Private Public Partnership Project (PPP)
Large-scale Integrated Project (IP)

D.4.4.1b: FI-WARE User and Programmers Guide

Project acronym: FI-WARE
Project full title: Future Internet Core Platform
Contract No.: 285248
Strategic Objective: FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform
Project Document Number: ICT-2011-FI-285248-WP4-D.4.4.1b
Project Document Date: 2012-11-12
Deliverable Type and Security: Public
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium
1.1 Executive Summary
This document describes the usage of each Generic Enabler provided by the "Cloud Hosting" chapter.
This document consolidates new contents and also contents in previous issues of Release 1. The reason for re-delivering parts that were already issued is twofold:
- FI-WARE has made an effort to create a unified and improved format. The parts generated in the past are also provided in the new enhanced format for the sake of uniformity and readability.
- A single reference document per chapter is clearer and easier to handle than two incremental issues.

1.2 About This Document
This document comes along with the Software implementation of components, each release of the document being referred to the corresponding Software release (as per D.x.2), to provide documentation of the features offered by the components and interfaces to users/adopters. Moreover, it explains the way they can be exploited in their developments.

1.3 Intended Audience
The document targets users as well as programmers of FI-WARE Generic Enablers.

1.4 Chapter Context
The Cloud Chapter offers Generic Enablers that comprise the foundation for designing a modern cloud hosting infrastructure that can be used to develop, deploy and manage Future Internet applications and services. The solution focuses on fundamental cloud capabilities enabling provisioning and life cycle management of virtual machines and associated resources (compute, storage, network, images, etc) hosting FI applications and services, as well as object storage capabilities which can be used directly by FI applications and services via a REST API. In future releases, additional capabilities will be added, notably the support for complex services comprising multiple virtual machines, including monitoring, policy-based elasticity, as well as many others, as outlines in Roadmap of Cloud Hosting. The following diagram shows the main components (Generic Enablers) that comprise the first release of FI-WARE architecture.
The GEs in the above diagram are grouped into Core GEs, providing the core hosting capabilities at different abstraction levels (resources, services, objects, etc) and Ecosystem GEs, addressing various specific needs across the Core GEs, and establishing the ecosystem that enables the end-to-end capabilities provided by a cloud offering.

The Core GEs include:

- **Data Center Resource Management (DCRM) GE**, offering provisioning and life cycle management of virtualized resources (compute, storage, network) associated with virtual machines.
- **Object Storage GE**, offering provisioning and life cycle management of object-based storage containers and elements
- **Service Management (SM) GE**, offering provisioning and life cycle management of composite services comprising several resources provided by one of the above GEs. In the first release of FI-WARE, Service Management GE will consume resources provided by Data Center Resource Management GE, via the corresponding APIs.

The Ecosystem GEs include:

- **Monitoring GE**, collecting metrics associated with each of the Core GEs, and offering them to GEs which are interested to consume such metrics. For example, Service Management GE consumes metrics associated with KPIs of the various service components in order to drive auto-scaling decisions. In the future, more advanced metrics-related capabilities will be provided, such as processing (before it is delivered to the consumer), archival and analysis of metrics.
- **Identity Management GE**, providing a unified management of users, roles and tokens, that can be used by other GEs for authentication and authorization purposes. This GE will be provided by the Security Chapter.
1.5 Structure of this Document

The document is generated out of a set of documents provided in the public FI-WARE wiki. For the current version of the documents, please visit the public wiki at http://wiki.fi-ware.eu/

The following resources were used to generate this document:
- D.4.4.1b FI-WARE User and Programmers Guide front page
- IaaS Data Center Resource Management - User and Programmers Guide
- IaaS Service Management - User and Programmers Guide
- Object Storage - User and Programmers Guide
- Self Service Interfaces - User Guide
- Cloud Proxy - User and Programmers Guide

1.6 Typographical Conventions

Starting with October 2012 the FI-WARE project improved the quality and streamlined the submission process for deliverables, generated out of the public and private FI-WARE wiki. The project is currently working on the migration of as many deliverables as possible towards the new system.

This document is rendered with semi-automatic scripts out of a MediaWiki system operated by the FI-WARE consortium.

1.6.1 Links within this document

The links within this document point towards the wiki where the content was rendered from. You can browse these links in order to find the "current" status of the particular content. Due to technical reasons not all pages that are part of this document can be linked document-local within the final document. For example, if an open specification references and "links" an API specification within the page text, you will find this link firstly pointing to the wiki, although the same content is usually integrated within the same submission as well.

1.6.2 Figures

Figures are mainly inserted within the wiki as the following one:

```
[[Image:....|size|alignment|Caption]]
```

Only if the wiki-page uses this format, the related caption is applied on the printed document. As currently this format is not used consistently within the wiki, please understand that the rendered pages have different caption layouts and different caption formats in general. Due to technical reasons the caption can't be numbered automatically.

1.6.3 Sample software code

Sample API-calls may be inserted like the following one.

```
1.7 Acknowledgements
The current document has been elaborated using a number of collaborative tools, with the participation of Working Package Leaders and Architects as well as those partners in their teams they have decided to involve; IBM, Intel, Technicolor, Telefonica.

1.8 Keyword list

1.9 Changes History

<table>
<thead>
<tr>
<th>Release</th>
<th>Major changes description</th>
<th>Date</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>First draft of deliverable submission generated</td>
<td>2012-11-11</td>
<td>IBM</td>
</tr>
</tbody>
</table>

1.10 Table of Contents

1.1 Executive Summary .................................................................................................................................. 2
1.2 About This Document .................................................................................................................................. 2
1.3 Intended Audience .................................................................................................................................... 2
1.4 Chapter Context .......................................................................................................................................... 2
1.5 Structure of this Document ....................................................................................................................... 4
1.6 Typographical Conventions ....................................................................................................................... 4
1.6.1 Links within this document .................................................................................................................. 4
1.6.2 Figures .................................................................................................................................................... 4
1.6.3 Sample software code ............................................................................................................................ 4
1.7 Acknowledgements ..................................................................................................................................... 5
1.8 Keyword list ............................................................................................................................................... 5
1.9 Changes History ........................................................................................................................................ 5
1.10 Table of Contents .................................................................................................................................... 5
2 IaaS Data Center Resource Management - User and Programmers Guide ........................................................................ 7
2.1 Introduction ............................................................................................................................................... 7
2.2 User Guide .................................................................................................................................................. 7
### Future Internet Core Platform

2.3 Developer Guide ........................................................................................................... 7
2.3.1 Accessing OCCI within Python .................................................................................. 8
2.3.2 Accessing OCCI from the CLI ................................................................................... 8

3 IaaS Service Management - User and Programmers Guide ........................................... 9
3.1 Introduction .................................................................................................................. 9
3.2 User Guide .................................................................................................................. 9
3.3 Developer Guide ......................................................................................................... 9
  3.3.1 Accessing Service Manager from the CLI .............................................................. 10
  3.3.2 Accessing Service Manager from a Browser ......................................................... 19

4 Object Storage - User and Programmers Guide ............................................................... 24
4.1 Introduction ................................................................................................................. 24
4.2 User Guide .................................................................................................................. 24
4.3 Developer Guide ......................................................................................................... 24
  4.3.1 Programming against the CDMI interface ........................................................... 25
  4.3.2 Using CLI commands ........................................................................................... 26

5 Self Service Interfaces - User Guide ................................................................................. 27
5.1 Introduction ................................................................................................................. 27
5.2 User Portal User Guide ............................................................................................... 27
5.3 Scripts User Guide ..................................................................................................... 28
2 IaaS Data Center Resource Management - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

2.1 Introduction

Welcome the User and Programmer Guide for the Data Centre Resource Manager Generic Enabler. This generic enabler is built on an Open Source project, the OCCI API for OpenStack, and so where possible this guide points to the appropriate online content that has been created for this project. The online documents are being continuously updated and improved, and so will be the most appropriate place to get the most up to date information on installation and administration.

2.2 User Guide

The OpenStack wiki offers a wiki page which describes how to use the OCCI interface in detail. It includes a bunch of commands which can be used to manipulate instance in the Cloud.

Since OCCI is based on the HTTP protocol there is a rich variety of tools which can be used to access the OpenStack OCCI interface.

The following sections will go into more detail on how to use and develop against OCCI.

2.3 Developer Guide

OCCI is based upon HTTP and therefore all devices which can handle HTTP traffic and possible OCCI clients. This also means that most programming languages can be used to access OCCI.

To give a feeling of how OCCI work lets take a look at an HTTP request and the corresponding response:

```plaintext
> GET /- HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0
OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: localhost:8787
> Accept: */*
> X-Auth-Token: 0cc2bab32f3246919c2d2cbea314d850

> HTTP/1.1 200 OK
< Content-Length: 10917
< Content-Type: text/plain
< Server: pyssf OCCI/1.1
< Date: Thu, 28 Jun 2012 09:28:38 GMT
<
```
In this case the HTTP GET operation is used. Please note the X-Auth-Token which is needed by OpenStack for authentication and assignment of user to a project. OCCI uses the Create Retrieve Update Delete (CRUD) operations which map almost to the HTTP verbs POST, GET, PUT and DELETE. It should also be mentioned that OCCI can deal with more than one media-type. In the example above ‘/*’ maps to the Content-Type ‘text/plain’ as the response demonstrates. Clients can also particularly request and information using ‘text/occi’, ‘text/plain’ and in future ‘application/occi+json’ The format of how Information is rendered on the wire are described by ABNF notations which can be found in the OCCI specification.

2.3.1 Accessing OCCI within Python

Just as an example how to programmatically access OCCI the following Python code gives a quick introduction:

```python
headers = {"Content-type": "text/occi",
           "Accept": "text/occi"}
conn = httplib.HTTPConnection('http://localhost:8787')
conn.request('GET', '/', None, headers)
response = conn.getresponse()
```

In this case a header is prepared and send on to the service using an HTTP connection. Very similar setups can also be achieved using the Java programming language or something like C.

2.3.2 Accessing OCCI from the CLI

'curl' is a command which can be used to perform any kind of HTTP operation - and therefore is also usable for OCCI. The library 'libcurl' can be integrated in C programs as well. The following command for example will issue the creation of an VM:

```
curl -v -X POST localhost:8787/compute/ \
  -H 'Content-Type: text/occi' \
  -H 'X-Auth-Token: '$KID \
  -H 'Category: compute; scheme="http://schemas.ogf.org/occi/infrastructure#"; class="kind"' \
  -H 'Category: itsy; scheme="http://schemas.openstack.org/template/resource#"; class="mixin"' \
  -H 'Category: cirros-0.3.0-x86_64-uec; scheme="http://schemas.openstack.org/template/os#"; class="mixin"'
```

More examples can be seen on the OpenStack wiki.
3 IaaS Service Management - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

3.1 Introduction

Welcome the User and Programmer Guide for the Service Manager Generic Enabler. This generic enabler is built on a proprietary solution using standard interface to communicate with, like OpenStack Compute API, and so where possible this guide points to the appropriate online content that has been created for this specific API. The online documents are being continuously updated and improved, and so will be the most appropriate place to get the most up to date information on using this interface.

3.2 User Guide

The OpenStack API reference offers a [1] page, which describes how to use the OpenStack Compute API (plus Identity and extensions from third parties not all included in the implementation currently) in details. It includes a bunch of commands, which can be used to manipulate instance in the Cloud.

Since OpenStack Compute API is based on the HTTP protocol there are a rich variety of tools, which can be used to access the Service Manager interface.

The following sections will go into more detail on how to user and develop against OpenStack Compute API.

3.3 Developer Guide

OpenStack Compute API is based upon HTTP and therefore all devices, which can handle HTTP traffic, are possible clients. This also means that most programming languages can be used to access Service Manager through OpenStack Compute API.

To give a feeling of how OCCI work lets take a look at an HTTP request and the corresponding response:

```
* About to connect() to 130.206.80.91 port 8774 (#0)
*   Trying 130.206.80.91... connected
* Connected to 130.206.80.91 (130.206.80.91) port 8774 (#0)
> GET /v2.0/c8da25c7a373473f8e8945f5b0da8217/flavors HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8r zlib/1.2.3
> Host: 130.206.80.91:8774
> Access-Control-Request-Method: GET
> Access-Control-Request-Headers: Content-Type, X-Auth-Token
> Origin: http://10.95.41.137
```
In this case the HTTP GET operation is used. Please note the X-Auth-Token, which is needed by Service Manager for authentication and assignment of user to a project. OpenStack Compute API uses the Create Retrieve Update Delete (CRUD) operations, which map almost to the HTTP verbs POST, GET, PUT and DELETE. OpenStack compute API deal with two media types, application/xml and application/json, both Content-Type and Accept, which means that we can send a xml content and receive a json response and vice versa.

3.3.1 Accessing Service Manager from the CLI
The access through the CLI is made using the curl program. Curl [2] is a client to get documents/files from or send documents to a server, using any of the supported protocols (HTTP, HTTPS, FTP, Gopher, Dict, Telnet, LDAP or File) and therefore is also usable for OpenStack Compute API. Use the curl command line tool or use libcurl from within your own programs in C. Curl is free and open software that compiles and runs under a wide variety of operating systems.

The normal operations sequence to create a VM could be summarized in the following list:

1. Get a valid token for the tenant that we have (It is not a Service Manager Operation itself it is a OpenStack Keystone operation.)
curl -d '{"auth": {"tenantName": "$TENNANT",  
"passwordCredentials": {"username": "$USERNAME",  "password":  
$PASSWORD}}}' 
-H "Content-type: application/json" -H "Accept: application/xml"  
http://130.206.80.100:35357/v2.0/tokens

Both $TENNANT (Project), $USERNAME and $PASSWORD must be values previously  
created in the OpenStack Keystone. The IP address 10.95.171.115 and the Port 35357 are  
the data of our internal installation of OpenStack Keystone, if you planned to execute it you  
must changed it by the corresponding IP and Port of the FIWARE Keystone or IdM IP and  
Port addresses.

We obtained two data from the previous sentence:

- X-Auth-Token

<token expires="2012-10-25T16:35:42Z"  
id="a9a861db6276414094bc1567f664084d">  

- Tenant-Id

<tenant enabled="true" id="c8da25c7a373473f8e8945f5b0da8217"  
name="$TENNANT">

2. Get the list of available flavors.

```
curl -v -H "Access-Control-Request-Method: GET" -H "Access-Control-  
Request-Headers: Content-Type, X-Auth-Token" -H "Origin:  
http://130.206.80.93" -H 'X-Auth-Token:  
a9a861db6276414094bc1567f664084d' -H "Content-Type: application/xml"  
-H "Accept: application/xml" -X GET  
"http://$SERVICEMANAGER_IP:8774/v2.0/c8da25c7a373473f8e8945f5b0da821  
7/flavors"
```

$SERVICEMANAGER_IP will be the IP in which the OpenStack Compute API is installed.  
We obtained the Id of the flavor, and taking one for example:

```
<ns3:flavor id="001" name="M1_TINY 100Mb">  
<ns2:link rel="self"  
href="http://$SERVICEMANAGER_IP:8774/v2.0/c8da25c7a373473f8e8945f5b0da8217/flavors/001"/>  
</ns3:flavor>
```

More functionalities can be seen on the OpenStack Compute API [3].

3. Get the list of available images.

```
$ curl -v -H "Access-Control-Request-Method: GET" -H "Access-  
Control-Request-Headers: Content-Type, X-Auth-Token" -H "Origin:  
```
We obtained the list of available images, in our example only one:

```json
{
  "images": [
    {
      "id": "16e3fe14-ecf6-1be3-edfa-fe180ee2e6f2",
      "name": "cirros-0.3.0-x86_64-uec",
      "link": [
        {
          "rel": "self",
          "href": "http://$SERVICEMANAGER_IP:8774/v2.0/images/16e3fe14-ecf6-1be3-edfa-fe180ee2e6f2"
        }
      ]
    }
  ]
}
```

4. Create a Virtual Data Center
In this step, we create a Virtual Data Center with the name v3 and Id 3 with the following characteristics:

- Two different network configuration:
  - `management` with a 1Gb of Bandwidth
  - `service` with a 1Gb of Bandwidth
- Storage capacity of 50,000 GB
- Compute capacity of 10,000 units
- Memory of 10,000 GB

This information will be used afterward during the creation of the services and servers in order to know the available capacity of the virtual data center.

```
```
"http://$SERVICEMANAGER_IP:8774/v2.0/c8da25c7a373473f8e8945f5b0da8217/vdc" -d '<?xml version="1.0" encoding="UTF-8"?>

  <VDC name="v3" Id="3">
  <Description>A description...</Description>
  <Topology>
    <Zone type="public-protected">
      <VirtualNetwork id="1" name="service">
        <Description>Godzillas ADSL</Description>
        <Mode>routed</Mode>
        <Bandwidth units="Mb">6</Bandwidth>
        <Size>2</Size>
      </VirtualNetwork>
    </Zone>
    <Zone type="private-protected">
      <VirtualNetwork id="2" name="management">
        <Description>Godzillas Intranet</Description>
        <Mode>isolated</Mode>
        <Bandwidth units="Gb">1</Bandwidth>
        <Size>254</Size>
      </VirtualNetwork>
    </Zone>
  </Topology>
  <StorageCapacity>
    <Disk>
      <Units>byte * 2 ^ 30</Units>
      <Limit>50000.0</Limit>
    </Disk>
  </StorageCapacity>
  <ComputeCapacity>
    <Cpu>
      <Units># CPUs</Units>
      <Limit>10000.0</Limit>
    </Cpu>
    <Memory>
      <Units>byte * 2 ^ 30</Units>
      <Limit>10000.0</Limit>
    </Memory>
  </ComputeCapacity>
</VDC>
We obtain a 200 Ok message with the following content body:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VDC xmlns="http://schemas.tcloud.telefonica.com/tcloud/1"
     xmlns:nss2="http://schemas.dmtf.org/ovf/envelope/1"
     xmlns:nss3="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData"
     xmlns:nss4="http://schemas.dmtf.org/wbem/wscim/1/common"
     xmlns:nss5="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ResourceAllocationSettingData"
     xmlns:nss6="http://schemas.telefonica.com/claudia/ovf" name="v3"/>
```

5. Create of a service inside of the previous virtual data center.

In this step, we create a service identified by $s1$ associated to the previous virtual data center, but without any server defined. In the following steps we provide the definition of the servers associated to this service.

```
</InstantiateOvfParams name="s1"
 xsi:schemaLocation="http://schemas.tcloud.telefonica.com/tcloud/1
tcloud.xsd"
 xmlns="http://schemas.tcloud.telefonica.com/tcloud/1"
 xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
 xmlns:vssd="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData"
 xmlns:rasd="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ResourceAllocationSettingData"
 xmlns:rsrvr="http://schemas.telefonica.com/claudia/ovf"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <ovf:Envelope>
     <ovf:NetworkSection>
       <ovf:Info>The service considers one internal network and one public</ovf:Info>
```

We obtain a 200 Ok message with the following content body:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Service xmlns:ns2="http://schemas.dmtf.org/ovf/envelope/1"
xmlns:ns3="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData"
xmlns:ns4="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ResourceAllocationSettingData"
xmlns:ns5="http://schemas.telefonica.com/claudia/ovf"
xmlns:ns6="http://schemas.tcloud.telefonica.com/tcloud/1"
xmlns:ns7="http://schemas.telefonica.com/tcloud/1"
name="s1"/>
```

6. Create of a server inside the previous service.
We obtain a 200 Ok message with the following content body:

```xml
<server xmlns="http://docs.openstack.org/compute/api/v1.1" flavorRef="001" name="demoFIWARE" imageRef="cirros-0.3.0-x86_64-uec"/>
```

7. Get the list of servers associated to a service.
Given a virtual data center v3 and a service s1, the list of servers associated to them, can be obtained through the following commands:

```
$ curl -v -H "Access-Control-Request-Method: GET" -H "Access-
Control-Request-Headers: Content-Type, X-Auth-Token" -H "Origin:
http://130.206.80.93" -H 'X-Auth-Token:
a9a861db6276414094bc1567f664084d' -H "Content-Type: application/xml"
-H "Accept: application/xml" -X GET
"http://$SERVICEMANAGER_IP:8774/v2.0/c8da25c7a373473f8e8945f5b0da821
7/vdc/v3/services/s1/servers" -d '<?xml version="1.0" encoding="UTF-
8"?>
<server xmlns="http://docs.openstack.org/compute/api/v1.1"
flavorRef="001" name="demoFIWARE" imageRef="cirros-0.3.0-x86_64-uec"
/>
```

We obtain a return message with 200 Ok with the following body content:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<servers xmlns:ns2="http://www.w3.org/2005/Atom"
xmlns:ns3="http://docs.openstack.org/compute/api/v1.1">
    <ns3:server name="s1 demoFIWARE" id="0b7ed9b9-1ed7-4471-97df-11ca8295e50a">
        <ns3:metadata/>
        <ns3:personality/>
    </ns3:server>
</servers>
```
8. Check the list of server of other service

Given that the server was created in the virtual data center v3 and service s1, we try to obtain the list of servers of the virtual data center v2 and service s1 in order to check that the Claudia does not return any value.

```
```

We obtain a 200 Ok message with the following body content:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<servers xmlns:ns2="http://www.w3.org/2005/Atom"
xmlns:ns3="http://docs.openstack.org/compute/api/v1.1">
  
  <ns3:server name="s1 test1"
id="0e4c3452a6cf40cd84db6f9adfe178a8">
    <ns2:link rel="self"
href="http://$SERVICEMANAGER_IP:8774/v2.0/c8da25c7a373473f8e8945f5b0da8217/vdc/v2/services/s1/servers/0e4c3452a6cf40cd84db6f9adfe178a8"/>
  </ns3:server>

  <ns3:server name="s1 test2"
id="823fe9ebecb44c093ccae5be1c7e938">
    <ns2:link rel="self"
href="http://$SERVICEMANAGER_IP:8774/v2.0/c8da25c7a373473f8e8945f5b0da8217/vdc/v2/services/s1/servers/823fe9ebecb44c093ccae5be1c7e938"/>
  </ns3:server>

</servers>
```

9. Delete the server s1 test
Give a server name \textit{s1 test} (id=0e4c3452a6cf40cd84db6f9adfe178a8), a service \textit{s1} and a virtual data center \textit{v2}, we try to delete the server through the following commands:

\begin{verbatim}
\end{verbatim}

The command return a 204 Ok without any content body.

10. Delete the server \textit{s1 test2}
Give a server name \textit{s1 test2} (id=823fe9eabecb44c093ccae5be1c7e938), a service \textit{s1} and a virtual data center \textit{v2}, we try to delete the server through the following commands:

\begin{verbatim}
\end{verbatim}

The command return a 204 Ok without any content body.

11. List the servers of a specific service and virtual data center
In order to check the two previous operation, we obtain the list of servers contained in the virtual data center \textit{v2} and service \textit{s1}.

\begin{verbatim}
\end{verbatim}

The command return a 200 Ok with the following content body:

\begin{verbatim}
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<servers xmlns:ns2="http://www.w3.org/2005/Atom"
xmlns:ns3="http://docs.openstack.org/compute/api/v1.1"/>
\end{verbatim}
How you can see, the service information does not include any reference to the previous deleted servers.

### 3.3.2 Accessing Service Manager from a Browser

We are using the Chrome browser [4] with the Simple REST Client plugin [5] in order to send http commands to the Service Manager. You can use it also in Firefox through RESTClient add-ons [6].

We follow the same sequence that we take previously in order to create a VM:

1. Get a valid token for the tenant that we have (It is not a Service Manager Operation itself it is a OpenStack Keystone operation).

2. Get the list of available flavors.
3. Get the list of available images.

   ![Simple REST Client]

   **Request**
   
   URL: `http://130.206.80.91:8774/v2/0/06a3323c7a374742b8085f/0b5b0c9217/Images`
   
   **Method:** GET | POST | PUT | DELETE | HEAD | OPTIONS
   
   **Headers:**
   - X-Auth-Token: 5406ba5e377220d6c25f5b15c51dc810f
   - Content-Type: application/x
   - Accept: application/x
   - Access-Control-Request-Method: GET
   - Access-Control-Request-Headers: Content-Type, X-Auth-Token, Origin
   - Origin: http://127.0.0.1

   **Response**
   
   Status: 200 OK
   
   **Headers:**
   - Date: Wed, 24 Oct 2012 16:05:16 GMT
   - X-Server: Apache-Coyote/1.1
   - Content-Length: 1198
   - WWW-Authenticate: keystone url: http://130.206.80.100:35357/v2.0
   
   **Data:**
   
   ```
   <xml version="1.0" encoding="UTF-8" standalone="yes" xmlns="http://www.w3.org/2005/Atom" xmlns:xml="http://www.w3.org/2005/Atom">
     <entry>
       <id>http://130.206.80.91:8774/v2/0/06a3323c7a374742b8085f/0b5b0c9217/Images</id>
       <title type="html">Images</title>
       <link rel="self" href="http://130.206.80.91:8774/v2/0/06a3323c7a374742b8085f/0b5b0c9217/Images?detail=1"/>
       <link rel="alternate" type="application/xml" href="http://130.206.80.91:8774/v2/0/06a3323c7a374742b8085f/0b5b0c9217/Images"/>
       <link rel="self" href="/images/0b5b0c9217/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e5/0e5f1e48788b16b10c5df8c64b9d29e
   
   4. Create a Virtual Data Center
   
   In this step, we create a Virtual Data Center with the name v4 and Id 4 with the following characteristics:
   
   - Two different network configuration:
     - *management* with a 1Gb of Bandwith
     - *service* with a 1Gb of Bandwith
- Storage capacity of 50,000 GB
- Compute capacity of 10,000 units
- Memory of 10,000 GB

5. Create of a service inside of the previous virtual data center.
   In this step, we create a service identified by $s^f$ associated to the previous virtual data center, but without any server defined. In the following steps we provide the definition of the servers associated to this service
6. Create of a server inside the previous service.

7. Get the list of servers associated to a service.

Given a virtual data center $v4$ and a service $s1$, the list of servers associated to them, can be obtained through the following commands:
8. Get the list of servers (OpenStack API compute compliant). Given the tenant id and the security token that we are using along the process, we obtain the complete list of servers available in OpenStack:
4 Object Storage - User and Programmers Guide

You can find the content of this chapter as well in the wiki of fi-ware.

4.1 Introduction

Welcome the User and Programmer Guide for the Object Storage Generic Enabler. This generic enabler is built on an Open Source project, the CDMI API for OpenStack, and so where possible this guide points to the appropriate online content that has been created for this project. The online documents are being continuously updated and improved, and so will be the most appropriate place to get the most up to date information on installation and administration.

4.2 User Guide

The CDMI specification is available from SNIA

4.3 Developer Guide

Again CDMI just like OCCI uses many HTTP like operations. For example to retrieve a Data object from an CDMI container a simple HTTP GET command will do. Please note that with every CDMI call you need to provide the CDMI version you are using!

The following request:

```
GET /MyContainer/MyDataObject.txt HTTP/1.1
Host: cloud.example.com
Accept: application/cdmi-object
X-CDMI-Specification-Version: 1.0.1
X-Auth-Token: 0cc2bab32f3246919c2d2cbea314d850
```

should return the following response

```
HTTP/1.1 200 OK
X-CDMI-Specification-Version: 1.0.1
Content-Type: application/cdmi-object
{
    "objectType" : "application/cdmi-object",
    "objectID" : "0000706D0010B84FAD185C425D8B537E",
    "objectName" : "MyDataObject.txt",
    "parentURI" : "/MyContainer/",
    "parentID" : "00007E7F00102E230ED82694DAA975D2",
    "domainURI" : "/cdmi_domains/MyDomain/",
    "capabilitiesURI" : "/cdmi_capabilities/dataobject/",
    "completionStatus" : "Complete",
```
In this case the CDMI specific media-types are used. Other can be used to if you use the HTTP mime-types accordingly.

4.3.1 Programming against the CDMI interface

As mentioned using the CDMI interface is straight forward since it is build upon the HTTP protocol. The following sections will focus on how to create an client application.

4.3.1.1 Authentication

Authentication is done through OpenStack's KeyStone service. In general the service needs to know a 'user', 'password' and a 'tenant' which reflects to a project name. These values need to be encoded in a JSON rendered string which than will be send to the KeyStone service. The following Python code snippet shows how this request could look like:

```python
cconn = httplib.HTTPConnection('192.168.56.101:5000')
headers = {'Content-type':'application/json'}

body = json.dumps({
    "auth": {
    "tenantName": tenant_name + ",
    "passwordCredentials": {
    "username": username + ",
    "password": password + 
    }}
'])

cconn.request("POST", '/v2.0/tokens', body, headers)
response = cconn.getresponse()
if response.status not in [200]:
    print 'Failed to auth: ', response.reason, response.read()

heads = {}
for item in response.getheaders():
    heads[item[0]] = item[1]

body = response.read()
cconn.close()

return heads, body
```

When authenticated some important values can be extracted from the 'body'. First of all the keystone id - which is needed for authentication. And the publicURL for the Object Storage. This URL will be used later for the CDMI calls.
4.3.1.2  Create a Container

Just as another example: the following Python code snippet shows how a Container can be created:

```python
conn = httpplib.HTTPConnection(URL)
headers = {'X-Auth-Token': token,
           'content-type': 'application/directory',
           'content-length': '0'}
conn.request('PUT', URN '/' + container_name, None, headers)
res = conn.getresponse()
```

Please note the Content-Type and the X-Auth-Token which will need to be send within the headers of the requests.

4.3.2  Using CLI commands

Use the Authentication token and URL which is in the response from the last curl command to perform an *GET* operation:

```bash
curl -v -X GET -H 'X-Auth-Token: AUTH_tk56b01c82710b41328db7c9f953d3933d'
http://127.0.0.1:8080/v1/AUTH_test
```

Create a Container:

```bash
curl -v -X PUT -H 'X-Auth-Token: AUTH_tk56b01c82710b41328db7c9f953d3933d' -H 'Content-tType: application/directory' -H 'Content-Length: 0'
http://127.0.0.1:8080/v1/AUTH_test/<container_name>
```

Query the capabilities of a Container:

```bash
curl -v -X GET -H 'X-Auth-Token: AUTH_tk56b01c82710b41328db7c9f953d3933d' -H 'X-CDMI-Specification-Version: 1.0.1'
http://127.0.0.1:8080/cdmi/cdmi_capabilities/AUTH_test/<container_name>
```

Add an Object to a Container:

```bash
http://127.0.0.1:8080/v1/AUTH_test/<container_name>/<object_name> -d '<Some JSON>'
```
5 Self Service Interfaces - User Guide

You can find the content of this chapter as well in the wiki of fi-ware.

5.1 Introduction

Welcome to the User Guide of the Self Service Interfaces. The Self Service Interfaces provide a support for the users of the cloud infrastructure and platform to manage their services and resources deployed in cloud. For the moment it consist of open source implementation of a User Portal and Scripts. The User Portal is implemented in a form of a Web GUI following the same functionality as the OpenStack Dashboard. All about the implementation and the functionality of the OpenStack Dashboard can be found under the Horizon project. The Scripts facilitate direct approach to the underlying cloud resources through a command line. They are addressed to be used by administrators of the cloud.

5.2 User Portal User Guide

The User Portal offers a stand-alone open-source implementation of the OpenStack Dashboard. Initially it provides the same functionality as the OpenStack Dashboard. Some interactions include (create, delete, update) over the virtual resources (images, instances, flavors, services, etc.).

The User Portal is a Backbone-based Web client-side HTML5 application, implemented in JavaScript. The backbone model-view-controller methodology contributes for better dynamism and makes it a stand-alone client application with no need for Web server to interact with the backend. Instead, it aims to improving the user experience by using AJAX for dynamic rendering of the views. Moreover it contributes to improving the user-experience of the OpenStack portal by offering responsive design which makes it adaptable to multiple device screens (desktop, smart phone, tablet, etc), customizable object oriented CSS and internationalization i18n to support different languages. The User Portal functionality is tightly related to a JStack Library - a JavaScript equivalent of the OpenStack API that represent the Scripts Interface.

The design perspective of the User Portal is the following. There is one implementation that follows exactly the same functionality and design pattern of the OpenStack Dashboard. It is aimed for all use-case and third-party projects that want to use it as an independent component.

The other implementation is build upon the same methodology. However, apart from the basic functionality, it is designed to further extend the functionality of the OpenStack Dashboard and offer advanced interaction with the Service Manager GE, the Resource Manager GE, Object Storage GE and Cloud Proxy GE. Some of the new operations will include: creation and management of vApps, VDC, possibility to create elasticity rules and display monitoring statistics etc. The design would be changed according the new functionalities and style guide requirements of the FI-Ware project. The audience of this second implementation is the Cloud Hosting architecture adopters who would choose the User Portal as integral part of the architecture, to manage resources and platform components in coordination with the rest of the GEs.

In order to facilitate initial testing of the User Portal functionalities, there is a web server test-page established that runs the client application and permits petitions over a local basic
installation of the OpenStack infrastructure. A common log in user name: "admin" and password: "password" is fixed.
To be able to try the user portal individually over a different setup of the OpenStack infrastructure, one should download the code from the github page and run it locally or over a web server. However, additional fine-tuning is required in order for this to work properly, due to the lack of CORS implementation inside OpenStack Nova component, which prevent a direct access to Nova APIs from external domains.
After integration with the SM GE is finished, the user of the portal will be able to make the same operations to the infrastructure components, through the SM. Furthermore, additional functionalities will be leveraged by the SM for the user to inter-operate with the cloud resources and the PaaS GE.

5.3 Scripts User Guide

The Scripts Interface is a Node.js based package with a JavaScript implementation of OpenStack API called JStack API. Using this library a direct command line communication is enabled for more advanced users and administrators of the cloud infrastructure and platform.
To be able to try and run the scripts, the user should download and install the NPM package jstack-client with the command:

```
npm install jstack-client
```

You can also download the source code from this page and follow the given instructions.