceObj.XauthToken = authenticationResponseObj.x_auth_token;
        resourceObj.verb = "DELETE";
        resourceObj.accept = "application/json";
        resourceObj.metadata = "";
        response = proxyObj.AccessResource(resourceObj);
        System.out.println("Delete container: " + response);
  }

}
```

Program output

```
X-Auth-Token to use is d8f2635c3d7142fbb32572728e177963

Selected proxy server is
http://10.32.19.10:8888/v1/AUTH_7b9a902423a582c9eda266dcf3ad697473909b181af608c0497bda5fa37ce321

List containers: [{"count": 3, "bytes": 40, "name": "merlin"}]

Create container:

Create object:

Read object: This is the data in my object

List objects in container: [{"hash": "7b0a002987665da5f2a4c480ae865e33", "last_modified":
"2014-01-13T18:41:25.504270", "bytes": 29, "name": "object1", "content_type": "text/plain;
charset=ISO-8859-1"}]
```

```
Write object metadata:
List object metadata:
 --> X-Object-Meta-Name1:value1
 --> Content-Length:29
 --> X-Object-Meta-Name2:value2
 --> Accept-Ranges:bytes
 --> Last-Modified:Mon, 13 Jan 2014 18:41:25 GMT
 --> Etag:7b0a002987665da5f2a4c480ae865e33
 --> X-Timestamp:1389638485.58705
 --> Content-Type:text/plain; charset=ISO-8859-1
 --> Date:Mon, 13 Jan 2014 18:41:25 GMT
 --> Connection:keep-alive

Delete object:
Delete container:
```

authToken

```
package example;

import java.util.List;

public class authToken {
          // class representing the access token returned from the identity service
          public Access access;

          public class Tenant
          {
              public Boolean enabled;
              public String id;
              public String name;
          }

          public class Token
          {
              public String expires;
              public String id;
              public Tenant tenant;
          }
```

```
public class Endpoint
{
    public String adminURL;
    public String region;
    public String internalURL;
    public String id;
    public String publicURL;
}

public class ServiceCatalog
{
    public List<Endpoint> endpoints;
    public List<Object> endpoints_links;
    public String type;
    public String name;
}

public class Role
{
    public String name;
}

public class User
{
    public String username;
    public List<Object> roles_links;
    public String id;
    public List<Role> roles;
    public String name;
}

public class Metadata
{
    public Boolean is_admin;
    public List<String> roles;
}

public class Access
{
    public Token token;
```

```
            public List<ServiceCatalog> serviceCatalog;
            public User user;
            public Metadata metadata;
        }

}
```

Authentication

The authentication class utilizes libraries from the Google™ gson project[2] to convert the Json response string into a Java class. As in the C# example this is shown primarily to provide the reader with a way to process the Json response. The Json response is deserialized into the authenticationToken class.

```
package example;


import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;


import com.google.gson.Gson;


public class authentication
{


   public authResponse authenticate(String user, String password, String tenant, String url)
throws Exception
   {


      URL identityUrl = new URL(url);
      HttpURLConnection request = (HttpURLConnection) identityUrl.openConnection();


      request.setRequestProperty("Content-type", "application/json");
      request.setRequestMethod("POST");
      String urlParameters = "{\"auth\":{\"passwordCredentials\":"
         + "{\"username\":\"" + user + "\","
         + "\"password\":" + "\"" + password + "\"},"
         + "\"tenantName\":\"" + tenant + "\"}}";


      // Send post request
```

[2] code.google.com/p/google-gson/

```
request.setDoOutput(true);
DataOutputStream stream = new DataOutputStream(request.getOutputStream());
stream.writeBytes(urlParameters);
stream.flush();
stream.close();

BufferedReader sr = new BufferedReader(
new InputStreamReader(request.getInputStream()));

String inputLine;

StringBuffer jsonResponse = new StringBuffer();

while ((inputLine = sr.readLine()) != null)
{
   jsonResponse.append(inputLine);
}
sr.close();
authToken authenticationToken = null;
Gson gson = new Gson();
authenticationToken = gson.fromJson(jsonResponse.toString(), authToken.class);

if(authenticationToken == null)
{
   System.out.println("Unable to authenticate user");
   return null;
}

authResponse authResponseObj = new authResponse();
for (authToken.ServiceCatalog sc : authenticationToken.access.serviceCatalog)
{
   for (authToken.Endpoint ep : sc.endpoints)
   {
      if (ep.publicURL.contains(":8888"))
      {
         authResponseObj.objectStoreURL = ep.publicURL;
      }
   }
}
authResponseObj.x_auth_token = authenticationToken.access.token.id;
```

```
      return authResponseObj;
   }
}
```

authResponse

```
class authResponse
{
   public String x_auth_token;
   public String objectStoreURL;
}
```

httpProxy

The httpProxy class differs from the C# example in that the GET, HEAD, PUT, and POST processing are separated out to better annotate the use of the popular Apache HTTP client project[3] providing HTTP access from the application to the object storage server.

```
package example;

import java.io.IOException;
import java.io.InputStream;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpHead;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.params.BasicHttpParams;
import org.apache.http.params.HttpParams;

class httpProxy
{
    @SuppressWarnings("deprecation")
  public String AccessResource(proxyResource resourceObj) throws IOException
    {
        String responseString = "";
```

[3] hc.apache.org/httpclient-3.x/

```
        HttpClient client = new DefaultHttpClient();

//get
        if(resourceObj.verb.equals("GET"))
        {
           HttpGet getcmd = new HttpGet(resourceObj.url);
           getcmd.setHeader("X-Auth-Token", resourceObj.XauthToken);
           getcmd.setHeader("accept", resourceObj.accept);

           HttpResponse response = client.execute(getcmd);
           int code = response.getStatusLine().getStatusCode();
           int length = (int) response.getEntity().getContentLength();
           byte buffer[] = new byte[length];
           InputStream r = response.getEntity().getContent();
           r.read(buffer, 0, length);
           r.close();
           responseString = new String(buffer, "UTF-8");
           client.getConnectionManager().shutdown();
        }
//head
        if(resourceObj.verb.equals("HEAD"))
        {
           HttpHead headcmd = new HttpHead(resourceObj.url);
           headcmd.setHeader("X-Auth-Token", resourceObj.XauthToken);
           HttpResponse response = client.execute(headcmd);

           org.apache.http.Header[] headers = response.getAllHeaders();
           for (org.apache.http.Header header : headers)
           {
              responseString += " --> " + header.getName() + ":" + header.getValue() + "\n";
           }

           client.getConnectionManager().shutdown();
        }

//put
        if(resourceObj.verb.equals("PUT"))
        {
              HttpPut putcmd = new HttpPut(resourceObj.url);

              putcmd.setHeader("X-Auth-Token", resourceObj.XauthToken);
```

```
                HttpEntity e = new StringEntity("This is the data in my object");
                putcmd.setEntity(e);
                HttpResponse response = client.execute(putcmd);
             client.getConnectionManager().shutdown();
        }
//post
        if(resourceObj.verb.equals("POST"))
        {
           HttpPost postcmd = new HttpPost(resourceObj.url);
           postcmd.setHeader("X-Auth-Token", resourceObj.XauthToken);
           String [] nameValuePairs = resourceObj.metadata.split(",");
           for (String pair : nameValuePairs)
           {
                 String[] nameValue = pair.split(":");
                 postcmd.addHeader("X-Object-Meta-" + nameValue[0], nameValue[1]);
           }
           HttpResponse response = client.execute(postcmd);
           client.getConnectionManager().shutdown();
        }

//delete
        if(resourceObj.verb.equals("DELETE"))
        {
           HttpDelete deletecmd = new HttpDelete(resourceObj.url);
           deletecmd.setHeader("X-Auth-Token", resourceObj.XauthToken);
           HttpResponse response = client.execute(deletecmd);
           client.getConnectionManager().shutdown();
        }

        return responseString;
    }
}
```

proxyResource

```
class proxyResource
{
  public String verb;
  public String url;
  public String XauthToken;
  public String accept;
  public String metadata;
}
```

Appendix B PHP code

The following PHP examples mirror the functionality from the C# and Java examples above. The PHP code examples utilize the curl library for communicating with the object store HTTP interface.

Authenticating the user

```
<?php


//Connect to Keystone Server and get Authorization
 $username='user1';
 $password='pass1';
 $tenantname='tenant1';
 $URL='http://10.32.19.165:5000/v2.0/tokens';


 $array_container_url=array();
 $array_obj_url=array();
 $array_container_obj_url=array(array());
 $array_endpoint_container_obj_url=array(array(array()));


 //Set up Header info for Curl lib to list endpoints
 $header="Content-type: application/json";
 // Set up fields  info for Curl lib
 $fields='{"auth": {"tenantName": "' . $tenantname.'", "passwordCredentials":{"username":
"'.$username.'", "password": "'.$password.'"}}} ';
 // Set up URL info for Curl lib


 $detailsObtained = false;
 $cobj=curl_init($URL);
 curl_setopt($cobj,CURLOPT_RETURNTRANSFER,1);
 curl_setopt($cobj,CURLOPT_HTTPHEADER,array(
        $header,
        'Content-Length: ' . strlen($fields))
         );
 curl_setopt($cobj,CURLOPT_POSTFIELDS,$fields);
 $response=curl_exec($cobj);
 curl_close($cobj);
    $out_str=json_decode($response, true);


    $token = $out_str['access']['token']['id'];
    $expire_date=$out_str['access']['token']['expires'];
    $tenant_enabled=$out_str['access']['token']['tenant']['enabled'];
    $tenant_id=$out_str['access']['token']['tenant']['id'];
```

```
$tenant_name=$out_str['access']['token']['tenant']['name'];

$array_endpoint_admin_url = array();
$array_endpoint_region  = array();
$array_endpoint_internal_url  = array();
$array_endpoint_id = array();
$array_endpoint_public_url  = array();

//Endpoints - load into arrays
foreach ( $out_str['access']['serviceCatalog'][0]['endpoints'] as $endpoint )
{
  $array_endpoint_admin_url[] =  $endpoint['adminURL'];
  $array_endpoint_region[] =   $endpoint['region'];
  $array_endpoint_internal_url[]  =  $endpoint['internalURL'];
  $array_endpoint_public_url[]  =  $endpoint['publicURL'];
  $array_endpoint_id[] =   $endpoint['id'];
}

//Find number of endpoints
$number_of_end_point = count($array_endpoint_id);

 // Remove all this code later
// Print all variables for debug
echo "Token  - $token \n\r";
echo "Expire Date  - $expire_date \n\r";
echo "Tenant Enabled  - $tenant_enabled \n\r";
echo "Tenant Id - $tenant_id \n\r";
echo "Tenant Name  - $tenant_name \n\r";

echo "\n\r";
for($i=0;$i<$number_of_end_point;$i++ )
{
  echo "End Point - "  . ($i + 1)  . " \n\r";
  echo "Endpoint Admin  url  - $array_endpoint_admin_url[$i] \n\r";
  echo "Endpoint Region  - $array_endpoint_region[$i] \n\r";
  echo "Endpoint Internal url  - $array_endpoint_internal_url[$i] \n\r";
  echo "Endpoint Id  - $array_endpoint_id[$i] \n\r";
  echo "Endpoint Public url  - $array_endpoint_public_url[$i] \n\r";
}
```

```
?>
```

Listing containers

```
function listContainers($endpoint_public_url,$token)
{
     $array_containers=array();
     $detailsObtained = false;

     $cobj=curl_init($endpoint_public_url);

     curl_setopt($cobj,CURLOPT_RETURNTRANSFER,1);
     curl_setopt($cobj, CURLOPT_HTTPHEADER, array('X-Auth-Token: '.$token)) ;
     $response=curl_exec($cobj);

     curl_close($cobj);

     //Repalace all crlf with space and trim off the last white space
     $response = rtrim(ereg_replace("[\r\n]", ' ', $response));
    if ($response)
     {
        $detailsObtained = true;
        //Check Return Code - if completed then add code to carry on - we just check a
response happen here
     }
     if ($detailsObtained)
     {
       //Insert all the containers into a temporary array
       $array_containers =  explode(" ",$response);
       $number_of_containers = count($array_containers);

       // Remove below code later - this is an example of simple debug code
       // Print all variables for debug
        echo  $number_of_containers . " - Container(s) found at EndPoint
$endpoint_public_url\n\r";
        for($j=0;$j<$number_of_containers;$j++ )
        {
          $array_container_url[] = $endpoint_public_url . "/" . $array_containers[$j] ;
          echo "\t Container Name - "  . $array_containers[$j]  . " \n\r";

        }
    // Remove above code later
```

```
      }
      else
      {
        echo "No Containers found at EndPoint $endpoint_public_url\n\r";
      }

//Return an array of container URL's
 return($array_container_url);

 }
```

Creating containers

```
function addContainer($container_name,$end_point,$token)
{

     $new_container_str = $end_point . "/" . $container_name . "\n\r";
     $detailsObtained = false;
     $cobj=curl_init($end_point);
     curl_setopt($cobj,CURLOPT_RETURNTRANSFER,1);
     curl_setopt($cobj, CURLOPT_HTTPHEADER, array('X-Auth-Token: '.$token)) ;
     curl_setopt($cobj, CURLOPT_PUT, 1);
     curl_setopt($cobj, CURLOPT_URL, $new_container_str);

     $response=curl_exec($cobj);
     curl_close($cobj);

return($response);

}
```

Creating objects

```
function addObject($f_name,$end_point,$token)
{     $detailsObtained = false;
     $cobj=curl_init($end_point."?format=json");
     $dest_filename_and_url=$end_point . "/" . $f_name;
     curl_setopt($cobj, CURLOPT_URL, $dest_filename_and_url);
     curl_setopt($cobj, CURLOPT_PUT, 1);
     curl_setopt($cobj, CURLOPT_HTTPHEADER, array('X-Auth-Token: '.$token)) ;
     curl_setopt($cobj, CURLOPT_CUSTOMREQUEST, "PUT");
```

```
    $fh_res = fopen( $f_name , 'r');

    curl_setopt($cobj, CURLOPT_INFILE, $fh_res);
    curl_setopt($cobj, CURLOPT_INFILESIZE, filesize($f_name));

    curl_setopt($cobj,CURLOPT_RETURNTRANSFER,1);
    $response=curl_exec($cobj);
    fclose($fh_res);
    curl_close($cobj);

  if ($response)
    {
       $detailsObtained = true;
    }

    return($response);

}
```

List the contents of a container

```
function listObjects($container_url,$token)
{
    $array_objects=array();
    $detailsObtained = false;
    $cobj=curl_init($container_url);
    curl_setopt($cobj,CURLOPT_RETURNTRANSFER,1);
    curl_setopt($cobj, CURLOPT_HTTPHEADER, array('X-Auth-Token: '.$token)) ;
    $response=curl_exec($cobj);
    curl_close($cobj);

    //Repalace all crlf with space and trim off the last white space
    $response = rtrim(ereg_replace("[\r\n]", ' ', $response));

    if ($response)
    {
       $detailsObtained = true;
       checkReturnCode($response);
    }
```

```
     if ($detailsObtained)
     {
       //Insert all the containers into a temporary array
       $array_objects =  explode(" ",$response);
       $number_of_objects = count($array_objects);

       // Remove below code later
       // Print all variables for debug
        echo "Objects found in Container - ". substr( $container_url, strrpos(
$container_url, '/' )+1 )   ." \r\n";
        for($m=0;$m<$number_of_objects;$m++ )
        {
          $array_obj_url[] = $container_url . "/" . $array_objects[$m] ;
          echo "\t\t\t"  . $array_obj_url[$m]  . " \n\r";

        }
    // Remove above code later

      }
      else
      {
        echo "No Objects found in Container - ". substr( $container_url, strrpos(
$container_url, '/' )+1 )   ." \r\n";
        $array_obj_url=array();

      }
      echo " \n\r";
      return($array_obj_url);
 }
```

Writing object metadata

```
function changeObjectMetaData($object_name,$container_path,$token,$new_meta_data)
{

     //Get all the metadata of the object first
     $meta_data = listObjectMetaData($object_name,$container_path,$token);
     $header_array = preg_split('/\n|\r\n?/', $meta_data['header']);
     $header_array[] = 'X-Auth-Token: '.$token;
     $header_array[]="X-Object-Meta-Sport-Type:Tennis";
     $header_array[]="X-Object-Meta-Player:Martina";

     $new_object_str = $container_path . "/" . $object_name ;
     echo "Object path  - $new_object_str \n\r";
     $detailsObtained = false;
     $cobj=curl_init($c_path);
     curl_setopt($cobj,CURLOPT_RETURNTRANSFER,1);
    curl_setopt($cobj, CURLOPT_HTTPHEADER, $header_array) ;
     curl_setopt($cobj, CURLOPT_CUSTOMREQUEST, "POST");
     curl_setopt($cobj, CURLOPT_URL, $new_object_str);

     $response=curl_exec($cobj);

     curl_close($cobj);


return($response);
}
```

Listing object metadata

```
function listObjectMetaData($object_name,$container_path,$token)
{

     $new_object_str = $container_path . "/" . $object_name ;

     $cobj=curl_init($new_object_str);
     curl_setopt($cobj, CURLOPT_RETURNTRANSFER, 1);
     curl_setopt($cobj, CURLOPT_HEADER, 1);

     $headr = array();
     $headr[] = 'X-Auth-Token: '.$token;
```

```
$detailsObtained = false;
curl_setopt($cobj, CURLOPT_HTTPHEADER, $headr) ;
curl_setopt($cobj, CURLOPT_CUSTOMREQUEST, "GET");
curl_setopt($cobj, CURLOPT_URL, $new_object_str);

$response=curl_exec($cobj);

$header_size = curl_getinfo($cobj, CURLINFO_HEADER_SIZE);
$header = substr($response, 0, $header_size);
$body = substr($response, $header_size);

$header_array = preg_split('/\n|\r\n?/', $header);

//Debug info - remove when complete
foreach ($header_array as $header_element )
{
echo "$header_element \n\r";
}

 curl_close($cobj);

if ($response)
{
   $detailsObtained = true;
}

  //return array containing header and body
return array('body' => $body, 'header' => $header);

}
```

Deleting an object

```
function deleteObject($object_name,$container_path,$token)
{

     $new_object_str = $container_path . "/" . $object_name ;     $detailsObtained = false;
     $cobj=curl_init($container_path);
     curl_setopt($cobj,CURLOPT_RETURNTRANSFER,1);
     curl_setopt($cobj, CURLOPT_HTTPHEADER, array('X-Auth-Token: '.$token)) ;
     curl_setopt($cobj, CURLOPT_CUSTOMREQUEST, "DELETE");
```

```
        curl_setopt($cobj, CURLOPT_URL, $new_object_str);

        $response=curl_exec($cobj);

        curl_close($cobj);

        if ($response)
        {
           //Print out Response
           //print_r(json_decode($response));
           $detailsObtained = true;
        }

    return($response);

    }
```

Summary

As mentioned earlier, the intent of this document is to help jump start first-time OpenStack application developers through working examples illustrating the interaction between the application and the object store. Hopefully that goal has been accomplished and you are well on your way to developing your first OpenStack application.

Learn more at
hp.com/go/storeall

4AA5-0807ENW, February 2014