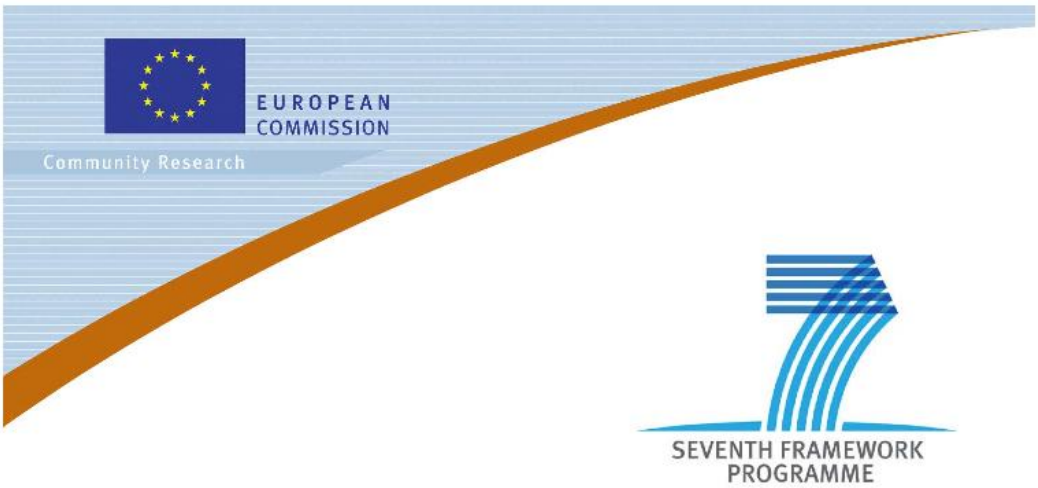


Private Public Partnership Project (PPP)

Large-scale Integrated Project (IP)

D.4.3.3: FI-WARE Installation and Administration 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.4.3.3
Project Document Date: 2014-07-31
Deliverable Type and Security: PU
Author: FI-WARE Consortium
Contributors: FI-WARE Consortium

1.1 Executive Summary

This document describes the installation and administration process of each Generic Enabler developed within in the "Cloud Hosting" chapter. The system requirements for the installation of a Generic Enabler are outlined with respect to necessary hardware, operating system and software. Each GE has a section dedicated to the software installation and configuration process as well as a section, which describes sanity check procedures for the system administrator to verify that the GE installation was successful.

1.2 About This Document

The "FI-WARE Installation and Administration Guide" comes along with the software implementation of components, each release of the document referring to the corresponding software release (as per D.x.3), to facilitate the users/adopters in the installation (if any) and administration of components (including configuration, if any).

1.3 Intended Audience

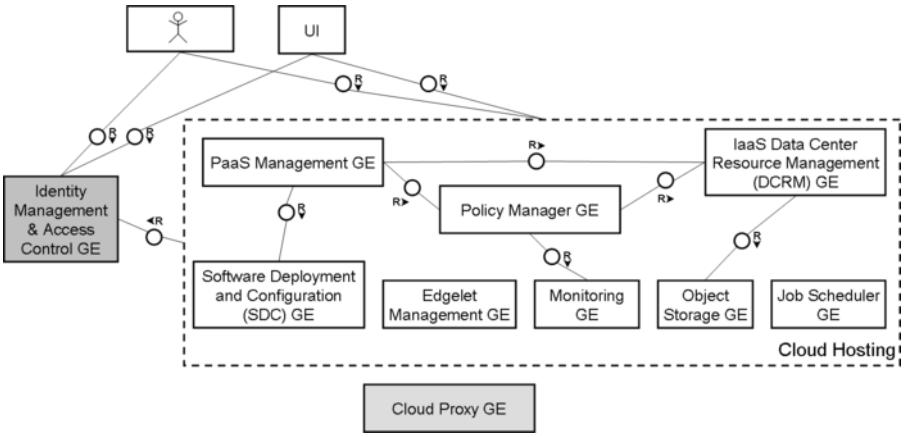
The document targets system administrators as well as system operation teams of FI-WARE Generic Enablers from the FI-WARE project.

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, as outlined in Materializing Cloud Hosting in FI-WARE.

The capabilities available in the second release of FI-WARE Cloud Hosting platform are outlined in Roadmap of Cloud Hosting.

The following diagram shows the main components (Generic Enablers) that comprise the second release of FI-WARE architecture.



The architecture comprises a set of Generic Enablers that together provide hosting capabilities of several kinds and at several levels of resource abstraction -- aiming at the needs of different applications hosted on the cloud platform. **IaaS Data Center Resource Management (DCRM) GE** is offering provisioning and life cycle management of virtualized resources (compute, storage, network) associated with **virtual machines**, which can run general purpose Operating Systems as well as arbitrary software stacks. Application developers and providers can use these virtual machines to develop and deploy their own software components that comprise their application stacks. **Object Storage GE** offers provisioning and life cycle management of **object**-based storage containers and elements, which can be efficiently used to store unstructured fixed content (such as images, videos, etc) as well as accompanying metadata. **Job Scheduler GE** offers the application to submit and manage computational jobs in a unified and scalable manner. **Edgelet Management GE** offers the capability to host lightweight application components, called *edgelets*, on devices typically located outside of the Data Center, such as those provided by the **Cloud Proxy GE** (developed jointly by the Cloud chapter and

the Interfaces to Network and Devices chapter). **Software Deployment and Configuration (SDC) GE** offers a flexible framework for installation and customization of software products within individual virtual machines. **Policy Manager GE** provides a framework for rule-based management of cloud resources, including application auto-scaling based leveraging metrics collected by **Monitoring GE**. Lastly, **PaaS Management GE** uses the above capabilities to offer holistic provisioning and ongoing management of complex workloads comprising sophistical combination of interdependent VMs and associated resources (such as multi-tier web applications or even complete custom-built PaaS environments), as well as configuration and management of software components within the VMs.
Each of the above GEs provides a REST API that can be used programmatically. The human actor represents the programmatic user of the different capabilities of the Cloud GEs via REST APIs. Moreover, the Cloud chapter provides a Web-based **Portal** (part of of the **UI** layer) , which surfaces main capabilities in an interactive manner --such as provisioning and monitoring of VM instances and services.
Cloud Hosting Generic Enablers are using the **Identity Management and Access Control** framework provided by the Security chapter, as outlined in the Cloud Security Architecture.

1.5 Structure of this Document

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

The following resources were used to generate this document:

D.4.3.3_Installation_and_Administration_Guide_front_page

IaaS Data Center Resource Management - Installation and Administration Guide

Object Storage - Installation and Administration Guide

PaaS Management - Installation and Administration Guide

Software Deployment And Configuration - Installation and Administration Guide

Policy Manager - Installation and Administration Guide

Monitoring - Installation and Administration Guide

Self-Service Interfaces - Installation and Administration Guide

Job Scheduler - Installation and Administration Guide

Edgelets - Installation and Administration 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 our wikis. 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 part of the links contained in the deliverables generated from wiki pages cannot be rendered to fully working links. This happens for instance when a wiki page references a section within the same wiki page (but there are other cases). In such scenarios we preserve a link for readability purposes but this points to an explanatory page, not the original target page.

In such cases where you find links that do not actually point to the original location, we encourage you to visit the source pages to get all the source information in its original form. Most of the links are however correct and this impacts a small fraction of those in our deliverables.

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.

```
http://[SERVER_URL]?filter=name:Simth*&index=20&limit=10
```

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, Thales, UPM, INRIA.

1.8 Keyword list

FI-WARE, PPP, Architecture Board, Steering Board, Roadmap, Reference Architecture, Generic Enabler, Open Specifications, I2ND, Cloud, IoT, Data/Context Management, Applications/Services Ecosystem, Delivery Framework , Security, Developers Community and Tools , ICT, es.Internet, Latin American Platforms, Cloud Edge, Cloud Proxy.

1.9 Changes History

Release	Major changes description	Date	Editor
v0	First draft	2013-07-24	IBM
v1	General review. Ready for delivery	2013-07-28	IBM

1.10 Table of Contents

2 IaaS Data Center Resource Management - Installation and Administration Guide

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

2.1 IaaS Data Center Resource Management Installation

This guide provides a step-by-step instructions to get DCRM installed and running on a machine, as well as a list of perquisites, requirements and possible troubleshooting that may occur during the installation.

2.1.1 Requirements

The instructions below were tested on the following environment, it may or may not be the same on other configurations.

- Minimal topology

5 physical nodes (servers):

host1 - Cloud Controller node: Keystone, Dashboard, Cinder, Pivot

host2 - Network node: Quantum server

host3 - Compute node: Nova, zk service, nfs server, Quantum agent

host4 - Compute node: Nova, zk service, nfs client, Quantum agent

host5 - Compute node: Nova, zk service, nfs client, Quantum agent

2.1.1.1 *Hardware Requirements*

The following table contains the recommended minimum resource requirements for each physical node running in the infrastructure:

Resource	**Requirement (for each)**
CPU	8 cores with at least 2.4 GHZ, VT-x enabled
Physical RAM	32GB
Disk Space	1TB The actual disk space depends on the amount of data being stored within the Repositories NoSQL database System.
Network	Two 1 Gbps Network Interface Card (NIC).

2.1.1.2 *Operating System Support*

- Ubuntu 12.04 (LTS) x86_64

2.1.1.3 *Software Requirements*

The following software is needed:

- MySql server - mandatory
- tgt (linux SCSI target user-space tool) - mandatory
- Open-iScsi (iSCSI implementation) - mandatory
- RabbitMQ server - mandatory
- KVM, libvirt (Virtualization software) - mandatory
- Apache server - mandatory
- LVM (Logical volume manager) - mandatory
- ntp (Network time protocol) - mandatory

2.1.1.4 *Storage Requirements*

All compute nodes should use shared storage. Mount the shared storage to the node's nova/instances folder so all VMs' disks are placed on it.

2.1.2 DCRM

2.1.2.1 *Install DCRM*

- Install OpenStack's Havana version on Ubuntu 12.04 - Precise Pangolin [1]
- Install Zookeeper (it is recommended to maintain three installations on various nodes) [2]
- Install client side python libraries on every nova node

```
sudo apt-get install python-zookeeper python-pip
sudo pip install evzookeeper
```

More details about Zookeeper for Openstack installation and configuration can be found here Zookeeper install and configure

- Install Shared storage

Set one of the environment's compute node as NFS server, and make sure all other compute nodes mount the shared storage and use it for their nova/instances folder ubuntu-nfs-install

- Download DCRM GE 3.3 Release files from CLOUD-DCRM:3.3 Files (password protected, available to the members of PPP programme only)

- On each of the nodes:

1. cd /usr/lib/python2.7/dist-packages
2. Extract nova.tar.gz files
3. Extract dove-neutron.tar under under neutron/plugins/
4. Run source delete_originals.sh

2.1.2.2 *Configure DCRM*

- Configure live migration support

Update libvirtd configuration and nova.conf. For more details follow the guide: configure-live-migration

- As for this release of DCRM, Pivot and Pulsar are mutually exclusive.

DCRM can be run with either or both:

1. Using advanced scheduler - PIVOT scheduler

2. Supporting dynamic over-commit based on VMs usage - PULSAR scheduler

The mode in which the system is run is set in the nova.conf configuration file. First, create a nova.conf file based on the common configuration below, next select either advanced scheduler mode (Pivot) or dynamic over-commit mode (Pulsar) by configuring the relevant component. The relevant configuration parameters for Pivot and Pulsar are provided in the installation and configuration section of each specific component.

Below is the base **common** nova.conf configuration file:

```
[DEFAULT]
logdir=/var/log/nova
logging_default_format_string
= %(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s %
(threadName)s/%(thread)d [-] %(instance)s%(message)s
logging_context_format_string
= %(asctime)s.%(msecs)03d %(levelname)s %(name)s %(threadName)
s/%(thread)d
[%(request_id)s %(user)s %(tenant)s] %(instance)s %(message)s
logging_exception_prefix = %(asctime)s.%(msecs)03d %(process)d
TRACE %(name)s %(threadName)s/%(thread)d %(instance)s
verbose=True
debug=True
state_path=/var/lib/nova
lock_path=/run/lock/nova
api_paste_config=/etc/nova/api-paste.ini
scheduler_driver=nova.scheduler.pivot_scheduler.PivotScheduler
rabbit_host= IP_ADDRESS
nova_url=http://IP_ADDRESS:8774/v1.1/
sql_connection=mysql://novaUser:PASSWORD@IP_ADDRESS/nova
root_helper=sudo nova-rootwrap /etc/nova/rootwrap.conf

# Auth
use_deprecated_auth=false
```

```
auth_strategy=keystone

# Imaging service
glance_api_servers=IP_ADDRESS:9292
image_service=nova.image.glance.GlanceImageService

# Vnc configuration
novnc_enabled=true
novncproxy_base_url=http://IP_ADDRESS:6080/vnc_auto.html
novncproxy_port=6080
vncserver_proxyclient_address=IP_ADDRESS
vncserver_listen=0.0.0.0

# Network settings
network_api_class=nova.network.quantumv2.api.API
quantum_url=http://10.10.10.51:9696
quantum_auth_strategy=keystone
quantum_admin_tenant_name=service
quantum_admin_username=quantum
quantum_admin_password=service_pass
quantum_admin_auth_url=http://10.10.10.51:35357/v2.0
libvirt_vif_driver =
nova.virt.libvirt.vif.QuantumDOVEVIFDriver
linuxnet_interface_driver=nova.network.linux_net.LinuxOVSInter
faceDriver
#If you want Quantum + Nova Security groups
firewall_driver=nova.virt.firewall.NoopFirewallDriver
security_group_api=quantum
#If you want Nova Security groups only, comment the two lines
above and uncomment line -1-.
#-1-
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDri
ver

#Metadata
service_quantum_metadata_proxy = True
quantum_metadata_proxy_shared_secret = helloOpenStack

# Compute #
```

```
compute_driver=libvirt.LibvirtDriver


# Cinder #
volume_api_class=nova.volume.cinder.API
osapi_volume_listen_port=5900


rpc_response_timeout=600
service_down_time=60
zk_servers=IP_ADDRESS:2181 # Comma separated zk servers ip
addresses:port.

#(e.g.zk_servers=9.148.26.140:2181,9.148.26.141:2181,9.148.26.
142:2181)
svcgroup_backend=zk
scheduler_monitoring_enabled=True
```

2.1.3 ResourceManager Advanced Placement : Pivot

PIVOT is an advanced placement manager for clouds capable of deploying, optimizing and relocating virtual machines under various constraints while globally optimizing the cloud utilization. PIVOT both produces plans for deploying and migrating virtual machines and handles their safe execution.

2.1.3.1 ***Install Software***

PIVOT is a Java application and requires a JRE.

- Download and install IBM JRE 1.6 from [3]

After installation, make sure java bin files are in the specified in PATH.

- Copy and extract PIVOT.tar.gz from CLOUD-DCRM:3.3 Files into /opt/IBM/
- Copy and extract cplex.tar.gz from CLOUD-DCRM:3.3 Files into /opt/IBM/

- Create Pivot's log directory

```
mkdir /var/log/PIVOT/
Grant the user running Openstack services with full access
```

- Create a symbolic link in /opt/IBM/PIVOT/

```
ln -s /var/log/PIVOT/ logs
```

- New metrics table are needed for both Pulsar and Pivot. Make sure the table instance_stats exists, if not run the db migration script from nova/db/sqlalchemy/migrate_repo as root:

```
python manage.py upgrade
mysql://<user>:<password>@localhost:3306/nova
```

2.1.3.2 *Configure Pivot*

- JRE path should point to the installed JRE

To enable the Pivot advanced scheduler set the following parameters in nova.conf:

```
scheduler_driver=nova.scheduler.pivot_scheduler.PivotScheduler
pivot_address=127.0.1.1

#Configuration can be fined tuned to your needs by changing
the relevant parameter:
scheduler_ongoing_enabled=True
scheduler_ongoing_tick=10
```

Pivot-related additional parameters in nova.conf have the following meaning:

- scheduler_ongoing_enabled parameter sets whether a periodic ongoing optimization task is enabled or disabled (True/False)
- scheduler_ongoing_tick parameter sets the interval in ticks (~ one minute) in which the periodic ongoing optimization task will be invoked (int)

2.1.4 ResourceManager Advanced Network : OpenDayLight OpenDove

2.1.4.1 *Install OpenDayLight Dove*

This section describes the installation process for OpenDove on a Havana OpenStack cluster installed with ubuntu packages.
You will need to re-install 4 components: OpenDayLight OpenDove (oDMC, oDCS, oDGW, ovs-agent) , NOVA, QuantumClient, Quantum and Quantum-agent.

On each compute node you will need to install OpenVswitch, Dove-agent and re-run nova-compute after applying DOVE patch to nova-compute.

On the control node you will need to re-install the quantum-server and qunatumclient.
Installing the ODL Controller Node (with ODMC and ODCS)

Official installation instructions for the ODL controller are here: https://wiki.opendaylight.org/view/OpenDaylight_Controller:Installation

ODL Controller runs inside JVM so any system capable of running JVM 1.7 with JDK can be used. Install jdk1.7

```
sudo apt-get install openjdk-7-jdk
```

Install maven

```
sudo apt-get install maven
```

Install make

```
sudo apt-get install make
```

Export environment variables (also through .profile or .bashrc). Note that JAVA_HOME can be different in your environment.

```
export ODL_USERNAME=username
export JAVA_HOME="/usr/lib/jvm/java-7-openjdk-amd64"
export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=256m"
export PATH="$JAVA_HOME/bin:$PATH"
```

Pulling/building ODL controller

Pull the controller code with your developer credentials:

```
git clone
ssh://${ODL_USERNAME}@git.opendaylight.org:29418/controller.gi
t
```

Or without such credentials:

```
git clone https://git.opendaylight.org/gerrit/p/controller.git
```

Build and install controller; first time build takes ~10 minutes (up to 30 minutes)

```
cd controller; mvn clean install
```

-
  - Make sure that the node is configured to allow communications on ports 8023, 8080, and 6633; make sure no other software is binding these ports as destination ports.

Run the controller

```
cd
./opendaylight/distribution/opendaylight/target/distribution.o
pendaylight-osgipackage/opendaylight
./run.sh
```

-
  - This screen will show the osgi> prompt when ready

Login to the controller UI

-
  - Point the browser to http://<controller ip>:8080
  - login with admin/admin

Pulling opendove

Pull the opendove code

```
git clone
ssh://${ODL_USERNAME}@git.opendaylight.org:29418/opendove.git
```

Building/starting the ODMC

Build the odmc code:

```
cd opendove; mvn clean install
```

Install the following odmc bundles without starting them, either using the OSGi UI, or using the OSGi CLI (“install file://<full_path>”):

- odmc/target/odmc-0.5.2-SNAPSHOT.jar
- odmc/implementation/target/odmc.implementation-0.5.2-SNAPSHOT.jar
- odmc/rest/target/odmc.rest-0.5.2-SNAPSHOT.jar
- odmc/rest/northbound/target/odmc.rest.northbound-0.5.2-SNAPSHOT.jar
- odmc/rest/southbound/target/odmc.rest.southbound-0.5.2-SNAPSHOT.jar

Start the odmc bundles, in this order (!), using the UI or the CLI ("start <bundle_name>"):

- odmc.implementation-0.5.2-SNAPSHOT.jar
- odmc.rest.northbound-0.5.2-SNAPSHOT.jar
- odmc.rest.southbound-0.5.2-SNAPSHOT.jar

Verification: Try using the ODMC southbound API, either using curl command (from shell) or by pointing your browser to the URL. [API is documented in https://jenkins.opendaylight.org/opendove/job/opendove-merge/lastSuccessfulBuild/artifact/odmc/rest/northbound/target/enunciate/build/docs/rest/index.html]

List registered service appliances (nb):

```
curl --user "admin":"admin" -X GET -v
http://127.0.0.1:8080/controller/nb/v2/opendove/odmc/serviceAp
pliances
```

List registered switches (nb):

```
curl --user "admin":"admin" -X GET -v
http://127.0.0.1:8080/controller/nb/v2/opendove/odmc/switches
```

Get ODCS cluster leader (sb):

```
curl -v --user "admin":"admin" -X GET
http://127.0.0.1:8080/controller/sb/v2/opendove/odmc/odcs/lead
er
```

Building/starting the ODCS

Install prerequisites:

```
make, python-dev, libevent-dev, libjansson-dev
```

Build the odcs code:

```
cd opendove/odcs; make
```

Start the odcs process:

```
sudo ./build/dcs_server.py 902

 (password=Hook&Ladder)
```

Configure the ODMC location:

```
 controller_add 1.2.3.4 8080
```

At this point ODCS should have been registered with the ODMC; test it by running the following APIs [API is documented in https://jenkins.opendaylight.org/opendove/job/opendove-merge/lastSuccessfulBuild/artifact/odmc/rest/northbound/target/enunciate/build/docs/

rest/index.html] List registered service appliances (nb) - should show appliance instance with no DCS role (isDCS flag is false).

```
 curl --user "admin":"admin" -X GET -v
http://127.0.0.1:8080/controller/nb/v2/opendove/odmc/serviceAp
pliances
```

Get ODCS cluster leader (sb) - should be empty at this point:

```
 curl -v --user "admin":"admin" -X GET
http://127.0.0.1:8080/controller/sb/v2/opendove/odmc/odcs/lead
er
```

Assign the ODCS appliance role at ODMC:

```
 curl --user "admin":"admin" -H "Accept: application/json" -H
"Content-type: application/json" -X PUT -v
http://127.0.0.1:8080/controller/nb/v2/opendove/odmc/odcs/<uui
d of odcs taken from serviceAppliance reply>/role -d
'{"service_appliance": {"isDCS": true}}'
```

At this point the http://127.0.0.1:8080/controller/nb/v2/opendove/odmc/serviceAppliances and http://127.0.0.1:8080/controller/sb/v2/opendove/odmc/odcs/leader commands should show the ODCS.

Configure oDGW with oDMC address

Start oDGW with build/dove_gateway.py from the dgadmin directory log in with password "admin" Point oDGW at oDMC with "service dmcadd <odmc address> 8080"

Enable oDGW Role

Issue a GET request to

```
http://<odmc_address>:8080/controller/nb/v2/opendove/odmc/serv
iceAppliances
```

In response, copy out the uuid of the oDGW and issue a PUT request to

```
http://<odmc_address>:8080/controller/nb/v2/opendove/odmc/odgw
/<uuid>/role
```

with the following JSON body:

```
 { "service_appliance": { "isDGW": true } }
```

Configure oDGW Dove Tunnel Interface

Issue a PUT request to http://<odmc_address>:8080/controller/nb/v2/opendove/odmc/odgw/<uuid>/ipv4 with the following JSON body:

```
 { "gw_ipv4_assignment": {  "ip": "<tunnel_ip>", "nexthop":
"<nexthop for tunnel>", "mask": "<mask for tunnel>", "vlan":
0, "intf_type":
   "dovetunnel" } }
```

At oDGW verify the address has been added with "service show"

Start OVS-Agent

At each host with OVS and the OVS-Agent installed, start the OVS-Agent with:

```
 sudo ./setup-dove.sh <compute node ip> <controller ip>:8080
1234
```

2.1.5 Quantum server installation

- Edit quantum.conf:

```
Change core_plugin to DOVE plugin:

core_plugin =
quantum.plugins.dove.dove_quantum_plugin.DOVEQuantumPluginV2
```

- Edit dove configuration file dove_quantum_plugin.ini and set DMC IP:

```
point to the mysql database (get it from the existing plugin
config file (e.g
/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini)

sql_connection =
mysql://quantumUser:passw0rd@10.10.10.51/quantum

# DOVE controller IP

[DMC]

url = <dmc_ip>

local_ip = <local IP>
```

- Copy dove configuration file into /etc/quantum/plugins/dove/ directory:

```
$ cd <quantum source dir>

$ cp etc/quantum/plugins/dove/dove_quantum_plugin.ini
/etc/quantum/plugins/dove/
```

- Edit /etc/default/quantum-server:

QUANTUM_PLUGIN_CONFIG="/etc/quantum/plugins/dove/dove_quantum_plugin.ini"

- Patch quantum source code with DOVE patch OR replace all quantum code to the DOVE patched source code

(/usr/lib/python2.7/dist-packages/quantum)

- Restart quantum-server service:

```
$ service quantum-server restart
```

Compute node

Openvswitch installation:

- Remove any OpenVswitch packages installed on host
- Install Openvswitch from origin repository:

```
$ git clone git://git.openvswitch.org/openvswitch

$ cd openvswitch

$ git checkout branch-2.0

$ vi INSTALL
```

- Install Openvswitch according to the instructions in the INSTALL file
- Initialize the ovsdb:

```
$ mkdir -p /usr/local/etc/openvswitch
$ ovsdb-tool create /usr/local/etc/openvswitch/conf.db
vswitchd/vswitch.ovsschema
```

- Create links to the Openvswitch executables:

```
$ ln -s /usr/local/bin/ovs-vsctl /usr/bin/ovs-vsctl
$ ln -s /usr/local/bin/ovs-ofctl /usr/bin/ovs-ofctl
$ ln -s /usr/local/bin/ovs-appctl /usr/bin/ovs-appctl
```

- Re-run nova-compute service:

```
$ service nova-compute restart
```

2.1.5.1 *Configure Dove*

Update /etc/quantum/quantum.conf:

```
   core_plugin =
quantum.plugins.dove.dove_quantum_plugin.DOVEQuantumPluginV2
```

Edit dove configuration file dove_quantum_plugin.ini and set DMC IP:

```
   # point to the mysql database
   sql_connection =
mysql://quantumUser:passw0rd@10.10.10.51/quantum


   # DOVE controller IP
   [DMC]
   url = <dmc_ip>
```

Update /etc/nova/nova.conf:

```
      quantum_use_dhcp=False
      libvirt_vif_driver =
nova.virt.libvirt.vif.QuantumDOVEVIFDriver
      Comment out security groups:
      ##security_group_api=quantum
```

2.1.6 ResourceManager Advanced Capacity Manager : IBM Adaptive Utilization Accelerator for Virtualized Environments (IBM PULSAR)

2.1.6.1 *Install IBM Pulsar*

New metrics table are needed for both Pulsar and Pivot. Make sure the table instance_stats exists, if not run the db migration script from nova/db/sqlalchemy/migrate_repo as root:

```
python manage.py upgrade
mysql://<user>:<password>@localhost:3306/nova
```

2.1.6.2 *Configure IBM Pulsar*

Below is an example nova.conf that enables dynamic overcommit with IBM Pulsar.

NOTE: for Pulsar to work properly nova.conf parameters must have the **same value** in both the scheduler node and **each of the compute hosts** nova.conf files.

```
scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter,UtilizationFilter
least_cost_functions=nova.scheduler.least_cost.compute_balance_performance_cost_fn
#
cpu_low_util_threshold=0.09
vcpu_throttle_ratio=10.0
cpu_allocation_ratio=1.0
ram_allocation_ratio=1.0
scheduler_ongoing_enabled=False
scheduler_monitoring_enabled=True
scheduler_pulsar_enabled=True
```

This is a brief explanation of the parameters relevant to IBM Pulsar in nova.conf:

- cpu_low_util_threshold sets under what level of utilization the VM is considered to be idle (float [0,1])
- vcpu_throttle_ratio sets the ratio by which idle VM resources will be reduced (float > 0)
- cpu_allocation_ratio sets virtual CPU to physical CPU allocation ratio (over-commit ratio, int > 0)
- ram_allocation_ratio sets virtual RAM to physical RAM allocation ratio (over-commit ratio, int > 0)
- scheduler_ongoing_enabled should be FALSE when running IBM Pulsar

- scheduler_monitoring_enabled should be TRUE when running IBM Pulsar
- scheduler_pulsar_enabled sets whether IBM Pulsar is enabled (True/False)

2.1.7 Start Services

- Start Zookeeper service on all installed nodes: /usr/share/zookeeper/bin/zkServer.sh start
- On each node, start the relevant Openstack services (according to "Minimal topology" section):

```
service SERVICE_NAME restart
```

-
  - examples:
    - service nova-api restart
    - service nova-scheduler restart
    - service nova-compute restart
    - service nova-conductor restart
    - service cinder-api restart
    - service cinder-scheduler restart
    - service cinder-volume restart
    - service quantum-server restart

2.2 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

```
Check Openstack installation:
        From the cli run nova service-list (run source openrc
before)
        make sure all services are registered, enabled and up)


+------------------+-------------+----------+---------+-------
+----------------------------+
| Binary           | Host        | Zone     | Status  | State
| Updated_at                 |
+------------------+-------------+----------+---------+-------
+----------------------------+
| nova-cert        | HOST_NAME   | internal | enabled | up
| 2013-07-11T10:39:39.000000 |
| nova-compute     | HOST_NAME   | nova     | enabled | up
| 2013-07-11T10:39:49.000000 |
```

```
| nova-compute     | HOST_NAME   | nova     | enabled | up
| 2013-07-11T10:39:44.000000 |
| nova-conductor   | HOST_NAME   | internal | enabled | up
| 2013-07-11T10:39:42.000000 |
| nova-consoleauth | HOST_NAME   | internal | enabled | up
| 2013-07-11T10:39:47.000000 |
| nova-scheduler   | HOST_NAME   | internal | enabled | up
| 2013-07-11T10:39:42.000000 |
+------------------+-------------+----------+---------+-------
+----------------------------+



Check Zookeeper installation:
From other host check that you can connect to the zk server
using cli
        /usr/share/zookeeper/bin/zkCli.sh
        in the cli prompt run:
        connect <SERVER_IP>:<PORT>    (the default port is
2181)
        verify you can see zk filesystem by running ls /


Check DOVE installation:
# sudo ovs-vsctl show
    Should look like this: <---
    8908804d-f2c2-41c0-85da-be5c93e480d0
    Bridge br-int
        Controller "tcp:172.25.8.241"
            is_connected: true
        fail_mode: secure
        Port dove
            Interface dove
                type: vxlan
                options: {dst_port="8472", key=flow,
remote_ip=flow}
        Port br-int
            Interface br-int
                type: internal
     ---->
 #sudo ovs-ofctl show br-int
     Should look like this: <---
```

```
    OFPT_FEATURES_REPLY (xid=0x2): dpid:0000566e572e264b
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS
ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN
SET_DL_SRC SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS
SET_TP_SRC SET_TP_DST ENQUEUE
 1(dove): addr:a6:1f:6b:5d:e8:fd
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-int): addr:56:6e:57:2e:26:4b
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
 ---->


 On each compute node run the dove controller:
  $ screen -S utilities/dove-controller ptcp: -d <dcs_ip>


 Check if new Dove API exist:
 $ quantum help
 - dove-policy-* API should be listed
```

```
Check PIVOT installation:


java -version


Verify valid response:
java version "1.6.0"
Java(TM) SE Runtime Environment (build pxa6460sr8fp1-
20100624_01(SR8 FP1))
```

```
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 Linux amd64-64
jvmxa6460sr8ifx-20100609_59383 (JIT enabled, AOT enabled)


Verify /opt/IBM/PIVOT and /opt/IBM/ILOG exist
```

2.2.1 End to End testing

Log in with the YOUR_USER_NAME and YOUR_PASSWORD to the Dashboard endpoint (HORIZON http web interface), usually located at http://dashboard and examine the various sections in the admin tab to make sure all instances, images and services are well defined and accessible.

Check registered services and their state:

```
    nova-manage service list
```

Make sure at least three nova-compute are running (have (: symbol for state), nova-volume, nova-scheduler, nova-network, nova-consoleauth, nova-cert

The next step is to try and create a VM, as this spans most of the components, once the VM has successfully created, it is a good indication for the system health.

From the command line run

```
  nova image-list
```

Results should contain at least one row

```


+----+---------------------------------------+--------+--------
+
| ID |                  Name                 | Status | Server
|
+----+---------------------------------------+--------+--------
+
| 14 | oneiric-server-cloudimg-amd64-kernel | ACTIVE |
|
| 15 | oneiric-server-cloudimg-amd64        | ACTIVE |
|
+----+---------------------------------------+--------+--------
+


```

Use the ID returned, for example in the above case 15, and execute

```
  $ nova boot --image 15 --flavor 2 test
```

A new VM with the name test should be created, login to the dashboard/use the following command to check it's state

```
  $ nova list


+------+---------------+--------+----------------------+
|  ID  |      Name     | Status |       Networks       |
```

```
+------+---------------+--------+---------------------+
| 1426 | test          | ACTIVE | internet=8.22.27.251 |
+------+---------------+--------+---------------------+
```

2.2.1.1 *Advanced scheduling testing*

Various testing scenarios that will help you make sure everything is working as desired

2.2.1.1.1 *Setup*

For this testing setup, you will need two similar hosts used as compute nodes, each with 8 cores and about 8-16 GB of memory

2.2.1.1.2 *Multi-deploy Scenario*

1. Start with both hosts being empty
2. Define a flavor consisting of 1 core of CPU and about 1/2 the memory capacity of one host
3. Run multi-deploy of 6 instances. It should succeed, with 2-4 instances being actually deployed, depending on the memory margins.
4. Remove all instances
5. Define a flavor consisting of 1.5 times of the memory capacity of one host
6. Run multi-deploy of 2 instances.

It should fail, with no instance being deployed.

2.2.1.1.3 *Placement Groups Scenario*

1. Start with both hosts being empty
2. Define a flavor consisting of 1 core and 1/10 of the memory capacity of one host
3. Deploy 3 instances of the above flavor using the same placement group

The first two deployments should succeed with instances going each to a different host. The third deployment should fail.

2.2.1.1.4 *Emergency Evacuation (VM Keeper) Scenario*

1. Start with both hosts being empty
2. Define a flavor consisting of 1 core and about 1/3 the memory capacity of one host
3. Do multi-deploy of 2 instances. They should be spread one on each host.
4. Shut down one host, and wait.

See that the instance on that host is now being rebuilt on the other host.

2.2.1.1.5 *Optimization Scenario*

1. Start with both hosts being empty
2. Define a flavor consisting of 8 cores and 1/4 the memory capacity of one host
3. Deploy 2 instances, using a placement group. They should be spread one on each host.

4. Deploy a third instance on either of the hosts, but without using the placement group. Note that only the third instance is movable, so it will now be referred to as "the ball". Each of the un-movable instances will be referred to as "a player".
5. Stress the ball instance. See instructions how to stress an instance below.
6. In the host with the ball instance, stress the player instance.
7. Run optimize using the instructions from the user guide
8. See that the ball is being moved to the other host
9. Stop stressing the player in one host and start stressing the player in the other host (where the ball is now).

Run stress and see that the ball is moved back to the first host. Repeat as much as you like.

Stress: for stressing an instance with N cores, run "stress -c N" in a command shell. Top stop stressing, simply kill the resulting process.

2.2.2 List of Running Processes

- /usr/sbin/apace2
- /usr/sbin/mysqld
- /usr/lib/erlang/erts-5.8.5/bin/epmd -daemon
- /usr/bin/java * com.ibm.pivot.PivotUDPServer (if running Pivot)
- /usr/bin/java * org.apache.zookeeper.server.quorum.QuorumPeerMain
- libvirt
- kvm

- Several nova related process should be running according to your topology:

```
   ps aux | grep "/usr/bin/nova" | grep -v grep | awk '{print
$12}'
```

  -
    - /usr/bin/nova-novncproxy
    - /usr/bin/nova-consoleauth
    - /usr/bin/nova-cert
    - /usr/bin/nova-api
    - /usr/bin/nova-scheduler
    - /usr/bin/nova-objectstore
    - /usr/bin/nova-compute
    - /usr/bin/nova-conductor
    - /usr/bin/cinder-api
    - /usr/bin/cinder-scheduler
    - /usr/bin/cinder-volume

- /usr/bin/quantum-server

2.2.3 Network interfaces Up & Open

Default ports for OpenStack projects and components: Horizon : 80 (WSGI), Keystone : 5000, Nova URL : 8774, glance : 9292, ec2 url: 8773, vncproxy:6080, libvirt migration : 16509, Zookeeper: 2181

The OCCI interface itself will start on port 8787 by default. It is mentioned however that port 35357 should be open as well. The later is the port for the authentication service Keystone

2.2.4 Databases

The last step in the sanity check, once that we have identified the processes and ports is to check the different databases that have to be up and accept queries.

Nova DB is installed on only one of the nodes (usually the controller node) and all other services connect to it remotely. To make sure the DB is installed and is up running, do the following:

```
  cat /etc/nova/nova.conf | grep sql_connection
```

This will provide you with the information of the DB node IP, MYSQL_USER and MYSQL_PASS use them to execute the next command

```
sudo mysql -uroot -p$MYSQL_PASS nova -e 'SELECT * FROM
services;'
```

If installed correctly, you will see something similar to the following (ignore the time and hosts fields)

```
--------+-------------+----------+---------+

| created_at          | updated_at          | deleted_at | id
| host        | binary           | topic       | report_count
| disabled | deleted |

+---------------------+---------------------+------------+----
+-------------+-----------------+-------------+--------------
+----------+---------+

| 2013-05-26 08:50:11 | 2013-07-11 10:53:53 | NULL       |  1
| server.. | nova-cert        | cert        |       395987 |
0 |       0 |

| 2013-05-26 08:50:11 | 2013-07-11 10:53:56 | NULL       |  2
| server.. | nova-conductor   | conductor   |       395124 |
0 |       0 |

| 2013-05-26 08:50:11 | 2013-07-11 10:53:51 | NULL       |  3
| server.. | nova-consoleauth | consoleauth |       395982 |
0 |       0 |

| 2013-05-26 08:50:11 | 2013-07-11 10:39:52 | NULL       |  4
| server.. | nova-scheduler   | scheduler   |       387150 |
0 |       0 |
```

```
| 2013-05-26 12:49:34 | 2013-07-11 10:53:54 | NULL        |  5
| server.. | nova-compute     | compute     |       383984 |
0 |       0 |
| 2013-06-12 15:05:44 | 2013-07-11 10:53:59 | NULL        |  6
| server.. | nova-compute     | compute     |       234934 |
0 |       0 |
+---------------------+---------------------+-------------+----
+-------------+--------


 sudo mysql -uroot -p$MYSQL_PASS cinder -e 'SELECT * FROM
services;'
-----+-----------------+-------------+----------+-----------
--------+
| created_at          | updated_at          | deleted_at |
deleted | id | host        | binary           | topic
| report_count | disabled | availability_zone |
+---------------------+---------------------+-------------+----
-----+----+-------------+-----------------+-----------------
+--------------+----------+-------------------+
| 2013-05-26 09:10:14 | 2013-07-11 10:54:32 | NULL        |
0 |  1 | os-server-1 | cinder-scheduler | cinder-scheduler |
395868 |        0 | nova              |
| 2013-05-26 09:10:14 | 2013-07-11 10:54:32 | NULL        |
0 |  2 | os-server-1 | cinder-volume    | cinder-volume    |
395828 |        0 | nova              |
+---------------------+---------------------+-------------+----
-----+----+---------
```

To make sure Pulsar was configured, execute

```
 describe instances;
```

This will show all instances table columns defined, the last two should be

```


| low_util                | tinyint(1)    | YES  |     | 0
|
| vcpu_throttle_ratio     | float         | YES  |     | 1
|
```

To make sure DOVE DB was configured, log in to DOVE node and execute

```
sudo mysql -uroot -p$MYSQL_PASS dove_quantum 'SHOW TABLES;'
```

If it returns errors/no tables lines are return, please go back to DOVE install&configure section and reinstall

2.3 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution.

In case of a problem, check /var/log/nova/*.log for records of type ERROR or EXCEPTION to check what is the source of the problem. Additional Openstack troubleshooting procedures are described here: [4]

If you are running with the advanced scheduler and unknown error occurred, check /var/log/PIVOT/*.log for exceptions. Next, try to restart the scheduler service. If the error persists, send PIVOT logs and nova-scheduler.log* to the IBM's development team.

Make sure all the services are running using nova-manage service list command. In case one of the services listed above is not running or with :(state, restart the service with the same commands listed in Start DCRM.

In case the services appear as down, make sure zookeeper has been started before the nova services have. If they havent - restart the services.

2.3.1 Resource availability

The resource availability should be at least 1GB of RAM and 6GB of Hard disk in order to prevent enabler's bad performance. This means that bellow these thresholds the enabler is likely to experience problems or bad performance.

2.3.2 Remote Service Access

We need to make sure all components are accessible: the OpenStack API, DOVE, PIVOT and Swift.

At keystone node make sure keystone is accessible:

```
    keystone catalog --service ec2
```

The response is the list of access URLs

```
    Service: ec2
+-------------+-----------------------------------------+
|   Property  |                 Value                   |
+-------------+-----------------------------------------+
| adminURL    | http://URL/services/Admin |
| internalURL | http://URL/services/Cloud |
| publicURL   | http://URL/services/Cloud |
```

```
| region      | RegionOne                                |
+-------------+------------------------------------------+
```

At Glance node make sure glance is accessible:

```
    glance index
```

The response is the list of images currently available and accessed from Glance

```
ID                                   Name
Disk Format          Container Format     Size
------------------------------------ ------------------------
----- --------------------

0da554a8-ea83-459e-a013-7442ca346f00
Ubuntu_12.04_cloudimg_amd64    qcow2                ovf
4752474112
```

At a Swift node make sure Swift is accessible:

```
    swift list
```

To verify OpenStack API is accessible (Apache application) acquire a valid token from the OpenStack Keystone server. Using curl, the following command can be used:

```
   curl -d
'{"auth":{"passwordCredentials":{"username":"<Insert Openstack
username here>", "password":"<Insert Openstack password
here>"}}}'

   -H "Content-type: application/json" http://<Insert IP
address of Keystone node here>:5000/v2.0/tokens
```

The resulting Keystone Token is returned near the beginning of the response

```
    {"access": {"token": {"expires": "2012-06-22T07:50:54Z",
"id": "d71c70e2d0834d4baaa7ec9f2b94b7ca", "tenant":
{"enabled": true, "id": "a4f4eb48f31c447e84606368a7193c9c",
"name": "demo", "description": null}}, "serviceCatalog":
[{"endpoints": [{"adminURL":
"http://192.168.255.135:8774/v2/a4f4eb48f31c447e84606368a7193c
9c", "region": "RegionOne", "publicURL":
"http://192.168.255.135:8774/v2/a4f4eb48f31c447e84606368a7193c
9c",
```

Use the value of the first id field as TOKEN, and publicURL as the URL, and execute

```
curl -v -X GET -H 'X-Auth-Token:TOKEN' -H "Content-type:
application/json" URL/servers/detail
```

several Nova Compute nodes should be returned as a response.

2.3.3 Resource consumption

The DCRM component manages the VMs and the hosts. Resource consumption can get quite high as more VMs are started and is depended on the VMs nominal resources requested (i.e. the VM's flavor) and more importantly, the VMs utilization (i.e. actual resources in used by the workload). There is no general assumption on the default consumption and it varies from one deployment to another.

In nova.conf overcommit and quota parameters can be configured to limit (as for the moment) Compute nodes' amount of Virtual Machines scheduled and the nominal to physical resources ratio (for RAM and CPU), which limit in a way the resources consumption.

The resource consumption depends on the number of VMs deployed and running, as well as the resources' over-commit ratio enabled for the system. All instances disks are on shared storage. Therefore, it is important to use at least a 1 TB of space in the shared storage, and make sure it's extendable.

The next table is an effort to state the amount of resources that are abnormally high or low (for RAM, CPU and I/O) and should alert the administrator of a problem/need to add more resources. The data is applied to the System Requirement section from the beginning of this guide.

Resource Consumption for non Compute node		
	Low Usage	**High Usage**
RAM	3,2GB ~ 40%	7,83GB ~ 98%
CPU	10% (of a 2500MHz core)	90% (of a 2500MHZ core)
HDD (local storage)	30%	80%

Resource Consumption for Compute node		
	Low Usage	**High Usage**
RAM	3,2GB ~ 40%	6,4GB ~ 80%
CPU	10% of a core	90% of a core
HDD (local)	30%	90%
HDD (shared)	15%	70%

2.3.4 I/O flows

Clients access the DCRM Interface through the client's Web Browser through the Dashboard component (an Apache application) using HTTP protocols to the controller component (OpenStack API) which forward the request to the relevant component (e.g. DOVE, PIVOT, etc.).

3 Object Storage - Installation and Administration Guide

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

3.1 Introduction

Welcome to the Installation and Administration Guide for the Object Storage Generic Enabler. This generic enabler is built on an Open Source project - the CDMI Interface for OpenStack Swift - 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.

This generic enabler requires installation of OpenStack[1] cloud management software (specifically the Swift object storage component). The Cloud Data Management Interface (CDMI) [2] API for OpenStack Swift is then installed on top of it.

3.2 System Installation

The following steps need to be performed to get the CDMI interface for OpenStack up & running:

```
1. Install OpenStack with Swift according to the Instructions
from OpenStack[3]

2. Install the CDMI interface from GitHub[4] using the command
'python setup.py install'

3. Configure the Swift proxy server.
```

To configure the proxy server edit the file /etc/swift/proxy-server.conf and add the "cdmi" filter before "proxy-server"

```
[pipeline:main]

pipeline = healthcheck cache tempauth *cdmi* proxy-server
```

Also add the following section to the same file:

```
[filter:cdmi]

use = egg:cdmiapi#cdmiapp
```

3.3 System Administration

There are no system administration tasks specific for the CDMI Interface of OpenStack Swift. This software is essentially a pass-through proxy that translates any invocations and delegates them to an underlying OpenStack Swift installation. Please refer to the OpenStack Swift documentation at http://swift.openstack.org for complete information on how to administer an OpenStack Swift deployment.

3.4 Sanity Check Procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of

tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

3.4.1 End to End testing

Please note that the following information is required before carrying out this procedure:

- the IP address of the CDMI node (e.g. on the FI-WARE testbed this is cdmiservice.lab.fi-ware.eu)
- the IP address of the Openstack Keystone node managing security for the Object Storage GE Deployment (e.g. on the FI-WARE testbed this is 130.206.80.100)
- a valid OpenStack (keystone) username and password

1. Verify that **http://cdmiservice.lab.fi-ware.eu:8080/cdmi** can be reached. By default, web access will receive a "401 Unauthorized" response.

2. Acquire a valid token from the OpenStack Keystone server. Using curl, the following command can be used:

```
   curl -d
'{"auth":{"passwordCredentials":{"username":"<Insert Openstack
username here>", "password":"<Insert Openstack password
here>"}}}' -H "Content-type: application/json" http://<Insert
IP address of Keystone node here>:5000/v2.0/tokens
```

The resulting Keystone Token is returned near the beginning of the response.

3. Verify that you can retrieve the capabilities of the root Object container:

```
   curl -v -X GET -H 'X-Auth-Token: <Insert Keystone Token
here>' -H 'X-CDMI-Specification-Version: 1.0.1'
http://cdmiservice.lab.fi-
ware.eu:8080/cdmi/cdmi_capabilities/AUTH_<tenant_id>/
```

Sample output of this command:

```
 > GET
/cdmi/cdmi_capabilities/AUTH_d418851c6d294381bbe6e082849686d6/
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: 130.206.80.102:8080

 > Accept: */*

 > X-Auth-Token: e40f876706d1470facf783a5ce656d63

 > X-CDMI-Specification-Version: 1.0.1

 >

 < HTTP/1.1 200 OK

 < Content-Type: application/cdmi-capability

 < X-Cdmi-Specification-Version: 1.0.1

 < Content-Length: 200
```

```
< X-Trans-Id: txe226c68ba16a481b9f5c6838006e4b50
< Date: Mon, 29 Oct 2012 15:04:35 GMT
<
{
  "capabilities": {
   "cdmi_list_children": true,
    "cdmi_create_container": true
  },
  "objectName": "AUTH_d418851c6d294381bbe6e082849686d6",
  "objectType": "application/cdmi-capability"
}
```

4. Create a test container:

```
   curl -v -X PUT -H 'X-Auth-Token: <Insert Keystone Token
here>' -H 'Content-tType: application/directory'-H 'Content-
Length: 0' http://cdmiservice.lab.fi-
ware.eu:8080/cdmi/AUTH_<tenant_id>/<container_name>/
```

Expected output:

```
> PUT
/v1/AUTH_288e7938013a48529a62a30d62edf0cc/test_container
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: 192.168.0.10:8080
> Accept: */*
> X-Auth-Token: 87efddec36c34e8da15fd2909ca7e8cd
> Content-tType: application/directory-H
>
< HTTP/1.1 201 Created
< Content-Length: 18
< Content-Type: text/html; charset=UTF-8
< X-Trans-Id: tx128ef4eb253b4535b23c46a68603b173
< Date: Thu, 26 Jul 2012 08:12:16 GMT
<
201 Created
```

This end-to-end test verifies several things:

- That the OpenStack authentication service is functional
- That the project and user account settings are valid
- That the CDMI Interface for OpenStack is visible via HTTP

- That the CDMI Interface for OpenStack is correctly configured to point to the underlying OpenStack Swift installation
- That the underlying OpenStack Swift installation is functional

3.4.2 List of Running Processes

In case OpenStack has been installed in the directory '/opt/stack/' the following command will allow the admin to see all processes running out of that directory:

```
ps -auxw | grep /opt/stack
```

The output should include the following two services:

```
/opt/stack/keystone/bin/keystone-all --config-file
/etc/keystone/keystone.conf --log-config
/etc/keystone/logging.conf -d --debug

/opt/stack/swift/bin/swift-proxy-server /etc/swift/proxy-
server.conf -v
```

Many other services based on your installation might also be available. Also noted that the 'keystone' is optional here as on some deployments keystone may be deployed on a separate node.

3.4.3 Network interfaces Up & Open

By default the CDMI interface runs on port 8080 and so TCP port 8080 should be accessible to the client.

By default the Keystone authentication service runs on port 5000 and so TCP port 5000 should be accessible to the client.

As per OpenStack Swift installation requirements, the proxy server and the storage nodes in the Swift Ring should all be in the same network and have access to each other.

3.4.4 Databases

Swift does not use traditional databases.

3.5 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

3.5.1 Resource availability

In the event of an issue it is recommended that the integrity and functionality of the OpenStack Swift deployment first be examined using the OpenStack Horizon web interface. Using the relevant OpenStack account details, it should be possible to use the OpenStack Horizon web interface to browse any containers and objects the user already has stored. It should also be possible to upload and store new objects. Any

issues that exist can then be debugged using the wealth of OpenSource Swift resources and documentation that already exists.

If OpenStack Swift appears to be working fine via the OpenStack Horizon interface, then the installation of the CDMI Interface for OpenStack needs to be examined. As there are no ongoing storage needs for this component, initial investigation should focus on possible network issues.

During debugging of resources,

- Verify that sufficient diskspace remains on a node using the UNIX command 'df'
- Check to see if unexpected error messages exist in /var/log

3.5.2 Remote Service Access

The CDMI Interface fo OpenStack Swift is a RESTful service, and so TCP port 8080 on the server must be open and accessible to the client.

Assuming OpenStack Keystone is used for authentication purposes, by default TCP port 5000 on the Keystone node also needs to be open.

3.5.3 Resource consumption

Swift and the CDMI interface have minimal resource constraints in and of themselves (e.g., 100MB memory, 10MB disk space). The main concern is that sufficient storage is available to accommodate all objects to be stored, as well as multiple copies of these objects for redundancy purposes (likely to be TBs of data in a typical environment).

Note that a Swift installation is recommended to have at least 5 storage nodes for redundancy purposes.

3.5.4 I/O flows

Clients access the Object Storage through the CDMI interface (port 8080). This is simple HTTP traffic. Depending on the request, calls are made to the OpenStack Swift service head node - again HTTP traffic. Based on the request, the Swift head node may call one or more Swift storage nodes to retrieve or store the required object. Storage requests will result in multiple copies of the data being stored on multiple storage nodes for redundancy purposes. Relevant responses and stored objects are returned back to the Client through standard HTTP responses.

3.6 References

1. ↑ http://www.openstack.org/
2. ↑ http://www.snia.org/cdmi
3. ↑ http://docs.openstack.org/developer/swift/howto_installmultinode.html
4. ↑ http://github.com/tmetsch/cdmi

4 PaaS Management - Installation and Administration Guide

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

4.1 PaaS Manager Installation

This guide tries to define the procedure to install the PaaS Manager in a machine, including its requirements and possible troubleshooting that we could find during the installation.

4.1.1 Requirements

In order to execute the PaaS Manager, it is needed to have previously installed the following software:

- Tomcat 7.X.X [1]
- PostgreSQL [2]

4.1.2 Database configuration

The PaaS Manager needs to have PostgreSQL installed in service mode and a database created. For CentOS, these are the instructions: First, it is required to install the PostgreSQL [3].

```
# yum install postgresql postgresql-server postgresql-contrib
```

4.1.2.1 *Start PostgreSQL*

Type the following commands to install the postgresql as service and start it

```
# chkconfig --add postgresql
# chkconfig postgresql on
# service postgresql initdb
# service postgresql start
```

4.1.2.2 *Create the DB*

Connect as postgres user to the PostgreSQL server and set the password for user postgres using alter user as below:

```
# su - postgres
postgres$ psql postgres postgres;
psql (8.4.13)
Type "help" for help.
```

```
postgres=# alter user postgres with password 'postgres';
```

Create the database

```
postgres=# create database paasmanager;
postgres=# grant all privileges on database paasmanager to
postgres;
```

exit quit "\q" and then "exit"

Edit file /var/lib/pgsql/data/pg_hba.conf and set authentication method to md5:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
# "local" is for Unix domain socket connections only
local   all         all                               md5
local   all         postgres                          md5
# IPv4 local connections:
host    all         all         0.0.0.0/0          md5
```

Edit file /var/lib/pgsql/data/postgresql.conf and set listen addresses to 0.0.0.0:

```
listen_addresses = '0.0.0.0'
```

Reload configuration

```
# service postgresql reload
```

4.1.3 Apache Tomcat configuration

4.1.3.1 *Install Tomcat 7*

Install Tomcat 7 together with standard Tomcat samples, documentation, and management web apps:

```
yum install tomcat7-webapps tomcat7-docs-webapp tomcat7-admin-
webapps
```

Start/Stop/Restart Tomcat 7 as a service. startp:

```
sudo service tomcat7 start
```

stop:

```
sudo service tomcat7 stop
```

restart:

```
sudo service tomcat7 restart
```

Add Tomcat 7 service to the autostart

```
sudo chkconfig tomcat7 on
```

4.1.3.2 *Install PaaS Manager application*

Once the prerequisites are satisfied, you shall create the context file as $CATALINA_HOME/conf/Catalina/localhost/paasmanager.xml (substituting PATH_TO_WEBAPP to the corresponding directory):

```
<Context path="/paasmanager" docBase="PATH_TO_WEBAPP"
reloadable="true" debug="5">
  <Resource name="jdbc/paasmanager" auth="Container"
type="javax.sql.DataSource"
driverClassName="org.postgresql.Driver"
   url="jdbc:postgresql://localhost:5432/paasmanager"
   username="postgres" password="postgres"
   maxActive="20" maxIdle="10" maxWait="-1"/>
</Context>
```

Include the library postgresql-8.4-702.jdbc4.jar in $CATALINA_HOME/lib

Configure the profile fiware in the catalina.properties. So that, open the file $CATALINA_HOME/conf/catalina.properties and write at the end

```
 spring.profiles.active=fiware
```

Start tomcat

```
$ sudo service tomcat7 start
```

Check that database schema is created: connect to PostgreSQL and describe table configuration_properties:

```
$ psql -U postgres -W paasmanager

paasmanager=# \d configuration_properties

     Table "public.configuration_properties"
  Column   |           Type           | Modifiers
-----------+--------------------------+-----------
```

```
key       | character varying(255)   | not null
namespace | character varying(255)   | not null
value     | character varying(32672) |
```

4.1.3.3 *Configure PaaS Manager application*

Go to the paasmanager database and configure the different properties in the configuration_properties. Mainly, all those related to keystone configuration. Concretely, you need to configure:

- openstack.keystone.url: the url where keystone Openstack service is installed
- openstack-tcloud.keystone.user: admin user
- openstack-tcloud.keystone.adminPass: admin password
- openstack-tcloud.keystone.tenant: a tenant (user)

4.2 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

4.2.1 End to End testing

Although one End to End testing must be associated to the Integration Test, we can show here a quick testing to check that everything is up and running. It involves to obtain the product information storaged in the catalogue. With it, we test that the service is running and the database configure correctly.

```
http://{PaaSManagerIP}:{port}/paasmanager/rest
```

The request to test it in the testbed should be

```
curl -v -H "Access-Control-Request-Method: GET" -H "Content-
Type: application xml" -H "Accept: application/xml"
-H "X-Auth-Token: 5d035c3a29be41e0b7007383bdbbec57" -H
"Tenant-Id: 60b4125450fc4a109f50357894ba2e28" -X GET "
http://{PaaSManagerIP}:{port}/paasmanager/rest/catalog/org/FIW
ARE/environment"
```

Whose result is the PaaS Manager API documentation.

4.2.2 List of Running Processes

Due to the PaaS Manager basically is running over the Tomcat, the list of processes must be only the Tomcat and PostgreSQL. If we execute the following command:

```
```

```
ps -ewF | grep 'postgres\|tomcat' | grep -v grep
```

It should show something similar to the following:

```
postgres  2057     1  0 30179   884   0 Nov05 ?
00:00:00 /usr/bin/postmaster -p 5432 -D /var/lib/pgsql/data

postgres  2062  2057  0 27473   248   0 Nov05 ?
00:00:00 postgres: logger process

postgres  2064  2057  0 30207   636   0 Nov05 ?
00:00:00 postgres: writer process

postgres  2065  2057  0 27724   160   0 Nov05 ?
00:00:00 postgres: stats buffer process

postgres  2066  2065  0 27521   204   0 Nov05 ?
00:00:00 postgres: stats collector process

root      2481     1  0 228407 96324  0 Nov05 ?
00:03:34 /usr/bin/java -
Djava.util.logging.config.file=/opt/apache-tomcat-
7.0.16/conf/...

postgres  2501  2057  0 31629   560   0 Nov05 ?
00:00:01 postgres: postgres paasmanager 127.0.0.1(49303) idle

postgres  7208  2057  0 30588  3064   0 Nov05 ?
00:00:00 postgres: postgres paasmanager 127.0.0.1(49360) idle
```

4.2.3 Network interfaces Up & Open

Taking into account the results of the ps commands in the previous section, we take the PID in order to know the information about the network interfaces up & open. To check the ports in use and listening, execute the command:

```
netstat –p –a | grep $PID/java
```

Where $PID is the PID of Java process obtained at the ps command described before, in the previous case 18641 tomcat and 23546 (postgresql). The expected results must be something similar to the following:

```
tcp        0      0 localhost.localdomain:8005  *:*
LISTEN      2481/java

tcp        0      0 *:8009                      *:*
LISTEN      2481/java

tcp        0      0 *:webcache                  *:*
LISTEN      2481/java

tcp        0      0 localhost.localdomain:49360
localhost.localdom:postgres ESTABLISHED 2481/java

tcp        0      0 localhost.localdomain:49303
localhost.localdom:postgres ESTABLISHED 2481/java
```

```
tcp        0      0 *:postgres                  *:*
LISTEN      2057/postmaster

tcp        0      0 *:postgres                  *:*
LISTEN      2057/postmaster

udp        0      0 localhost.localdomain:33556
localhost.localdomain:33556 ESTABLISHED 2057/postmaster


unix       2      [ ACC ]     STREAM     LISTENING     8921
2057/postmaster     /tmp/.s.PGSQL.5432
```

4.2.4 Databases

The last step in the sanity check, once that we have identified the processes and ports is to check the different databases that have to be up and accept queries. Fort he first one, if we execute the following commands:

```
psql -U postgres -d paasmanager
```

For obtaining the tables in the database, just use

```
paasmanager=# \dt


 Schema|                 Name                        | Type  |
Owner


---------+-------------------------------------------+-------+----
------
 public  | applicationinstance                       | tabla |
postgres
 public  | applicationrelease                        | tabla |
postgres
 public  | applicationrelease_applicationrelease     | tabla |
postgres
 public  | applicationrelease_artifact               | tabla |
postgres
 public  | applicationrelease_attribute              | tabla |
postgres
 public  | applicationtype                           | tabla |
postgres
 public  | applicationtype_environmenttype           | tabla |
postgres
 public  | artifact                                  | tabla |
postgres
 public  | artifact_artifact                         | tabla |
postgres
```

```
 public  | artifacttype                              | tabla |
postgres
 public  | attribute                                 | tabla |
postgres
 ...
```

4.3 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

4.3.1 Resource availability

The resource availability should be at least 1Gb of RAM and 6GB of Hard disk in order to prevent enabler's bad performance. This means that bellow these thresholds the enabler is likely to experience problems or bad performance.

4.3.2 Remote Service Access

The PaaS Manager should connect with the Service Manager, Service Deployment and Configuration (SDC) and IdM GE (aka Keystone in this first release). An administrator to verify that such links are available will use this information.

The first step is to check the IdM GE connectivity (without security the rest of components do not work). Thus, in order to check the connectivity between the PaaS Manager and the IdM GE, due to it must obtain a valid token and tenant for a user and organization with the following curl commands:

```
root@fiware:~# curl -d '{"auth": {"tenantName":
"<MY_ORG_NAME>", "passwordCredentials":{"username":
"<MY_USERNAME>", "password": "<MY_PASS>"}}}'-H "Content-type:
application/json" -H "Accept: application/xml"
http://<KEYSTONE_HOST>:<KEYSTONE_PORT>/v2.0/tokens
```

The <MY_ORG_NAME> will be the name of my Organization/Tennat/Project predefined in the IdM GE (aka Keystone). The <MY_USERNAME> and <MY_PASS>variables will be the user name and password predefined in the IdM GE and finally the <KEYSTONE_HOST> and <KEYSTONE_PORT> variables will be the IP direction and port in which we can find the IdM GE (aka Keystone). This request should return one valid token for the user credentials together with more information in a xml format:

```
<?xml version="1.0" encoding="UTF-8"?>
<access xmlns="http://docs.openstack.org/identity/api/v2.0">
  <token expires="2012-06-30T15:12:16Z"
id="9624f3e042a64b4f980a83afbbb95cd2">
```

```
    <tenant enabled="true"
id="30c60771b6d144d2861b21e442f0bef9" name="FIWARE">
      <description>FIWARE Cloud Chapter demo
project</description>
    </tenant>
  </token>
  <serviceCatalog>
  …
  </serviceCatalog>
  <user username="org" id="b988ec50efec4aa4a8ac5089adddbaf9"
name="org">
    <role id="32b6e1e715f14f1dafde24b26cfca310"
name="Member"/>
  </user>
</access>
```

After that, we can check that the Service Manager (Claudia) is up and running, for this purpose we can execute the following curl command, which is a simple GET operation to obtain the different existing flavours:

```
curl -v -H "Access-Control-Request-Method: GET" -H "Access-
Control-Request-Headers: Content-Type, X-Auth-Token" -H
"Origin: http://<IP LOCAL>" -H 'X-Auth-Token:
a9a861db6276414094bc1567f664084d' -H "Content-Type:
application/xml" -H "Accept: application/xml" -X GET
"http://<OPENSTACK API
HOST>:8774/v2.0/c8da25c7a373473f8e8945f5b0da8217/flavors"
```

With a result that the following one:

```
* About to connect() to <SERVICEMANAGER API HOST> port 8774
(#0)
*   Trying <SERVICEMANAGER  API HOST>... connected
* Connected to <SERVICEMANAGER API HOST> (<SERVICEMANAGER API
HOST>) 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: <SERVICEMANAGER API HOST>:8774
> Access-Control-Request-Method: GET
> Access-Control-Request-Headers: Content-Type, X-Auth-Token
> Origin: http://<IP LOCAL>
> X-Auth-Token: a9a861db6276414094bc1567f664084d
> Content-Type: application/xml
```

```
> Accept: application/xml
>
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Access-Control-Allow-Origin: http://<IP LOCAL>
< Access-Control-Allow-Credentials: true
< Www-Authenticate: Keystone
uri='http://<KEYSTONE_HOST>:<KEYSTONE_PORT>/v2.0/'
< Content-Type: application/xml
< Content-Length: 1128
< Date: Wed, 24 Oct 2012 12:46:49 GMT
<
* Connection #0 to host <SERVICEMANAGER API HOST> left intact
* Closing connection #0
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<flavors xmlns:ns2="http://www.w3.org/2005/Atom"
xmlns:ns3="http://docs.openstack.org/compute/api/v1.1">
  <ns3:flavor id="001" name="M1_TINY 0Gb">
    <ns2:link rel="self"
href="http://130.206.80.91:8774/v2.0/c8da25c7a373473f8e8945f5b
0da8217/flavors/001"/>
  </ns3:flavor>
  <ns3:flavor id="002" name="M1_SMALL 20Gb">
    <ns2:link rel="self"
href="http://130.206.80.91:8774/v2.0/c8da25c7a373473f8e8945f5b
0da8217/flavors/002"/>
  </ns3:flavor>
  <ns3:flavor id="003" name="M1_MEDIUM 40Gb">
    <ns2:link rel="self"
href="http://130.206.80.91:8774/v2.0/c8da25c7a373473f8e8945f5b
0da8217/flavors/003"/>
  </ns3:flavor>
  <ns3:flavor id="004" name="M1_LARGE 80Gb">
    <ns2:link rel="self"
href="http://130.206.80.91:8774/v2.0/c8da25c7a373473f8e8945f5b
0da8217/flavors/004"/>
  </ns3:flavor>
  <ns3:flavor id="005" name="M1_XLARGE 160Gb">
    <ns2:link rel="self"
href="http://130.206.80.91:8774/v2.0/c8da25c7a373473f8e8945f5b
0da8217/flavors/005"/>
```

```
  </ns3:flavor>
  <ns3:flavor id="006" name="M2_TINY 0Gb">
    <ns2:link rel="self"
href="http://130.206.80.91:8774/v2.0/c8da25c7a373473f8e8945f5b
0da8217/flavors/006"/>
  </ns3:flavor>
</flavors>
```

Finally, with the token id, we can perform a GET operation to the Service Deployment and Configuration (SDC) in order to get information about the software catalogue. For this purpose we can execute the following curl command:

```
curl -v -H "Access-Control-Request-Method: GET" -H "Access-
Control-Request-Headers: Content-Type, X-Auth-Token" -H
"Origin: http://<IP LOCAL>" -H 'X-Auth-Token:
a9a861db6276414094bc1567f664084d' -H "Content-Type:
application/xml" -H "Accept: application/xml" -X GET
"http://<SDC API HOST>:8080/sdc/rest/catalog/product"
```

It provides an XML with the product information:

```
<products>
   <product>
      <name>tomcat</name>
      <description>tomcat J2EE container</description>
      <attributes>
         <key>port</key>
         <value>8080</value>
      </attributes>
   </product>
   <product>
      <name>postgresql</name>
        ...
   </product>
<products>
```

4.3.3 Resource consumption

State the amount of resources that are abnormally high or low. This applies to RAM, CPU and I/O. For this purpose we have differentiated between:

- Low usage, in which we check the resources that the Tomcat requires in order to load the PaaS Manager.
- High usage, in which we send 100 concurrent accesses to the PaaS Manager.

The results were obtained with a top command execution over the following machine configuration:

Machine Info	
	Tomcat Node
Type Machine	Virtual Machine
CPU	1 core @ 2,4Ghz
RAM	1,4GB
HDD	9,25GB
Operating System	CentOS 6.3

The results of requirements both RAM, CPU and I/O to HDD is shown in the following table:

Resource Consumption (in Tomcat node)		
	Low Usage	**High Usage**
RAM	512MB ~ 63%	3GB ~ 78%
CPU	0,8% of a 2400MHz	90% of a 2400MHZ
I/O HDD	6GB	6GB

4.3.4 I/O flows

The application WAR is hearing from port 8080. Please refer to the installation process in order to know exactly which was the port selected.

5 Software Deployment And Configuration - Installation and Administration Guide

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

5.1 SDC Installation

This guide tries to define the procedure to install the SDC in a machine, including its requirements and possible troubleshooting that we could find during the installation. We have to talk about two nodes, one including the core functionality of the enabler itself and the other one which allocated the chef server.

5.1.1 Requirements

In order to execute the SDC, it is needed to have previously installed the following software:

- Chef node
  - Chef server [1]. For CentOS it is possible to follow the following instructions[2]
- SDC node
  - Tomcat 7.X.X [3]
  - PostgreSQL [4]
  - Webdav

5.1.2 Chef server

5.1.2.1 *Chef server Installation*

The installation of the chef-server involves to install the chef-server package, which can be obtained in [5]. We can just execute

```
rpm -Uvh chef-server-package.rpm
```

Verify the the hostname for the Chef server by running the hostname command. The hostname for the Chef server must be a FQDN. This means hostaname.domainame. In case it is not configure, you can do it

```
hostname chef-server.localdomain
```

and include it in the /etc/hosts After that, it is required to configure the certificates and other staff in the chef-server, with chef-server-ctl. This command will set up all of the required components, including Erchef, RabbitMQ, and PostgreSQL.

```
sudo chef-server-ctl reconfigure
```

In order to test verify the installation of Chef Server 11.x by running the following command:

```
sudo chef-server-ctl test
```

After that, you can obtain the different certificates for the different clients in /etc/chef-server. There you can find a chef-validator.pem (needed for all the nodes), the chef-

server-gui for the GUI.. You can copy them in order to use them later. The next step is to configure a client in the chef-server so that you can execute the chef-server CLI. To do that, you need to install the chef-client

```
curl -L https://www.opscode.com/chef/install.sh | sudo bash
```

and configure it with the following command. You can accept all the default

```
knife configure --initial
```

The validation_key attribute in the knife.rb file must specify the path to the validation key. The validation_client_name attribute defaults to chef-validator (which is the chef-validator.pem private key created by the open source Chef server on startup). When prompted for the URL for the Chef server, use the FQDN for the Chef server Once you have a client configured, you can run the CLI. Just one example:

```
knife client list
```

5.1.2.2 *Chef server cookbook repository*

The FI-WARE cookbook repository is in FI-WARE SVN repository. To upload the recipes into the chef server you need:

- To dowload the svn repository:

```
svn checkout https://forge.fi-ware.org/scmrepos/svn/testbed/trunk/cookbooks
```

- Inside the cookbooks folder, create a file update with the following content. It will update the repository and upload into the chef-server

```
svn update
knife cookbook upload --all -o BaseRecipes/
knife cookbook upload --all -o BaseSoftware/
knife cookbook upload --all -o GESoftware/
```

5.1.3 Database configuration

The SDC node needs to have PostgreSQL installed in service mode and a database created called SDC. For CentOS, these are the instructions: Firsly, it is required to install the PostgreSQL [6].

```
yum install postgreql postgresql-server posgresql-cotrib
```

5.1.3.1 *Start Postgresql*

Type the following commands to install the postgresql as service and restarted

```
# chkconfig --add postgresql
# chkconfig postgresql on
# service postgresql start
```

Then, you need to configure postgresql to allow for accessing. In /var/lib/pgsql/data/postgresql.conf

```
listen_addresses = '0.0.0.0'
```

in /var/lib/pgsql/data/pg_hba.conf, you need to add

```
host    all         all           0.0.0.0/0             md5
```

Restart the postgres

```
service postgresql restart
```

5.1.3.2 *Create the DB*

Connect to Postgresql Server using

```
# su - postgres
```

Connect as postgres to the postgres database and set the password for user postgres using alter user as below:

```
$ psql postgres postgres;
$ alter user postgres with password 'postgres';
```

Create the SDC DB

```
createdb sdc
```

Check that the database has been created correctly:

```
su - postgres
psql -U postgres sdc -h localhost
```

Then we need to create the database tables for the sdc. To do that obtain the files from [7] and execute

```
psql -d sdc -a -f db-initial.sql
psql -d sdc -a -f db-changelog.sql
```

5.1.4 Apache Tomcat configuration

5.1.4.1 *Install Tomcat 7*

Install Tomcat 7 together with standard Tomcat samples, documentation, and management web apps:

```
yum install tomcat7-webapps tomcat7-docs-webapp tomcat7-admin-
webapps
```

Start/Stop/Restart Tomcat 7 as a service. startp:

```
sudo service tomcat7 start
```

stop:

```
sudo service tomcat7 stop
```

restart:

```
sudo service tomcat7 restart
```

Add Tomcat 7 service to the autostart

```
sudo chkconfig tomcat7 on
```

5.1.4.2 *Install SDC applications*

Once the prerequisites are satisfied, you shall create the context file. To do that, change sdc.xml found in distribution file and store it in folder $CATALINA_HOME/conf/Catalina/localhost.

See the snipet bellow to know how it works:

```
<Context path="/sdc" docBase="war path" reloadable="true"
debug="5">
  <Resource name="jdbc/sdc" auth="Container"
type="javax.sql.DataSource"
driverClassName="org.postgresql.Driver"  <!-- select the
driver-->
   url="jdbc:postgresql://localhost:5432/sdc" <!-- select the
connection url-->
   username="postgres" password="postgres" <!-- select the
user/password-->
   maxActive="20" maxIdle="10" maxWait="-1"/>
</Context>
```

You also have to add the provided scripts found in the dist file (in folder /opt/sdc/scripts/) in the same folder (or everywhere you want if you prefer to change the default configuration).

Include the library postgresql-8.4-702.jdbc4.jar in $CATALINA_HOME

5.1.4.3 *Configure SDC application*

The configuration of SDC is in configuration_properties table. There, it is required to configure:

- openstack-tcloud.keystone.url: This is the url where the keystone-proxy is deployed

- openstack-tcloud.keystone.user: the admmin user
- openstack-tcloud.keystone.password: the admin password
- openstack-tcloud.keystone.tenant: the admin tenant
- sdc_manager_url: the final url, mainly http://sdc-ip:8080/sdc

The last step is to create a sdc client in the chef-server, so that, the SDC can communicate with the chef-server. To do that, we can use the chef-server-web-ui, which is usually deployed on https://chef-server-ip, go to https://chef-server-ip/clients and create a sdc client as administrator. Then, it is required to copy the private key.

In the sdc machine, it is required to copy this private key in /etc/chef/sdc.pem (you can configure the path also in the properties)

5.1.4.4 *Register SDC application into keystone*

The last step involves to regiter the SDC and chef-server endpoints into the keystone endpoint catalogue. To do that, you should write into the config.js in the keystone-proxy the following lines: [12:14:57] jesus.m.movilla: {"endpoints": [

```
        {"adminURL": "http://sdc-ip:8080/sdc/rest",
        "region": "myregion",
        "internalURL": "http://sdc-ip:8080/sdc/rest",
        "publicURL": "http://sdc-ip:8080/sdc/rest"
        }
        ],
        "endpoints_links": [],
        "type": "sdc",
        "name": "sdc"
    },
    {"endpoints": [
        {"adminURL": "https://chef-server-ip",
        "region": "myregion",
        "internalURL": "https://chef-server-ip",
        "publicURL": "https://chef-server-ip"
        }
        ],
        "endpoints_links": [],
        "type": "chef-server",
        "name": "chef-server"
    },
```

where myregion should be the name of the openstack region defined.

5.1.4.5 *Creating images sdc-aware*

The images to be deployed by the SDC, should have some features, like to have the chef-client installed and configured correctly with the chef-server. In the roadmap, it is considered to avoid all this process and to make possible any image to be SDC-aware, installing and configuring everything in booting status.

```
mkdir /etc/chef
mkdir /var/log/chef
curl -L https://www.opscode.com/chef/install.sh | bash
```

You should copy the chef-validator.pem from the chef-server into /etc/chef

Then, it is required to create a file called client.rb in /etc/chef. The validation.pem should be obtained from the chef-server in the folder /etc/chef-server and its called chef-validator.pem and rename to validation.pem in the /etc/chef folder of the image

```
log_location            "/var/log/chef/client.log"
ssl_verify_mode         :verify_none
validation_client_name "chef-validator"
validation_key          "/etc/chef/validation.pem"
client_key              "/etc/chef/client.pem"
chef_server_url         "https://cher-server-ip"
```

Finally, to start chef-client in boot time

```
chef-client -i 60 -s 6
```

5.2 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

5.2.1 End to End testing

Although one End to End testing must be associated to the Integration Test, we can show here a quick testing to check that everything is up and running. It involves to obtain the product information storaged in the catalogue. With it, we test that the service is running and the database configured correctly.

```
http://{IP VM SDC}:8080/sdc/rest/catalog/product
```

In addition, for checking that the SDC server is working properly, it is needed to check;

```
http://{IP VM Chef server}:4040
```

5.2.2 List of Running Processes

Due to the SDC basically is running over the Tomcat, the list of processes must be only the Tomcat and PostgreSQL. If we execute the following command:

```
ps -ewF | grep 'postgres\|tomcat' | grep -v grep
```

It should show something similar to the following:

```
root     18641     1  0 380364 287052   0 Nov08 ?
00:02:45 /usr/bin/java -
Djava.util.logging.config.file=/opt/apache-tomcat-7.0....

postgres 23546     1  8  53511   6284   0 09:22 ?
00:00:01 /usr/bin/postmaster -p 5432 -D /var/lib/pgsql/data

postgres 23550 23546  0  44264   1392   0 09:22 ?
00:00:00 postgres: logger process

postgres 23553 23546  0  53511   1616   0 09:22 ?
00:00:00 postgres: writer process

postgres 23554 23546  0  53511   1556   0 09:22 ?
00:00:00 postgres: wal writer process

postgres 23555 23546  0  53579   1968   0 09:22 ?
00:00:00 postgres: autovacuum launcher process

postgres 23556 23546  0  44331   1676   0 09:22 ?
00:00:00 postgres: stats collector process
```

For the Chef server node:

```
ps -ewF | grep 'chef' | grep -v grep
```

It should show something similar to the following:

```
rabbitmq  1757     1  2 153867 64876  0 Nov06 ?
01:50:45 /usr/lib64/erlang/erts-5.6.5/bin/beam -W ...

root      2075     1  0 295700 202080 0 Nov06 ?
00:05:37 java -Xmx256M -Xms256M -
Dsolr.data.dir=/var/chef/solr/data -...

root      2106     1  0 27534 23792   0 Nov06 ?
00:00:32 /usr/bin/ruby /usr/bin/chef-expander -d -c
/etc/chef/expander.rb ...

root      2107  2106  2 31599 39016   0 Nov06 ?
01:58:06 chef-expander worker #1 (vnodes 0-1023)

chef      3001     1  0 46487 60616   0 Nov07 ?
00:09:50 merb : chef-server (api) : worker (port 4000)


chef      3036     1  0 40514 59496   0 Nov07 ?
00:09:31 merb : chef-server-webui : worker (port 4040)
```

5.2.3 Network interfaces Up & Open

Taking into account the results of the ps commands in the previous section, we take the PID in order to know the information about the network interfaces up & open. To check the ports in use and listening, execute the command:

```
netstat –p –a | grep $PID/java
```

Where $PID is the PID of Java process obtained at the ps command described before, in the previous case 18641 tomcat and 23546 (postgresql). The expected results must be something similar to the following:

```
tcp        0      0 *:irdmi                     *:*
LISTEN      18641/java
tcp        0      0 *:8009                      *:*
LISTEN      18641/java
tcp        0      0 *:webcache                  *:*
LISTEN      18641/java
tcp        0      0 localhost:mxi               *:*
LISTEN      18641/java
tcp      120      0 tid_centos_sdc:33760
tid_centos_sdc:postgres     CLOSE_WAIT  18641/java
tcp      120      0 tid_centos_sdc:33752
tid_centos_sdc:postgres     CLOSE_WAIT  18641/java
tcp      120      0 tid_centos_sdc:33729
tid_centos_sdc:postgres     CLOSE_WAIT  18641/java
tcp      120      0 tid_centos_sdc:33732
tid_centos_sdc:postgres     CLOSE_WAIT  18641/java
tcp      120      0 tid_centos_sdc:33759
tid_centos_sdc:postgres     CLOSE_WAIT  18641/java
tcp      120      0 tid_centos_sdc:33731
tid_centos_sdc:postgres     CLOSE_WAIT  18641/java
tcp      120      0 tid_centos_sdc:33748
tid_centos_sdc:postgres     CLOSE_WAIT  18641/java
tcp        0      0 *:postgres                  *:*
LISTEN      23546/postmaster
tcp        0      0 *:postgres                  *:*
LISTEN      23546/postmaster
udp        0      0 localhost:45194
localhost:45194             ESTABLISHED 23546/postmaster


unix  2      [ ACC ]     STREAM     LISTENING     930790
23546/postmaster    /tmp/.s.PGSQL.5432
```

5.2.4 Databases

The last step in the sanity check, once that we have identified the processes and ports is to check the different databases that have to be up and accept queries. For he first one, if we execute the following commands:

```
psql -U postgres -d sdc
```

For obtaining the tables in the database, just use

```
\dt
```

5.3 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

5.3.1 Resource availability

The resource availability should be at least 1Gb of RAM and 6GB of Hard disk in order to prevent enabler's bad performance. This means that bellow these thresholds the enabler is likely to experience problems or bad performance.

5.3.2 Remote Service Access

The SDC should connect with the IdM GE (aka Keystone in this first release) and the Chef server to perform its functionality. An administrator to verify that such links are available will use this information.

The first step is to check the IdM GE connectivity (without security the rest of components do not work). Thus, in order to check the connectivity between the SDC and the IdM GE, due to it must obtain a valid token and tenant for a user and organization with the following curl commands:

```
root@fiware:~# curl -d '{"auth": {"tenantName":
"<MY_ORG_NAME>", "passwordCredentials":{"username":
"<MY_USERNAME>", "password": "<MY_PASS>"}}}'-H "Content-type:
application/json" -H "Accept: application/xml"
http://<KEYSTONE_HOST>:<KEYSTONE_PORT>/v2.0/tokens
```

The <MY_ORG_NAME> will be the name of my Organization/Tennat/Project predefined in the IdM GE (aka Keystone). The <MY_USERNAME> and<MY_PASS>variables will be the user name and password predefined in the IdM GE and finally the <KEYSTONE_HOST> and <KEYSTONE_PORT> variables will be the IP direction and port in which we can find the IdM GE (aka Keystone). This request should return one valid token for the user credentials together with more information in a xml format:

```
<?xml version="1.0" encoding="UTF-8"?>
<access xmlns="http://docs.openstack.org/identity/api/v2.0">
  <token expires="2012-06-30T15:12:16Z"
id="9624f3e042a64b4f980a83afbbb95cd2">
    <tenant enabled="true"
id="30c60771b6d144d2861b21e442f0bef9" name="FIWARE">
      <description>FIWARE Cloud Chapter demo
project</description>
    </tenant>
  </token>
  <serviceCatalog>
  …
  </serviceCatalog>
  <user username="org" id="b988ec50efec4aa4a8ac5089adddbaf9"
name="org">
    <role id="32b6e1e715f14f1dafde24b26cfca310"
name="Member"/>
  </user>
</access>
```

After that, we can check that the Chef server is running:

```
GET http://{CHEF SERVER IP}:4040/
```

It will return the Chef server main webpage.

5.3.3 Resource consumption

State the amount of resources that are abnormally high or low. This applies to RAM, CPU and I/O. For this purpose we have differentiated between:

- Low usage, in which we check the resources that the Tomcat requires in order to load the PaaS Manager.
- High usage, in which we send 100 concurrent accesses to the PaaS Manager.

The results were obtained with a top command execution over the following machine configuration:

Machine Info	
	Tomcat Node
Type Machine	Virtual Machine

CPU	1 core @ 2,4Ghz
RAM	1,4GB
HDD	9,25GB
Operating System	CentOS 6.3

The results of requirements both RAM, CPU and I/O to HDD is shown in the following table:

Resource Consumption (in Tomcat node)		
	Low Usage	**High Usage**
RAM	1GB ~ 63%	3GB ~ 78%
CPU	0,8% of a 2400MHz	90% of a 2400MHZ
I/O HDD	6GB	6GB

5.3.4 I/O flows

The application WAR is hearing from port 8080. Please refer to the installation process in order to know exactly which was the port selected.

6 Policy Manager - Installation and Administration Guide

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

6.1 Policy Manager Installation

This guide tries to define the procedure to install the Policy Manager in a machine, including its requirements and possible troubleshooting that we could find during the installation. We have to talk about two applications deployed in a Django server.

Final deployment into a production system should be performed in apache server with mod wsgi.

You could find instructions for deploy policy Manager in apache + mod_wgsi at the end of this document.

6.1.1 Requirements

In order to execute the Policy Manager, it is needed to have previously installed the following software of framework in the machine:

- Rule engine dependencies:
  - Python 2.7.6 [1].
  - Sqlite3 [2].
  - PyClips 1.0 [3]
  - RabbitMQ 3.3.0 [4]
- Facts engine dependencies:
  - Python 2.7.6 [5].
  - Redis 2.8.8 [6]

6.1.2 Rule engine installation

There is no need to configure any special options in django server. Run as default mode.

6.1.2.1 *Step 1: Install python*

If you do not have python installed by default, please, follow instructions for your Operating System in the official page: https://www.python.org/download/releases/2.7.6/

6.1.2.2 *Step 2: Install pyclips*

Download pyclips from http://sourceforge.net/projects/pyclips/files/pyclips/pyclips-1.0

To install pyClips execute this following commands:

```
$ tar -xvf pyclips-1.0.X.Y.tar.gz
$ cd pyclips
```

```
$ python setup.py build
$ su -c "python setup.py install"
```

Maybe you need to execute these commands using sudo.

If everything was OK, you should not receive any error.

6.1.2.3 *Step 3: Install RabbitMQ*

To install RabbitMQ Server, it is better to refer official installation page and follow instructions for the Operating System you use: http://www.rabbitmq.com/download.html

After installation, you should start RabbitMQ. Note that you only need one instace of RabbitMQ and It could be installed in a different server than fiware-facts or Rule Engine.

6.1.2.4 *Step 4: Download and execute the Rule Engine server*

Download the component by executing the following instruction:

```
git clone git@github.com:telefonicaid/fiware-cloto.git
```

It should show something like the following:

```
Cloning into 'fiware-cloto'...
remote: Counting objects: 1483, done.
remote: Compressing objects: 100% (692/692), done.
remote: Total 1483 (delta 951), reused 1261 (delta 743)
Receiving objects: 100% (1483/1483), 196.42 KiB | 0 bytes/s,
done.
Resolving deltas: 100% (951/951), done.
Checking connectivity... done.
```

Go to the directory where we download the server and execute the following commands:

1. Installing all dependencies

```
$ sudo pip install -r requirements.txt
```

It should install all dependencies showing at the end a message similar to:

```
Successfully installed nose django-nose coverage
djangorestframework mockito python-keystoneclient circus
netaddr
Cleaning up...
```

2. Creating the structure of Database

```
$ python manage.py syncdb
```

It should shown the following information when it is executed:

```
WARNING:RuleEngine:DataBase already exists: no such table:
cloto_serverinfo
INFO:RuleEngine:SERVER STARTED
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_groups
Creating table auth_user_user_permissions
Creating table auth_user
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table cloto_serverinfo
Creating table cloto_specificrule
Creating table cloto_subscription
Creating table cloto_entity_specificrules
Creating table cloto_entity_subscription
Creating table cloto_entity
Creating table cloto_tenantinfo
Creating table cloto_rule
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

If this command asks you to create a superuser, say no, it's unnecessary.

3. Configuring Rule engine

Before starting the rule engine, you should edit configuration.py located at cloto folder. Constants you need to complete are:

```
- All in # OPENSTACK CONFIGURATION: Openstack information
- RABBITMQ_URL: URL Where RabbitMQ is listening (no port
needed, it uses default port)
- CONTEXT_BROKER_URL: URL where Context Broker is listening
- NOTIFICATION_URL: URL where notification service is
listening (This service must be implemented by the user)
```

in addition you could modify other constants like NOTIFICATION_TIME, or DEFAULT_WINDOW_SIZE.

Finally you should modify ALLOWED_HOSTS parameter in settings.py adding the hosts you want to be accesible from outside, your IP address, the domain name, etc. An example could be like this:

```
ALLOWED_HOSTS = ['policymanager.host.com','80.71.123.2']
```

4. Starting the server

```
$ python manage.py runserver 8000
```

It should shown the following information when it is executed:

```
Validating models...

0 errors found
April 11, 2014 - 14:12:42
Django version 1.5.5, using settings 'cloto.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

If you want to start Rule Engine using other IP address, you should execute:

```
$ python manage.py runserver <IP>:8000
```

Where IP is a valid network intarface assigned. It is recommended if your Rule Engine will be called from different networks.

6.1.3 Facts installation

6.1.3.1 ***Step 1: Install python***

The process will be the same that be see in the previous section.

6.1.3.2 ***Step 2: Install Redis***

Download, extract and compile Redis with:

```
$ wget http://download.redis.io/releases/redis-2.8.8.tar.gz
$ tar xzf redis-2.8.8.tar.gz
$ cd redis-2.8.8
$ make
```

The binaries that are now compiled are available in the src directory. Run Redis with:

```
```

```
$ src/redis-server
```

It execute the redis server on port 6379.

You can interact with Redis using the built-in client:

```
$ src/redis-cli
redis> set foo bar
OK
redis> get foo
"bar"
```

6.1.3.3 *Step 3: Download and execute the facts engine server*

Download the component by executing the following instruction:

```
git clone git@github.com:telefonicaid/fiware-facts.git
```

It should show something like the following:

```
Cloning into 'fiware-facts'...
remote: Counting objects: 211, done.
remote: Compressing objects: 100% (136/136), done.
remote: Total 211 (delta 118), reused 152 (delta 63)
Receiving objects: 100% (211/211), 65.79 KiB | 0 bytes/s,
done.
Resolving deltas: 100% (118/118), done.
Checking connectivity... done.
```

Go to the directory where we download the server and execute the following commands:

Go to the directory where we download the server and execute the following commands:

1. Installing all dependencies

```
$ sudo pip install -r requirements.txt
```

It should install all dependencies showing at the end a message similar to:

```
Successfully installed redis flask gevent pika
Cleaning up...
```

Then, after the installation of the requirements associated to the facts engine, it is hour to execute the server, just run:

```
```

```
$ python facts.py
```

It should shown the following information when it is executed:

```
2014-04-11 10:42:19,344 INFO policymanager.facts
policymanager.facts 1.0.0


2014-04-11 10:42:19,344 INFO policymanager.facts Running in
stand alone mode
2014-04-11 10:42:19,345 INFO policymanager.facts Port: 5000
2014-04-11 10:42:19,345 INFO policymanager.facts PID: 6059


2014-04-11 10:42:19,345 INFO policymanager.facts
https://github.hi.inet/telefonicaid/fiware-facts
```

6.2 Installation into a Production System

If you want to deploy Policy Manager with this propose, you should deploy on Apache Server with mod_wsgi

6.2.1 Rule Engine

6.2.1.1 *Step 1: Install Apache with mod_wsgi*

Apache used to be installed on most of linux systems. If you do not have apache installed, try downloading from your package manager like apt-get or yum Also you can download from the official site http://httpd.apache.org/

After install apache, The official mod_wsgi documentation it's the best guide for all the details about how to use mod_wsgi on your system. https://code.google.com/p/modwsgi/wiki/InstallationInstructions

6.2.1.2 *Step 2: Apache configuration*

Once you've got mod_wsgi installed and activated, edit your httpd.conf file and add:

```
WSGIScriptAlias / PATH_TO_fiware-cloto/cloto/wsgi.py
WSGIPythonPath PATH_TO_fiware-cloto
<Directory PATH_TO_fiware-cloto/cloto>
<Files wsgi.py>
Order deny,allow
Allow from all
</Files>
</Directory>
```

```
<Directory PATH_TO_fiware-cloto>
<Files cloto.db>
Allow from all
</Files>
</Directory>
<Directory /var/log/fiware-cloto>
<Files RuleEngine.log>
Allow from all
</Files>
</Directory>
```

If you have apache above 2.2 version, you have to replace "Allow form all" with "Require all granted"

In addition you must add the port listening 8000 in case of fiware-cloto

```
Listen 8000
```

6.2.1.3 *Step 3: Run apache*

Finally , run apache service to have a fiware-cloto instance running

```
sudo apachectl start
```

6.2.2 Facts

6.2.2.1 *Step 1: Install Apache with mod_wsgi*

This step is the same as described in step 1 of Rule Engine. please follow those instructions.

6.2.2.2 *Step 2: Apache configuration*

Once you've got mod_wsgi installed and activated, edit your httpd.conf file and add:

```
WSGIScriptAlias / PATH_TO_fiware-facts/facts.py
WSGIPythonPath PATH_TO_fiware-facts
<Directory PATH_TO_fiware-facts>
<Files facts.py>
Order deny,allow
Allow from all
</Files>
</Directory>
<Directory /var/log/fiware-facts>
<Files fiware-facts.log>
Allow from all
```

```
</Files>
</Directory>
```

If you have apache above 2.2 version, you have to replace "Allow form all" with "Require all granted"

In addition you must add the port listening 5000 in case of fiware-facts

```
Listen 5000
```

6.2.2.3 ***Step 3: Run apache***

Finally , run apache service to have a fiware-facts instance running

```
sudo apachectl start
```

6.3 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

6.3.1 End to End testing

Although one End to End testing must be associated to the Integration Test, we can show here a quick testing to check that everything is up and running. For this purpose we send a request to our API in order to test the credentials that we have from then and obtain a valid token to work with.

In order to make a probe of the different functionalities related to the Policy Manager, we start with the obtention of a valid token for a registered user. Due to all operations of the Policy Manager are using the security mechanism which is used in the rest of the cloud component, it is needed to provide a security token in order to continue with the rest of operations. For this operation we need to execute the following curl sentence.

```
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 IdM, 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">
```

- Tennant-Id

```
<tenant enabled="true" id="c907498615b7456a9513500fe24101e0"
name=$TENNANT>
```

After it, we can check if the Policy Manager is up and running with a single instruction which is used to return the information of the status of the processes together with the queue size.

```
curl -v -H 'X-Auth-Token: a9a861db6276414094bc1567f664084d' -X
GET
http://130.206.81.71:8000/v1.0/c907498615b7456a9513500fe24101e
0
```

This operation will return the information regarding the tenant details of the execution of the Policy Manager

```
< HTTP/1.0 200 OK
< Date: Wed, 09 Apr 2014 08:25:17 GMT
< Server: WSGIServer/0.1 Python/2.6.6
< Content-Type: text/html; charset=utf-8
{
    "owner": "Telefonica I+D",
    "doc": "https://forge.fi-
ware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSp
ecification.Details.Cloud.PolicyManager",
    "runningfrom": "14/04/09 07:45:22",
    "version": 1.0,
    "windowsize": 5
}
```

For more details to use this GE, please refer to the Policy Manager - User and Programmers Guide.

6.3.2 List of Running Processes

Due to the Policy Manager basically is running over the python process, the list of processes must be only the python and redis in case of the facts engine. If we execute the following command:

```
ps -ewf | grep 'redis\|Python' | grep -v grep
```

It should show something similar to the following:

```
UID   PID  PPID   C   STIME     TTY       TIME   CMD
501  5287   343   0  9:42PM ttys001    0:02.49   ./redis-
server *:6379
501  5604   353   0  9:40AM ttys002    0:00.20
/Library/Frameworks/Python.framework/Versions/2.7/Resources/Py
thon.app/Contents/MacOS/Python facts.py
```

Where you can see the Redis server, and the run process to launch the Python program.

In case of the rule engine node, if we execute the following command:

```
ps -ewf | grep 'rabbitmq-server\|python' | grep -v grep
```

It should show something similar to the following:

```
UID        PID  PPID  C    SZ   RSS PSR STIME TTY
TIME CMD
root      1584     1  0 15:31 ?        00:00:00 /bin/sh
/etc/rc3.d/S80rabbitmq-server start
root      1587  1584  0 15:31 ?        00:00:00 /bin/bash -c
ulimit -S -c 0 >/dev/null 2>&1 ; /usr/sbin/rabbitmq-server
root      1589  1587  0 15:31 ?        00:00:00 /bin/sh
/usr/sbin/rabbitmq-server
root      1603  1589  0 15:31 ?        00:00:00 su rabbitmq -s
/bin/sh -c /usr/lib/rabbitmq/bin/rabbitmq-server
root      2038  2011  0 15:32 ?        00:00:01 python
cloto/environmentManager.py
root      2039  2011  1 15:32 ?        00:00:38
/usr/bin/python manage.py runserver 172.30.1.119:8000
```

where we can see the rabbitmq process, the run process to launch the Python program and the clips program.

6.3.3 Network interfaces Up & Open

Taking into account the results of the ps commands in the previous section, we take the PID in order to know the information about the network interfaces up & open. To check the ports in use and listening, execute the command:

```
lsof -i | grep "$PID1\|$PID2"
```

Where $PID1 and $PID2 are the PIDs of Python and Redis server obtained at the ps command described before, in the previous case 5287 (redis-server) and 5604 (Python). The expected results must be something similar to the following:

```
```

```
COMMAND    PID USER    FD  TYPE             DEVICE SIZE/OFF
NODE NAME
redis-ser 5287  fla    4u  IPv6 0x8a557b63682bb0ef      0t0
TCP *:6379 (LISTEN)
redis-ser 5287  fla    5u  IPv4 0x8a557b636a696637      0t0
TCP *:6379 (LISTEN)
redis-ser 5287  fla    6u  IPv6 0x8a557b63682b9fef      0t0
TCP localhost:6379->localhost:56046 (ESTABLISHED)
Python    5604  fla    7u  IPv6 0x8a557b63682bacaf      0t0
TCP localhost:56046->localhost:6379 (ESTABLISHED)
Python    5604  fla    9u  IPv4 0x8a557b6369c90637      0t0
TCP *:commplex-main (LISTEN)
```

In case of rule engine, the result will we the following:

```
COMMAND    PID USER    FD  TYPE             DEVICE SIZE/OFF
NODE NAME
python    2039       root    3u  IPv4  13290      0t0  UDP
*:12027
python    2039       root    4u  IPv4  13347      0t0  TCP
policymanager.novalocal:irdmi (LISTEN)
python    2044       root    3u  IPv6  13354      0t0  TCP
localhost:38391->localhost:amqp (ESTABLISHED)
```

6.3.4 Databases

The last step in the sanity check, once that we have identified the processes and ports is to check the database that have to be up and accept queries. For the first one, if we execute the following commands inside the code of the rule engine server:

```
$ sqlite3 cloto.db
```

Where cloto.db is the file that contains the information of the SQLite Databases. The previous command should show something like the following:

```
SQLite version 3.6.20
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

In order to show the different tables contained in this database, we should execute the following commands with the result that we show here:

```
sqlite> .tables
auth_group                  cloto_rule
```

```
auth_group_permissions      cloto_serverinfo
auth_permission             cloto_specificrule
auth_user                   cloto_subscription
auth_user_groups            cloto_tenantinfo
auth_user_user_permissions  django_content_type
cloto_entity                django_session
cloto_entity_specificrules  django_site
cloto_entity_subscription
sqlite>
```

Now, we can execute a simple test query in order to check the content of the table:

```
sqlite> .header on
sqlite> .width 2 14 7 26 80
sqlite> .mode column
sqlite> select * from cloto_serverinfo;
```

It is important that you execute the command "*.header on*", which allows you showing the header info of the tables. The other instructions are used in order to show the information in a more friendly way. And it should return with the following information:

```
id  owner           version  runningfrom                 doc
--  --------------  -------  --------------------------  -----
-------------------------------------------------------------
-------------
1   Telefonica I+D  1.0      2014-04-11 12:32:29.604238
https://forge.fi-
ware.org/plugins/mediawiki/wiki/fiware/index.php/FIWAR
```

6.4 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

6.4.1 Resource availability

The resource availability in the node should be at least 2Gb of RAM and 8GB of Hard disk in order to prevent enabler's bad performance in both nodes. This means that bellow these thresholds the enabler is likely to experience problems or bad performance.

6.4.2 Remote Service Access

We have internally two components to connect, the Rule engine component and the facts engine component. After that two internals component, we should connect with the the IdM GE. An administrator to verify that such links are available will use this information.

The first step is to check that the facts engine is up and running, for this purpose we can execute the following curl command, which is a simple GET operation:

```
root@fiware:~# curl http://$IP:$PORT/v1.0
```

The <IP/Host> variable will be the IP direction in which we have installed the facts engine. This request should return the status of the server if it is working properly:

```
{"fiware-facts":"Up and running..."}
```

In order to check the connectivity between the rule engine and the IdM GE, due to it must obtain a valid token and tenant for a user and organization with the following curl commands:

```
root@fiware:~# curl -d '{"auth": {"tenantName":
"<MY_ORG_NAME>", "passwordCredentials":{"username":
"<MY_USERNAME>", "password": "<MY_PASS>"}}}' -H "Content-type:
application/json" -H "Accept: application/xml"
http://<KEYSTONE_HOST>:<KEYSTONE_PORT>/v2.0/tokens
```

The <MY_ORG_NAME> will be the name of my Organization/Tennat/Project predefined in the IdM GE (aka Keystone). The <MY_USERNAME> and <MY_PASS> variables will be the user name and password predefined in the IdM GE and finally the <KEYSTONE_HOST> and <KEYSTONE_PORT> variables will be the IP direction and port in which we can find the IdM GE (aka Keystone). This request should return one valid token for the user credentials together with more information in a xml format:

```
<?xml version="1.0" encoding="UTF-8"?>
<access xmlns="http://docs.openstack.org/identity/api/v2.0">
  <token expires="2012-06-30T15:12:16Z"
id="9624f3e042a64b4f980a83afbbb95cd2">
    <tenant enabled="true"
id="30c60771b6d144d2861b21e442f0bef9" name="FIWARE">
      <description>FIWARE Cloud Chapter demo
project</description>
    </tenant>
  </token>
  <serviceCatalog>
  …
  </serviceCatalog>
```

```
  <user username="fla" id="b988ec50efec4aa4a8ac5089adddbaf9"
name="fla">
    <role id="32b6e1e715f14f1dafde24b26cfca310"
name="Member"/>
  </user>
</access>
```

With this information (extracting the token id), we can perform a GET operation to the rule engine in order to get the information related to the window size associated to a tenant. For this purpose we can execute the following curl commands:

```
curl -v -H 'X-Auth-Token: a9a861db6276414094bc1567f664084d' -X
GET "http://<Rule Engine
HOST>:8000/v1.0/c8da25c7a373473f8e8945f5b0da8217"
```

The <Rule engine HOST> variable will be the IP direction in which we have installed the Rule engine API functionality. This request should return the valid info for this tenant in the following json response structure:

```
{
    "owner": "Telefonica I+D",
    "doc": "https://forge.fi-
ware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSp
ecification.Details.Cloud.PolicyManager",
    "runningfrom": "14/04/11 12:32:29",
    "version": 1.0,
    "windowsize": 5
}
```

6.4.3 Resource consumption

State the amount of resources that are abnormally high or low. This applies to RAM, CPU and I/O. For this purpose we have differentiated between:

- Low usage, in which we check the resources that the JBoss or Tomcat requires in order to load the IaaS SM.
- High usage, in which we send 100 concurrent accesses to the Claudia and OpenStack API.

The results were obtained with a top command execution over the following machine configuration:

Machine Info		
	Rule Engine Node	**Facts Engine Node**

Type Machine	Virtual Machine	Virtual Machine
CPU	1 core @ 2,4Ghz	Intel(R) Xeon(R) CPU X5650 Dual Core @ 2.67GHz
RAM	2GB	2GB
HDD	20GB	20GB
Operating System	CentOS 6.3	CentOS 6.3

The results of requirements both RAM, CPU and I/O to HDD in case of Rule engine node is shown in the following table:

Resource Consumption (in JBoss node)		
	Low Usage	**High Usage**
RAM	1,2GB ~ 70%	1,4GB ~ 83,5%
CPU	1,3% of a 2400MHz	95% of a 2400MHZ
I/O HDD	6GB	6GB

And the results of requirements both RAM, CPU and I/O to HDD in case of Tomcat node is shown in the following table:

Resource Consumption (in Tomcat node)		
	Low Usage	**High Usage**
RAM	1,2GB ~ 63%	1,5GB ~ 78%
CPU	0,8% of a 2400MHz	90% of a 2400MHZ
I/O HDD	6GB	6GB

6.4.4 I/O flows

The rule engine application is hearing from port 8000 and the Fact-Gen application (by default) is hearing in the port 5000. Please refer to the installation process in order to know exactly which was the port selected.

7 Monitoring - Installation and Administration Guide

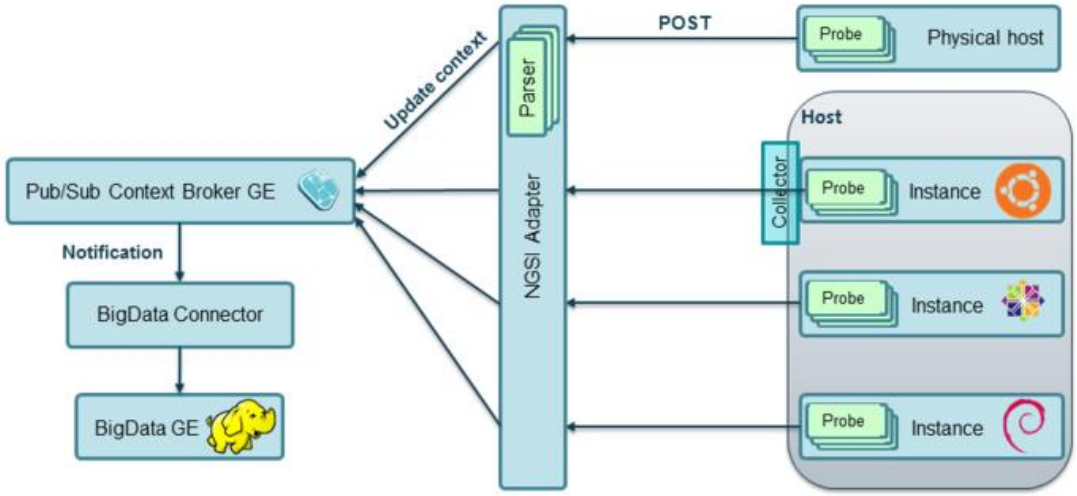
You can find the content of this chapter as well in the wiki of fi-ware.

7.1 Introduction

This guide defines the procedure to install the different components that build up the Monitoring GE, including its requirements and possible troubleshooting.

7.2 Installation

Monitoring infrastructure comprises several elements distributed across different hosts, as depicted in the following figure:



1. **Probes**, installed at every host being monitored.
2. **NGSI Adapter**, responsible for translating probe raw data into a common format (NGSI).
3. **Parsers** at NGSI Adapter, specific for the different probes that generate monitoring data.
4. **Context Broker GE**, where monitoring data (expressed as "NGSI context updates") will be published.
5. **BigData GE**, for analysis of historical context data.
6. **BigData Connector** between Context Broker and BigData GEs.

7.2.1 Installation of probes

Monitoring GE is agnostic to the framework used to gather monitoring data. It just assumes there are several probes collecting such data, which somehow will be forwarded to the adaptation layer (NGSI Adapter).

It is up to the infrastructure owner which tool (like Nagios, Zabbix, openNMS, perfSONAR, etc.) is installed for this purpose.

Probes must "publish" their data to NGSI Adapter. Depending on the exact monitoring tool installed, a kind of **collector** has to be deployed in order to send data to the adapter. An example specific for Nagios, implemented as an Event Broker Module, can be found here including installation details.

7.2.2 Installation of NGSI Adapter

7.2.2.1 *Requirements*

NGSI Adapter should work on a variety of operating systems, particularly on the majority of GNU/Linux distributions (e.g. Debian, Ubuntu, CentOS), as it only requires a V8 JavaScript Engine to run a Node.js server.

Hardware Requirements

The minimal requirements are:

- RAM: 2 GB

Software Requirements

NGSI Adapter is a standalone Node.js process, so `node` and its package manager `npm` should be installed previously. Please check https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager for detailed instructions:

- Installation in Ubuntu

```
$ sudo apt-get update
$ sudo apt-get install -y python-software-properties python
g++ make
$ sudo add-apt-repository -y ppa:chris-lea/node.js
$ sudo apt-get update
$ sudo apt-get install nodejs
```

- Installation in Debian

```
$ sudo apt-get install python g++ make checkinstall
$ mkdir ~/src && cd $_
$ wget -N http://nodejs.org/dist/node-latest.tar.gz
$ tar xzvf node-latest.tar.gz && cd node-v*
$ ./configure
$ sudo checkinstall -y --install=no --pkgversion 0.10.24  #
Replace with current version number.
$ sudo dpkg -i node_*
```

7.2.2.2 *Downloads*

Please download zip from https://github.com/Fiware/fiware-monitoring/archive/master.zip. This includes the ngsi_adapter directory corresponding to the NGSI Adapter component.

```
$ unzip master.zip
```

7.2.2.3 *Dependencies*

NGSI Adapter requires some packages, which can be installed using the `npm` package manager:

```
$ cd ngsi_adapter/src
$ npm install
```

7.2.3 Installation of parsers

NGSI Adapter currently includes a predefined set of parsers for Nagios probes at `src/lib/parsers/` directory, each parser named after its corresponding probe. New custom parsers should be placed here.

7.2.4 Installation of Context Broker GE

Please refer to Publish/Subscribe Broker - Orion Context Broker - Installation and Administration Guide

7.2.5 Installation of BigData GE

Please refer to BigData Analysis - Installation and Administration Guide

7.2.6 Installation of the connector

This component subscribes to changes at Context Broker and writes them into BigData GE storage. Historically the **ngsi2cosmos** connector implementation has been used (installation details here), although from March 2014 this component is deprecated and a brand new **Cygnus** implementation (installation details here) is available.

7.3 Running the monitoring components

As stated before, there are a number of distributed components involved in the monitoring. Please refer to their respective installation manuals for execution details (this applies to probes & monitoring software, Context Broker, BigData, etc.). This section focuses on NGSI Adapter specific instructions.

7.3.1 Running NGSI Adapter

Once installed, there are two ways of running NGSI Adapter: manually from the command line or as a system service. It is not recommended to mix both ways (e.g. start it manually but using the service scripts to stop it).

7.3.1.1 *From the command line*

You can run the adapter just typing the following command from `ngsi_adapter/src/` directory:

```
$ adapter
```

You can use command line arguments, e.g. to specify the port in adapter listens:

```
$ adapter --listenPort 5000
```

Help for command line options:

```
$ adapter --help
```

7.3.1.2 *As system service*

Software distribution includes *init.d* scripts to configure NGSI Adapter as a Linux service:

- `ngsi_adapter/scripts/init.d/redhat/ngsi_adapter` for RedHat and CentOS
- `ngsi_adapter/scripts/init.d/ubuntu/ngsi_adapter` for Ubuntu and Debian

Scripts have to be customized according to our actual installation. The following variables may be changed:

DAEMON

Full path of `adapter` script

DAEMON_ARGS

Command line arguments

DAEMON_USER

Linux user to run service

Once `ngsi_adapter` service has been configured, the following commands are available to control its execution:

```
$ sudo service ngsi_adapter start
$ sudo service ngsi_adapter stop
$ service ngsi_adapter status
```

7.3.1.3 *Configuration options*

These options can be used directly (in the case of running from the command line, but prepending `--` prefix) or as part of the default configuration (see `defaults` at the configuration file `ngsi_adapter/src/config/options.js`):

listenHost

The hostname or address at which NGSI Adapter listens

listenPort

The port number at which NGSI Adapter listens

brokerUrl

The URL of the Context Broker instance to publish data to

retries

Number of times a request to Context Broker is retried, in case of error

Besides, logging options are controlled by `opts` at the configuration file `ngsi_adapter/src/config/logger.js`:

logLevel

Verbosity of log messages

logFile

Full path of log file

logMaxSize

Maximum size (in bytes) of log file

logMaxFiles

Maximum number of rotating log files

7.4 Sanity check procedures

These are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

7.4.1 End to End testing

- At the monitored host, reschedule some probe execution to force the generation of new monitoring data.
- Check NGSI Adapter logs for incoming requests with raw data and outgoing Context Broker requests as NGSI updateContext() operations:

```
$ cat ngsi_adapter.log
... << HTTP POST
... >> 200 OK
... << Body ...raw monitoring data..
... POST http://contextbroker:1026/
```

- Finally, query Context Broker for new data (see details here)

7.4.2 List of Running Processes

A `node` process running the "adapter" server should be up and running, e.g.:

```
$ ps -C node -f | grep adapter
fiware   21930     1  0 Mar28 ?        00:06:06 node
/usr/local/monitoring/ngsi_adapter/src/adapter
```

Alternatively, we can check if service is running, e.g.:

```
$ service ngsi_adapter status
 * ngsi_adapter is running
```

7.4.3 Network interfaces Up & Open

NGSI Adapter uses TCP 1337 as default port, although it can be changed using the `--listenPort` command line option.

7.4.4 Databases

This component does not persist any data, and no database engine is needed.

7.5 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

7.5.1 Resource availability

Although we haven't done yet a precise profiling on NGSI Adapter, tests done in our development and testing environment show that a host with 2 CPU cores and 4 GB RAM is fine to run server.

7.5.2 Remote Service Access

- Probes at monitored hosts should have access to NGSI Adapter listen port (TCP 1337, by default)
- NGSI Adapter should have access to Context Broker listen port (TCP 1026, by default)
- Connector should have access to Context Broker listen port in order to subscribe to context changes
- Context Broker should have access to Connector callback port to notify changes

7.5.3 Resource consumption

Please refer to Context Broker and BigData resource consumption sections.

7.5.4 I/O flows

Figure at installation section shows the I/O flows among the different monitoring components:

- Probes send requests to NGSI Adapter with raw monitoring data
- NGSI Adapter sends request to Context Broker in terms of context updates of the monitored resources
- Context Broker notifies Connector with every context change
- Connector writes changes to BigData storage

8 Self-Service Interfaces - Installation and Administration Guide

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

8.1 Introduction

Welcome to the Installation and Administration Guide for the Self Service Interfaces Generic Enabler. This generic enabler is built on an Open Source project, 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.

The required parts of this generic enabler are Object Storage GE, Service Manager GE, DCRM GE, OpenStack's Keystone, or Identity Manager GE. And it can also work with PaaS Manager GE.

8.1.1 Requirements

In order to execute the Self Service Interfaces GE, it is needed to have previously installed the following software of framework:

- Administrative Scripting Toolkit
  - Node.js Server v0.8.17 or greater. [1]
  - Node Packaged Modules. It is usually included within Node.js [2]
- User Cloud Portal
  - Node.js Server v0.8.17 or greater. [3]
  - Node Packaged Modules. It is usually included within Node.js [4]
  - Ruby is also used for generating CSS files during installation.

8.2 System Installation

- **Prerequisites**

You should install Ruby, Ruby gem, Node and npm prior to run the Cloud portal.

This installation is divided into two parts:

- **Administrative Scripting Toolkit**

It provides advanced cloud management, such as reboot, resize and change password of servers.

To use the scripting toolkit you first need to install the jstack component:

```
npm install jstack-client -g
```

Once installed you can use the script, for instance, by running the next commands:

```
jstack-client -u username -p password -l
http://130.206.80.100:5000/v2.0/ -t tenant_id server-list
```

- **User Cloud Portal**

It supports daily user operations, such as start and stop instances, list images, create key-pairs, etc.

The following steps need to be performed to get the Cloud Portal up and running:

1. Download the Cloud Portal, using [GitHub].

```
git clone https://github.com/ging/fi-ware-cloud-portal.git
portal
```

2. Install all required libraries using NPM.

```
cd portal
sudo gem install sass
npm install
```

3. Configure installation

To configure Cloud portal you can copy the file named config.js.template to config.js and edit it with the corresponding info. Below you can see an example:

```
var config = {};

config.useIDM = false;

config.oauth = {
    account_server: ,
    client_id: ,
    client_secret: ,
    callbackURL:
};

config.keystone = {
    host: '130.206.80.123',
    port: 5000,
    admin_host: '130.206.80.123',
    admin_port: 35357,
    username: 'administrator',
    password: 'password',
    tenantId: '213127893128937128 9euwe'
};

module.exports = config;
```

4. Compile the code by running:

```
npm install
```

```
```

5. Launch the executable by running the next command with administrative permissions as it is going to be run on TCP Port 80:

```
node server.js
```

6. You can also install forever.js to run it in a production environment:

```
sudo npm install forever -g
```

7. And then run the server using forever:

```
forever start server.js
```

8. To know the status of the process you can run the next command:

```
forever status
```

8.3 System Administration

Self-Service Interfaces GE do not need specific system administration since it is based on Web interfaces.

8.4 Sanity Check Procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

8.4.1 End to End testing

Please note that the following information is required before carrying out this procedure:

- the IP address of the Cloud Portal node (e.g. on the FI-WARE testbed this is portal.lab.fi-ware.eu)
- the IP address of the Openstack Keystone node managing security for the Self-Service Interfaces GE Deployment (e.g. on the FI-WARE testbed this is 130.206.80.100)
- a valid OpenStack (keystone) username and password

1. Verify that **http://cloud.lab.fi-ware.eu/** can be reached. By default, web access will show a Login Page.

2. Acquire a valid username and password from the testbed, and access with those credentials.

The resulting web page is the landing page of the Cloud Portal.

3. Verify that you can list instances and images of your project.

8.4.2 List of Running Processes

In case you are using forever to run the Self-Service Interfaces the following command will allow the admin to see the process:

```
forever list
```

8.4.3 Network interfaces Up & Open

- TCP port 80 should be accessible to the web browsers in order to load the Portal.
- Cloud GEs should be accessible from the Cloud portal because it makes requests to them.

8.4.4 Databases

Cloud Portal does not use traditional databases. It makes requests directly to other Generic Enablers.

8.5 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GE. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

8.5.1 Resource availability

- Verify that 2.5MB of disk space is left using the UNIX command 'df'

8.5.2 Remote Service Access

Please make sure port 80 is accessible. All other GE's ports need to be accessible too.

8.5.3 Resource consumption

Self-Service Interfaces GE has very minimal resource constraints on the server since it does not have any database or complex application logic.

Typical memory consumption is 100MB and it consumes almost the 1% of a CPU core of 2GHz, but it depends on user demand. It also consumes a great number of TCP sockets and it increases depending again on the demand.

8.5.4 I/O flows

Clients access the Self Service Interface through the client's Web Browser. This is simple HTTP traffic. It makes requests periodically to the different Cloud GEs (SM GE, DCRM GE, Object Storage GE, Keystone, etc.) throught the Cloud portal.

9 Job Scheduler - Installation and Administration Guide

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

9.1 Introduction

Welcome to the *Installation and Administration Guide* for our **ProActive Cloud Job Scheduler**!

ProActive Cloud Job Scheduler is an open source implementation of the Job Scheduler Generic Enabler, derived from ProActive Parallel Suite product (see our Baseline Assets for more details). So, in order to provide a comprehensive documentation, -whenever possible- this guide will point to the appropriate on-line contents . Those are being continuously updated and improved, and so will represent the right place to get the most up-to-date information.

9.1.1 Minimal Systems Requirements

- Oracle JDK 1.6+
- 3GB RAM for Test Environment - 5+GB RAM for Production Environment
- 5GB free on the HDD for Test Environment - 30+GB free on the HDD for Production Environment

9.1.1.1 *Hardware configuration employed for testing*

Info	Value
Machine Type	Workstation
CPU	8 x core @ 2,27Ghz
RAM	23,5GB
HDD	1.5 TB
Operating System	Fedora Release 16 (Linux 3.6.11-4.fc16.x86_64)

9.2 System Installation

Our *ProActive Cloud Job Scheduler* is ready to use and does not require any complicated installation, but just few common operations here follow.

9.2.1 Download & Unzip

- Download at this link all the content of the release of your interest.

- As you may notice, the original *zip* archive has been split into several parts, so it is needed to recompose it. At this end, go to your download folder and, if yours is a Unix system, proceed as follows:

```
bash-4.2$ cat JobScheduler_3.2_part* > JobScheduler_3.2.zip
```

where we assumed your choice was about the release 3.2. Instead, for Windows systems, you have the possibility to run the *copy* command, in order to get the same archive as result, such as:

```
> copy /b JobScheduler_3.2_part* JobScheduler_3.2.zip
JobScheduler_3.2_part00
JobScheduler_3.2_part01
JobScheduler_3.2_part02
JobScheduler_3.2_part03
JobScheduler_3.2_part04
JobScheduler_3.2_part05
JobScheduler_3.2_part06
JobScheduler_3.2_part07
JobScheduler_3.2_part08
         1 file copied.
```

where the related output was added to be thorough.

- Once done, let us check the integrity of the just built archive, named *JobScheduler_3.2.zip*, leveraging some md5sum tools (available for both Unix and Windows systems at External Links section) and the related *md5-checksum.txt* file. Therefore, go to your download folder and type the following:

```
bash-4.2$ md5sum -c md5-checksum.txt
JobScheduler_3.2.zip: OK
```

- If *OK* is returned, then the archive is not corrupted and it can be decompressed in */opt* (Unix) or *C:\opt* (Windows) folder. That step might require that you change the current permissions of the *opt* folder, in order to succeeding in decompressing the content.

So, finally, we will have our main *ProActive Cloud Job Scheduler* folder (that will be our home folder) at */opt/ProActiveCloudJobScheduler_$RELEASE* or *C:\opt\ProActiveCloudJobScheduler_%RELEASE%*, according to your operating system, with the following sub-folders:

```
bash-4.2$ cd /opt
bash-4.2$ ls ProActiveCloudJobScheduler_$RELEASE
cloud_service_provider_conectors  programming  scheduling
scheduling_portal  scheduling_rest
```

Free your disk of your archives downloaded, which are not required in the next steps.

- Otherwise, if the checksum process goes wrong, restart from the beginning.

9.2.2 Environment setup

As stated, we assume that JDK 1.6+ is installed on your system and the related *JAVA_HOME* environment variable is defined. Then, the following actions are required:

- Additional environment variable definition:

```
JOB_SCHEDULER_GE_HOME=[/opt/ProActiveCloudJobScheduler_$RELEAS
E|C:\opt\ProActiveCloudJobScheduler_%RELEASE%] #respectively,
for [unix|windows] case
```

- finally, reboot the system.

9.3 Administration Guide

9.3.1 Configuration

Since any *ProActive Cloud Job Scheduler* component is built leveraging ProActive JVM, customizing all services for more challenging contexts might be your case. So, it might be wise to start getting familiar with **Appendix A - ProActive JVM Properties & Configuration**.

Afterwards, we let you focus on the existence of the *$JOB_SCHEDULER_GE_HOME/scheduling/config* file, where all static configuration files are gathered in sub-folders according to their scope.

In particular, *proactive* sub-folder contains the main *ProActiveConfiguration.xml*, shared by all services representing our GE implementation. So, if some globally visible JVM properties should be modified, you can discover how to inject them thanks to the help menu available for any script used. That might be the case when common computing nodes need to share a specific configuration. Instead, for the the Scheduler and the RM Service, we recommend to have an ad-hoc customization, by editing the associated configuration files *$JOB_SCHEDULER_GE_HOME/scheduling/config/[rm/scheduler]/settings.ini*.

There, you might notice that, by default the login methods activated are *[RM | Scheduler]FiwareIdentityManagementGELoginMethod*s, available to authenticate users against the *[RM | Scheduler]* Service leveraging an authentication login modules chain. That consists of the local *[RM | Scheduler]FileLoginModule*, followed by the *[RM | Scheduler]OpenStackKeystoneLoginModule*, which is needed to cope against any *Identity Management GEi* and requires the right settings of the related *OpenStack Keystone* properties at *$JOB_SCHEDULER_GE_HOME/scheudling/config/openstack/openstack.ini*, as stated in *$JOB_SCHEDULER_GE_HOME/scheduling/config/jaas.config*.

In order to configure the *ProActive Cloud Job Scheduler* we leverage the *$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js* script, whose help usage is accessible as follows:

```
bash-4.2$ $JAVA_HOME/bin/jrunscript -
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin -
Dpa.startserver.help=true
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js


---------------------------------

    Initialization process

---------------------------------

ProActive configuration built and available at
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
config/proactive/ProActiveConfiguration.xml

Deployment information file is built and available at
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
config/deployment.info


------- Printing JVM System variables having pattern
"^pa\.startserver.*"--------

pa.startserver.help=true

pa.startserver.user.properties.regex.pattern=^proactive.*


------- Printing JVM System variables having pattern
"^proactive.*"--------


---------------------------------

     HELP: USAGE EXAMPLES

---------------------------------


To run, please use jrunscript (jdk tool) as follows:
$JAVA_HOME/bin/jrunscript -
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin start-
server.js

1. For RMI deployment example, type: $JAVA_HOME/bin/jrunscript
-Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js


2. For RMI deployment with all the services listening to the
same IP address bound to the network interface eth0, type:
$JAVA_HOME/bin/jrunscript -
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin -
Dproactive.net.interface=eth0
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js


3. For PAMR deployment example, type:
$JAVA_HOME/bin/jrunscript -
```

```
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin -
Dproactive.communication.protocol=pamr
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js


4. For RMI deployment with jetty server listening to the port
8000 for http requests example, type:
$JAVA_HOME/bin/jrunscript -
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin -
Dpa.startserver.frontend.jetty.endpoint.port=8000
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js


5. For specifying the Java input/output temporary directory,
type: $JAVA_HOME/bin/jrunscript -
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin -
Djava.io.tmpdir=PathToYourTempDirectory
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js


6. For launching the services in background (for
administrators only), type: $JAVA_HOME/bin/jrunscript -
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin -
Dpa.startserver.daemon.mode=true
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js
```

If you prefer the usage of the typical shell/batch scripts, we have moved the relate documentation in the **Appendix B - Configuration and Launching via shell/batch scripts**.

By the way, in order to be aware of the whole configuration picture and do not miss important details about, we recommend you to refer to

- ProActive Resource Manager Administration Guide
- ProActive Scheduler Administration Guide

Over there, you can find information about sub-components configuration, such as Authentication, Authorization & Accounting system, internal databases and Dataspaces, which are basically the place where users can push/pull/delete data to be processed.

Going a bit further, there are several type of Dataspaces: each one with its own purpose, but all of them placed by default in your local filesystem (see here):

- the *GLOBALSPACE* is a virtual place, typically under the Scheduler service control domain, shared among all the users, where anyone has the read and write permissions;
- the *USERSPACE* is a virtual place, typically under the Scheduler service control domain, whose access and manipulation is limited to the user in question only;
- the *INPUTSPACE* and *OUTPUTSPACE* are virtual, additional places where users can put/pull/delete data keeping them under their own control domain, which might be typically remote with respect to the Scheduler host location. Those add flexibility to fit the needs of the most exigent users, who cannot/do not want move their data from their premises.

In order to customize their actual location, their configuration guidelines are embedded in the *$JOB_SCHEDULER_GE_HOME/scheduling/config/scheduler/settings.ini* file or here. By the way, in order to quickly overwrite the default location of them in the local file system, you can just specifying the desired path where to store them inside the JVM option *tmp.io.tmpdir*, as expressed in the help usage example. To give you a better idea of what we are speaking about, here follows an example of output logged at *$JOB_SCHEDULER_GE_HOME/scheduling/config/scheduler/.logs/Scheduler.log*, while the Scheduler service is booting up:

```
...
[2013-12-19 13:23:56,836 INFO ] Creating scheduler
authentication interface...
[2013-12-19 13:23:56,838 DEBUG] Looking up authentication
interface 'rmi://172.16.0.1:1100/SCHEDULER'
[2013-12-19 13:23:56,901 DEBUG] Looking up authentication
interface 'rmi://172.16.0.1:1099/RMAUTHENTICATION'
[2013-12-19 13:23:58,348 INFO ] Starting default INPUT space
server...
[2013-12-19 13:23:58,352 INFO ] Started default INPUT space
server at
paprmi://172.16.0.1:1100/DefaultInputSpace?proactive_vfs_provi
der_path=/
[2013-12-19 13:23:58,352 INFO ] default INPUT space server
local path is /tmp/scheduling/defaultinput
[2013-12-19 13:23:58,352 INFO ] Starting default OUTPUT space
server...
[2013-12-19 13:23:58,356 INFO ] Started default OUTPUT space
server at
paprmi://172.16.0.1:1100/DefaultOutputSpace?proactive_vfs_prov
ider_path=/
[2013-12-19 13:23:58,356 INFO ] default OUTPUT space server
local path is /tmp/scheduling/defaultoutput
[2013-12-19 13:23:58,356 INFO ] Starting shared GLOBAL space
server...
[2013-12-19 13:23:58,359 INFO ] Started shared GLOBAL space
server at
paprmi://172.16.0.1:1100/GlobalSpace?proactive_vfs_provider_pa
th=/
[2013-12-19 13:23:58,359 INFO ] shared GLOBAL space server
local path is /tmp/scheduling/defaultglobal
[2013-12-19 13:23:58,359 INFO ] Starting USER spaces server...
[2013-12-19 13:23:58,363 INFO ] Started USER spaces server at
paprmi://172.16.0.1:1100/UserSpaces?proactive_vfs_provider_pat
h=/
[2013-12-19 13:23:58,363 INFO ] USER spaces server local path
is /tmp/scheduling/defaultuser
```

```
...
```

where, you might notice, each dataspace is exposed also through its own *proactive protocol* (*pap*), which depends on the current *proactive.communication.protocol* option (*rmi* by default).

Finally, depending on your network topology, firewall configuration and your needs, it may be required to open some ports on the machine will host your *ProActive Cloud Job Scheduler*. For instance, when you are deploying nodes into the cloud, that machine should have a public IP address, in order to be visible from outside. Moreover, a truly understanding of each protocol behaviour is *a must*, since each one may be suitable for a given network environment. By the way, ProActive Network Configuration will try to help you in such a direction.

Be aware that dealing with networking configuration is never obvious. So, as initial recommendation while configuring new protocols usage, we advise you to involve ports that are not yet busy and offer you a quick way to check if a specific port is free, thanks to netstat command-line tool:

```
natstat -p -a | grep <port number> # for Unix systems
natstat -p -a | findstr <port number> # for Windows systems
```

9.3.2 Launching

The default configuration of *ProActive Cloud Job Scheduler* relies on *rmi* architecture and on its *rmi* protocol for backend communication: each backend component (Resource Manager Service, Scheduler Service, Rest Server) has its own dedicated registry, listening to a given port (respectively 1099, 1100, 1101).

Assuming that we are interested to make the *ProActive Cloud Job Scheduler* be available at our *virbr4* network interface (which has a private IP address), the deployment is easily achievable as follows, by leveraging the *jrunscript* executable shipped by default within the JDK 6+:

```
bash-4.2$ $JAVA_HOME/bin/jrunscript -
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin -
Dproactive.net.interface=virbr4
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js


---------------------------------

    Initialization process

---------------------------------

ProActive configuration built and available at
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
config/proactive/ProActiveConfiguration.xml

Deployment information file is built and available at
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
config/deployment.info


------- Printing JVM System variables having pattern
"^pa\.startserver.*"--------
```

```
pa.startserver.user.properties.regex.pattern=^proactive.*


------- Printing JVM System variables having pattern
"^proactive.*"--------

proactive.net.interface=virbr4


---------------------------------

    Starting server processes

---------------------------------


Running Resource Manager process ...

> Starting the resource manager...

> The resource manager with 4 local nodes created on
rmi://172.16.0.1:1099/

Resource Manager stdout/stderr redirected into
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
.logs/RM-stdout.log


Running Scheduler process ...

> RM URL : rmi://172.16.0.1:1099

> Starting the scheduler...

> Connecting to the resource manager on rmi://172.16.0.1:1099

> The scheduler created on rmi://172.16.0.1:1100/

Scheduler stdout/stderr redirected into
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
.logs/Scheduler-stdout.log


Running Jetty process ...

Checking for
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/rest

Checking for
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/rm

Checking for
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/scheduler

Injecting the Resource Manager and Scheduler backend urls into
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/rest/WEB-INF/portal.properties

Injecting the REST server root url into
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/scheduler/scheduler.conf
```

```
Injecting the REST server root url into
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/rm/rm.conf

Jetty stdout/stderr redirected into
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
.logs/Jetty-stdout.log

Waiting for jetty to start ...

Rest Server webapp deployed at
http://172.16.0.1:8080/rest

Resource Manager webapp deployed at http://172.16.0.1:8080/rm

Scheduler webapp deployed at
http://172.16.0.1:8080/scheduler


Opening browser ...

Please use demo/demo as login/password to connect

Preparing to wait for processes to exit ...

Hit CTRL+C to terminate all server processes and exit
```

Briefly, let us describe what happened behind the scene:

- the script is built to offer a self-configuring deployment process.
- the script is splitted into two parts: the *Initialization process* and *Services Booting process*, whose logs help you to get the main actions executed.
- the default ProActive JVM properties can be overwritten by passing *frontend* and *backend* JVM properties, whose scope is just the script, to reflect better the software architecture.
- During the *Initialization process*
  - an IP address is elected and consistently shared among all the services as *proactive.hostname* to avoid unexpected behaviour.
  - the IP address election algorithm is similar to that one available in the ProActive manual.
  - all the JVM user properties having a given regex pattern (*proactive.** as default) will be injected in the *$JOB_SCHEDULER_GE_HOME/scheduling/config/proactive/ProActiveConfiguration.xml*, now generated at runtime
  - also the *$JOB_SCHEDULER_GE_HOME/scheduling/config/deployment.info* file will be generated at runtime. There you can find the main variables, such as frontend/backend enpoints URLs, Java home, classpath, *ProActive Cloud Job Scheduler* home, protocol and so on, all related to the current deployment, which you would like to export at your needs.
- While, the *Services Booting process* implies
  - checking the ports availability for each service
  - generating booting logs for each service

- creating the web applications (REST server, RM and Scheduler portals) configurations at runtime.
- finally, starting the web browser with three tabs, each one opened at the right endpoint for each web application.

Finally, as administrator, if you would like to run our *ProActive Cloud Job Scheduler* implementation as a unique daemon, we provide also this feature. In particular, what we offer consists of

- leveraging the *pa.startserver.daemon.mode=true* property setting;
- setting up the server as daemon, running at boot time for Linux Standard Base-compliant distros in the default rc levels, under a system user;
- managing its own life-cycle (start|stop|uninstall|clean-wars and many other targets);
- hardening the system where the daemon is running.

In order to achieve that, please follow the steps provided at *$JOB_SCHEDULER_GE_HOME/scheduling/bin/unix/daemon/LinuxStandardBase/readme.txt*.

9.4 Sanity Check Procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

9.4.1 End to End testing

By assuming the default configuration, the REST Server should be reachable at **http://172.16.0.1:8080/rest** endpoint (as coming out when launching the *jrunscript* script). So, for testing it, you might proceed as follows:
1. Trying to access the REST Server endpoint via browser. You should get the welcome message from our *ProActive Cloud Job Scheduler* and some info about the exact root URL for accessing REST resources, together with the related link to the local documentation:

```
You have reached the root location of the rest API of the
ProActive's Scheduler and Resource Manager.


* If you look for the exact URL to use within your client,
please use the following URL:
''http://172.16.0.1:8080/rest/rest ''

* If you look for documentation of the rest api, you can
consult the documentation of the rest api
```

2. Try to access as a not allowed user, like *tizio* as username with *caio* as password.

```
bash-4.2$ curl -v -X POST -d "username=tizio&password=caio"
http://localhost:8080/rest/rest/scheduler/login
```

You should get an output similar to:

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1...   % Total    % Received % Xferd
Average Speed   Time    Time     Time  Current
                                Dload  Upload   Total   Spent
Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-
- --:--:--     0connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /rest/rest/scheduler/login HTTP/1.1
> User-Agent: curl/7.21.7 (x86_64-redhat-linux-gnu)
libcurl/7.21.7 NSS/3.13.5.0 zlib/1.2.5 libidn/1.22
libssh2/1.2.7
> Host: localhost:8080
> Accept: */*
> Content-Length: 28
> Content-Type: application/x-www-form-urlencoded
>
} [data not shown]
100    28    0     0  100    28      0     27  0:00:01
0:00:01 --:--:--    27< HTTP/1.1 404 Not Found
< Content-Type: application/json
< Transfer-Encoding: chunked
< Server: Jetty(6.1.18)
<
{ [data not shown]
100  1509    0  1481  100    28   1106     20  0:00:01
0:00:01 --:--:--  1107
* Connection #0 to host localhost left intact
* Closing connection #0
{
...
    "errorMessage": "Authentication failed",
    "httpErrorCode": 404,
    "stackTrace": "javax.security.auth.login.LoginException:
Authentication failed\n\tat
org.ow2.proactive.authentication.AuthenticationImpl.authentica
te(AuthenticationImpl.java:177)\n\tat
org.ow2.proactive.scheduler.authentication.SchedulerAuthentica
tion.login(SchedulerAuthentication.java:104)\n\tat
sun.reflect.NativeMethodAccessorImpl.invoke0(Native
```

```
Method)\n\tat
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccess
orImpl.java:39)\n\tat
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMeth
odAccessorImpl.java:25)\n\tat
java.lang.reflect.Method.invoke(Method.java:597)\n\tat
org.objectweb.proactive.core.mop.MethodCall.execute(MethodCall
.java:395)\n\tat
org.objectweb.proactive.core.body.request.RequestImpl.serveInt
ernal(RequestImpl.java:253)\n\tat
org.objectweb.proactive.core.body.request.RequestImpl.serve(Re
questImpl.java:197)\n\tat
org.objectweb.proactive.core.body.BodyImpl$ActiveLocalBodyStra
tegy.serveInternal(BodyImpl.java:661)\n\tat
org.objectweb.proactive.core.body.BodyImpl$ActiveLocalBodyStra
tegy.serve(BodyImpl.java:575)\n\tat
org.objectweb.proactive.core.body.AbstractBody.serve(AbstractB
ody.java:944)\n\tat
org.objectweb.proactive.Service.serve(Service.java:121)\n\tat
org.ow2.proactive.authentication.AuthenticationImpl.runActivit
y(AuthenticationImpl.java:241)\n\tat
org.objectweb.proactive.core.body.ActiveBody.run(ActiveBody.ja
va:196)\n\tat java.lang.Thread.run(Thread.java:662)\n"

...

}
```

as expected and consistent with REST API Faults, at Exception to Mapped Error Table.

3. Try to access as the default administrator, that is by logging in as *admin* user with *admin* password:

```
bash-4.2$ curl -v -X POST -d "username=admin&password=admin"
http://localhost:8080/rest/rest/scheduler/login
```

You should get back a random session id, required to perform any operation against REST Server:

```
* About to connect() to localhost port 8080 (#0)

*   Trying 127.0.0.1... connected

* Connected to localhost (127.0.0.1) port 8080 (#0)

> POST /rest/rest/scheduler/login HTTP/1.1

> User-Agent: curl/7.21.7 (x86_64-redhat-linux-gnu)
libcurl/7.21.7 NSS/3.13.5.0 zlib/1.2.5 libidn/1.22
libssh2/1.2.7

> Host: localhost:8080

> Accept: */*

> Content-Length: 29

> Content-Type: application/x-www-form-urlencoded
```

```
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Transfer-Encoding: chunked
< Server: Jetty(6.1.18)
<
* Connection #0 to host localhost left intact
* Closing connection #0
2112db2b13e0cfa4c207d832a2034ac7a124c322112db2b13e0cfa4c208000
```

Store it in a variable, such as SESSION_ID:

```
SESSION_ID=2112db2b13e0cfa4c207d832a2034ac7a124c322112db2b13e0
cfa4c208000
```

4. Check if the Resource Manager Service is working, by typing the following tricky command:

```
bash-4.2$ curl -v -X GET  -H sessionid:`curl -v -X POST -d
"username=admin&password=admin"
http://localhost:8080/rest/rest/rm/login`
http://localhost:8080/rest/rest/rm/isactive
```

Expected output:

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /rest/rest/rm/isactive HTTP/1.1
> User-Agent: curl/7.21.7 (x86_64-redhat-linux-gnu)
libcurl/7.21.7 NSS/3.13.5.0 zlib/1.2.5 libidn/1.22
libssh2/1.2.7
> Host: localhost:8080
> Accept: */*
>
sessionid:4192db2b13e0cfa4c207d2c2a2034ac7a124c322112db2b13e0c
fa4c208000
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Transfer-Encoding: chunked
< Server: Jetty(6.1.18)
<
* Connection #0 to host localhost left intact
```

```
* Closing connection #0
true
```

4. Finally, check if the Scheduler Service is working, by typing the following tricky command:

```
 bash-4.2$ curl -v -X GET  -H sessionid:`curl -v -X POST -d
"username=admin&password=admin"
http://localhost:8080/rest/rest/scheduler/login`
http://localhost:8080/rest/rest/scheduler/isconnected
```

Expected output:

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1...   % Total    % Received % Xferd
Average Speed   Time    Time     Time  Current
                                Dload  Upload   Total   Spent
Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-
- --:--:--     0connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /rest/rest/scheduler/login HTTP/1.1
> User-Agent: curl/7.21.7 (x86_64-redhat-linux-gnu)
libcurl/7.21.7 NSS/3.13.5.0 zlib/1.2.5 libidn/1.22
libssh2/1.2.7
> Host: localhost:8080
> Accept: */*
> Content-Length: 29
> Content-Type: application/x-www-form-urlencoded
>
} [data not shown]
100    29    0     0  100    29      0     28  0:00:01
0:00:01 --:--:--    28< HTTP/1.1 200 OK
< Content-Type: application/json
< Transfer-Encoding: chunked
< Server: Jetty(6.1.18)
<
{ [data not shown]
100    91    0    62  100    29     40     18  0:00:01
0:00:01 --:--:--    40
* Connection #0 to host localhost left intact
* Closing connection #0
* About to connect() to localhost port 8080 (#0)
```

```
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /rest/rest/scheduler/isconnected HTTP/1.1
> User-Agent: curl/7.21.7 (x86_64-redhat-linux-gnu)
libcurl/7.21.7 NSS/3.13.5.0 zlib/1.2.5 libidn/1.22
libssh2/1.2.7
> Host: localhost:8080
> Accept: */*
>
sessionid:2112db2b13e0cfa4c207ce02a2034ac7a124c322112db2b13e0c
fa4c208000
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Transfer-Encoding: chunked
< Server: Jetty(6.1.18)
<
* Connection #0 to host localhost left intact
* Closing connection #0
true
```

9.4.2 List of Running Processes

In order to check if all required processes are running to let *ProActive Cloud Job Scheduler* work correctly, the following commands should be typed:

```
ps -eaf | grep RMStarter # Check if the RM Service is up

# OUTPUT expected similar to:

lcantelm  4553  4534  1 14:40 pts/6    00:00:17
/home/lcantelm/application/jdk1.6.0_43/jre/bin/java -
Dproactive.home=/user/lcantelm/home/ProActiveCloudJobScheduler
_3.2/scheduling -
Dpa.rm.home=/user/lcantelm/home/ProActiveCloudJobScheduler_3.2
/scheduling -
Dpa.scheduler.home=/user/lcantelm/home/ProActiveCloudJobSchedu
ler_3.2/scheduling -
Dpa.openstack.home=/user/lcantelm/home/ProActiveCloudJobSchedu
ler_3.2/scheduling -Djava.security.manager -
Djava.security.policy=file:/user/lcantelm/home/ProActiveCloudJ
obScheduler_3.2/scheduling/config/security.java.policy-server
-
```

```
Dderby.stream.error.file=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/.logs/derby.log -Xms128m -Xmx1048m -
Dproactive.configuration=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/config/proactive/ProActiveConfigurati
on.xml -Djava.io.tmpdir=/tmp -Dproactive.rmi.port=1099 -
Dlog4j.configuration=file:/user/lcantelm/home/ProActiveCloudJo
bScheduler_3.2/scheduling/config/log4j/rm-log4j-server
org.ow2.proactive.resourcemanager.utils.RMStarter -ln
```

Then, check if the default LocalInfrastructure with four nodes is deployed properly, by typing

```
bash-4.2$ ps -eaf | grep java | grep LocalNodes


# OUTPUT expected similar to:


lcantelm  4680  4553  0 14:41 pts/6    00:00:03
/user/lcantelm/home/application/jdk1.6.0_43/bin/java -
Djava.security.policy=/user/lcantelm/home/ProActiveCloudJobSch
eduler_3.2/scheduling/config/security.java.policy-client -
Dlog4j.configuration=file:/user/lcantelm/home/ProActiveCloudJo
bScheduler_3.2/scheduling/config/log4j/log4j-defaultNode -
Dproactive.configuration=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/config/proactive/ProActiveConfigurati
on.xml -
Dpa.rm.home=/user/lcantelm/home/ProActiveCloudJobScheduler_3.2
/scheduling/ -
cp :/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/schedul
ing/dist/lib/jruby.jar:/user/lcantelm/home/ProActiveCloudJobSc
heduler_3.2/scheduling/dist/lib/sigar/sigar.jar:/user/lcantelm
/home/ProActiveCloudJobScheduler_3.2/scheduling/dist/lib/jytho
n-2.5.4-
rc1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/groovy-all-
2.1.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/dist/lib/commons-logging-
1.1.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/dist/lib/ProActive_Scheduler-
core.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sc
heduling/dist/lib/ProActive_SRM-
common.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/dist/lib/ProActive_ResourceManager.jar:/user/lcante
lm/home/ProActiveCloudJobScheduler_3.2/scheduling/dist/lib/Pro
Active_Scheduler-
worker.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/dist/lib/ProActive_Scheduler-
mapreduce.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3
.2/scheduling/dist/lib/commons-httpclient-
3.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/commons-codec-
1.3.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/ProActive.jar:/user/lcantelm/home/ProActiveCl
oudJobScheduler_3.2/scheduling/addons:/user/lcantelm/home/ProA
```

```
ctiveCloudJobScheduler_3.2/scheduling/addons/json-path-
0.8.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/jackson-core-asl-
1.9.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/guava-
14.0.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/addons/commons-lang-
2.6.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/addons/json-smart-
1.1.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/jackson-mapper-asl-
1.9.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/rest-api-schemas-
5.1.0.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/vcloud-java-sdk-5.1.0.jar
org.ow2.proactive.resourcemanager.utils.RMNodeStarter -v
UlNBCjEwMjQKUlNBL0VDQi9QS0NTMVBhZGRpbmcKEi9owV3rhwjqQTPLrc4pdL
eC03iaZRH+mqfRFwRBnFLkGKiQeCcNHAlXXhiuLOb3YRdxXgbvH041oXqspgON
IaDeaIh7jTfpqKgWTHxMQZc3vIJZIphOTDDww5vavqylct+FPw7dyiJMHoj/1I
1nkXFMiO8TYV7XpOfb14KpRmiuefwszP6q9EtQ25u5n0V6ODA0dt2V/cQO6MLR
utAK1AuLaj6oULaSaeXkk0c1cvvW9mVdsybBoE0NgWGrGqfGTGtI9cLbN4LIMB
OdeU++AYfyUOyYCyWcZjziGYGIBqpo0+/nkIBHoH3jTubLu0S3ObFb3/CoWALy
Al39Zj7+yb9C7r7OYtRjZWOOH/ecUBfdP+pnJSXZ4NoUcs2UJp2r -n local-
LocalNodes-0 -s LocalNodes -r rmi://172.16.0.1:1099/ -p 30000

lcantelm  4681  4553  0 14:41 pts/6    00:00:03
/user/lcantelm/home/application/jdk1.6.0_43/bin/java -
Djava.security.policy=/user/lcantelm/home/ProActiveCloudJobSch
eduler_3.2/scheduling/config/security.java.policy-client -
Dlog4j.configuration=file:/user/lcantelm/home/ProActiveCloudJo
bScheduler_3.2/scheduling/config/log4j/log4j-defaultNode -
Dproactive.configuration=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/config/proactive/ProActiveConfigurati
on.xml -
Dpa.rm.home=/user/lcantelm/home/ProActiveCloudJobScheduler_3.2
/scheduling/ -
cp :/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/schedul
ing/dist/lib/jruby.jar:/user/lcantelm/home/ProActiveCloudJobSc
heduler_3.2/scheduling/dist/lib/sigar/sigar.jar:/user/lcantelm
/home/ProActiveCloudJobScheduler_3.2/scheduling/dist/lib/jytho
n-2.5.4-
rc1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/groovy-all-
2.1.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/dist/lib/commons-logging-
1.1.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/dist/lib/ProActive_Scheduler-
core.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sc
heduling/dist/lib/ProActive_SRM-
common.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/dist/lib/ProActive_ResourceManager.jar:/user/lcante
lm/home/ProActiveCloudJobScheduler_3.2/scheduling/dist/lib/Pro
Active_Scheduler-
worker.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/dist/lib/ProActive_Scheduler-
mapreduce.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3
```

```
.2/scheduling/dist/lib/commons-httpclient-
3.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/commons-codec-
1.3.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/ProActive.jar:/user/lcantelm/home/ProActiveCl
oudJobScheduler_3.2/scheduling/addons:/user/lcantelm/home/ProA
ctiveCloudJobScheduler_3.2/scheduling/addons/json-path-
0.8.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/jackson-core-asl-
1.9.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/guava-
14.0.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/addons/commons-lang-
2.6.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/addons/json-smart-
1.1.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/jackson-mapper-asl-
1.9.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/rest-api-schemas-
5.1.0.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/vcloud-java-sdk-5.1.0.jar
org.ow2.proactive.resourcemanager.utils.RMNodeStarter -v
UlNBCjEwMjQKUlNBL0VDQi9QS0NTMVBhZGRpbmcKEi9owV3rhwjqQTPLrc4pdL
eC03iaZRH+mqfRFwRBnFLkGKiQeCcNHAlXXhiuLOb3YRdxXgbvH041oXqspgON
IaDeaIh7jTfpqKgWTHxMQZc3vIJZIphOTDDww5vavqylct+FPw7dyiJMHoj/1I
1nkXFMiO8TYV7XpOfb14KpRmiuefwszP6q9EtQ25u5n0V6ODA0dt2V/cQO6MLR
utAK1AuLaj6oULaSaeXkk0c1cvvW9mVdsybBoE0NgWGrGqfGTGtI9cLbN4LIMB
OdeU++AYfyUOyYCyWcZjziGYGIBqpo0+/nkIBHoH3jTubLu0S3ObFb3/CoWALy
Al39Zj7+yb9C7r7OYtRjZWOOH/ecUBfdP+pnJSXZ4NoUcs2UJp2r -n local-
LocalNodes-3 -s LocalNodes -r rmi://172.16.0.1:1099/ -p 30000

lcantelm  4683  4553  0 14:41 pts/6    00:00:03
/user/lcantelm/home/application/jdk1.6.0_43/bin/java -
Djava.security.policy=/user/lcantelm/home/ProActiveCloudJobSch
eduler_3.2/scheduling/config/security.java.policy-client -
Dlog4j.configuration=file:/user/lcantelm/home/ProActiveCloudJo
bScheduler_3.2/scheduling/config/log4j/log4j-defaultNode -
Dproactive.configuration=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/config/proactive/ProActiveConfigurati
on.xml -
Dpa.rm.home=/user/lcantelm/home/ProActiveCloudJobScheduler_3.2
/scheduling/ -
cp :/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/schedul
ing/dist/lib/jruby.jar:/user/lcantelm/home/ProActiveCloudJobSc
heduler_3.2/scheduling/dist/lib/sigar/sigar.jar:/user/lcantelm
/home/ProActiveCloudJobScheduler_3.2/scheduling/dist/lib/jytho
n-2.5.4-
rc1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/groovy-all-
2.1.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/dist/lib/commons-logging-
1.1.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/dist/lib/ProActive_Scheduler-
core.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sc
heduling/dist/lib/ProActive_SRM-
common.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
```

```
scheduling/dist/lib/ProActive_ResourceManager.jar:/user/lcante
lm/home/ProActiveCloudJobScheduler_3.2/scheduling/dist/lib/Pro
Active_Scheduler-
worker.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/dist/lib/ProActive_Scheduler-
mapreduce.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3
.2/scheduling/dist/lib/commons-httpclient-
3.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/commons-codec-
1.3.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/ProActive.jar:/user/lcantelm/home/ProActiveCl
oudJobScheduler_3.2/scheduling/addons:/user/lcantelm/home/ProA
ctiveCloudJobScheduler_3.2/scheduling/addons/json-path-
0.8.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/jackson-core-asl-
1.9.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/guava-
14.0.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/addons/commons-lang-
2.6.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/addons/json-smart-
1.1.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/jackson-mapper-asl-
1.9.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/rest-api-schemas-
5.1.0.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/vcloud-java-sdk-5.1.0.jar
org.ow2.proactive.resourcemanager.utils.RMNodeStarter -v
UlNBCjEwMjQKUlNBL0VDQi9QS0NTMVBhZGRpbmcKEi9owV3rhwjqQTPLrc4pdL
eC03iaZRH+mqfRFwRBnFLkGKiQeCcNHAlXXhiuLOb3YRdxXgbvH041oXqspgON
IaDeaIh7jTfpqKgWTHxMQZc3vIJZIphOTDDww5vavqylct+FPw7dyiJMHoj/1I
1nkXFMiO8TYV7XpOfb14KpRmiuefwszP6q9EtQ25u5n0V6ODA0dt2V/cQO6MLR
utAK1AuLaj6oULaSaeXkk0c1cvvW9mVdsybBoE0NgWGrGqfGTGtI9cLbN4LIMB
OdeU++AYfyUOyYCyWcZjziGYGIBqpo0+/nkIBHoH3jTubLu0S3ObFb3/CoWALy
Al39Zj7+yb9C7r7OYtRjZWOOH/ecUBfdP+pnJSXZ4NoUcs2UJp2r -n local-
LocalNodes-2 -s LocalNodes -r rmi://172.16.0.1:1099/ -p 30000

lcantelm  4685  4553  0 14:41 pts/6    00:00:03
/user/lcantelm/home/application/jdk1.6.0_43/bin/java -
Djava.security.policy=/user/lcantelm/home/ProActiveCloudJobSch
eduler_3.2/scheduling/config/security.java.policy-client -
Dlog4j.configuration=file:/user/lcantelm/home/ProActiveCloudJo
bScheduler_3.2/scheduling/config/log4j/log4j-defaultNode -
Dproactive.configuration=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/config/proactive/ProActiveConfigurati
on.xml -
Dpa.rm.home=/user/lcantelm/home/ProActiveCloudJobScheduler_3.2
/scheduling/ -
cp :/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/schedul
ing/dist/lib/jruby.jar:/user/lcantelm/home/ProActiveCloudJobSc
heduler_3.2/scheduling/dist/lib/sigar/sigar.jar:/user/lcantelm
/home/ProActiveCloudJobScheduler_3.2/scheduling/dist/lib/jytho
n-2.5.4-
rc1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/groovy-all-
2.1.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
```

```
cheduling/dist/lib/commons-logging-
1.1.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/dist/lib/ProActive_Scheduler-
core.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sc
heduling/dist/lib/ProActive_SRM-
common.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/dist/lib/ProActive_ResourceManager.jar:/user/lcante
lm/home/ProActiveCloudJobScheduler_3.2/scheduling/dist/lib/Pro
Active_Scheduler-
worker.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/dist/lib/ProActive_Scheduler-
mapreduce.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3
.2/scheduling/dist/lib/commons-httpclient-
3.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/commons-codec-
1.3.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/dist/lib/ProActive.jar:/user/lcantelm/home/ProActiveCl
oudJobScheduler_3.2/scheduling/addons:/user/lcantelm/home/ProA
ctiveCloudJobScheduler_3.2/scheduling/addons/json-path-
0.8.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/jackson-core-asl-
1.9.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/guava-
14.0.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/
scheduling/addons/commons-lang-
2.6.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/sch
eduling/addons/json-smart-
1.1.1.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/jackson-mapper-asl-
1.9.5.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/rest-api-schemas-
5.1.0.jar:/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/s
cheduling/addons/vcloud-java-sdk-5.1.0.jar
org.ow2.proactive.resourcemanager.utils.RMNodeStarter -v
UlNBCjEwMjQKUlNBL0VDQi9QS0NTMVBhZGRpbmcKEi9owV3rhwjqQTPLrc4pdL
eC03iaZRH+mqfRFwRBnFLkGKiQeCcNHAlXXhiuLOb3YRdxXgbvH041oXqspgON
IaDeaIh7jTfpqKgWTHxMQZc3vIJZIphOTDDww5vavqylct+FPw7dyiJMHoj/1I
1nkXFMiO8TYV7XpOfb14KpRmiuefwszP6q9EtQ25u5n0V6ODA0dt2V/cQO6MLR
utAK1AuLaj6oULaSaeXkk0c1cvvW9mVdsybBoE0NgWGrGqfGTGtI9cLbN4LIMB
OdeU++AYfyUOyYCyWcZjziGYGIBqpo0+/nkIBHoH3jTubLu0S3ObFb3/CoWALy
Al39Zj7+yb9C7r7OYtRjZWOOH/ecUBfdP+pnJSXZ4NoUcs2UJp2r -n local-
LocalNodes-1 -s LocalNodes -r rmi://172.16.0.1:1099/ -p 30000



bash-4.2$ ps -eaf | grep SchedulerStarter # Check if the
Scheduler Service is up


# OUTPUT expected similar to:


lcantelm  4629  4534  1 14:41 pts/6    00:00:14
/home/lcantelm/application/jdk1.6.0_43/jre/bin/java -
Dproactive.home=/user/lcantelm/home/ProActiveCloudJobScheduler
```

```
_3.2/scheduling -
Dpa.rm.home=/user/lcantelm/home/ProActiveCloudJobScheduler_3.2
/scheduling -
Dpa.scheduler.home=/user/lcantelm/home/ProActiveCloudJobSchedu
ler_3.2/scheduling -
Dpa.openstack.home=/user/lcantelm/home/ProActiveCloudJobSchedu
ler_3.2/scheduling -Djava.security.manager -
Djava.security.policy=file:/user/lcantelm/home/ProActiveCloudJ
obScheduler_3.2/scheduling/config/security.java.policy-server
-
Dderby.stream.error.file=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/.logs/derby.log -Xms128m -Xmx1048m -
Dproactive.configuration=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/config/proactive/ProActiveConfigurati
on.xml -Djava.io.tmpdir=/tmp -
Dlog4j.configuration=file:/user/lcantelm/home/ProActiveCloudJo
bScheduler_3.2/scheduling/config/log4j/scheduler-log4j-server
-Dproactive.rmi.port=1100
org.ow2.proactive.scheduler.util.SchedulerStarter -u
rmi://172.16.0.1:1099


bash-4.2$ ps -eaf | grep JettyLauncher # Check if the Jetty
Server is up


# OUTPUT expected similar to:


lcantelm  4901  4534 10 14:41 pts/6    00:02:00
/home/lcantelm/application/jdk1.6.0_43/jre/bin/java -
Djava.security.manager -
Djava.security.policy=file:/user/lcantelm/home/ProActiveCloudJ
obScheduler_3.2/scheduling/config/security.java.policy-client
-
Dpa.rm.home=/user/lcantelm/home/ProActiveCloudJobScheduler_3.2
/scheduling -
Dpa.scheduler.home=/user/lcantelm/home/ProActiveCloudJobSchedu
ler_3.2/scheduling -
Dpa.openstack.home=/user/lcantelm/home/ProActiveCloudJobSchedu
ler_3.2/scheduling -
Dproactive.configuration=/user/lcantelm/home/ProActiveCloudJob
Scheduler_3.2/scheduling/config/proactive/ProActiveConfigurati
on.xml -Dproactive.rmi.port=1101
org.ow2.proactive.utils.JettyLauncher -p 8080
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/rest
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/rm
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
dist/war/scheduler
```

9.4.3 Network interfaces Up & Open

- Make sure that network interfaces, your host relies on for TCP/IP communications, are up and consistent with your IPv*x* addressing plan. The following commands may be useful:

```
ifconfig -a # for Unix systems
ipconfig /all # for Windows systems
```

- TCP port 8080 should be accessible to the client. To test if it is working, a tcp connection handshake via *telnet <ip_address> 8080* should succeed (be sure that the firewall does not avoid that), as follows

```
bash-4.2$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

- Default configuration may be not enough to guarantee the right working. Make sure that your ProActive TCP/IP configuration reflects your IPvx networking plan, when launching each service.

9.4.4 Databases

Our *ProAcitve Cloud Job Scheduler* relies on Hibernate framework for data persistence automation and actually supports Java DERBY and MySQL databases. The default database is Java Derby DB, which is started automatically when launching the Scheduler or the Resource Manager Service and, so, it does not appear as a distinct process inside the operating system. The only way to check if anything goes wrong is to check the related log file, that is *$JOB_SCHEDULER_GE_HOME/scheduling/.logs/derby.log*.

The configuration files related to both the Scheduler and RM Service are inside *$JOB_SCHEDULER_GE_HOME/scheduling/config* folder. To be more precise, at *$JOB_SCHEDULER_GE_HOME/scheduling/config/[rm|scheduler/database/hibernate/hibernate.cfg.xml*.

Finally, at Start Scheduler section of ProActive Administration Guide, the default configuration is explained. In case of MySQL DB, it is just a matter of uncommenting the right part, by tuning it at your needs, and comment that related to Java DERBY.

9.5 Diagnosis Procedures

The Diagnosis Procedures are the first steps that a System Administrator will take to locate the source of an error in a GEi. Once the nature of the error is identified with these tests, the system admin will very often have to resort to more concrete and specific testing to pinpoint the exact point of error and a possible solution. Such specific testing is out of the scope of this section.

At this end, the nature of the *Job Scheduler GE*, which deals with highly distributed systems, is not helpful for identifying exactly the fault. By the way, as you might guess, the strongest tool you could rely on is an efficient logging system architecture, which centralizes all the logs in the *$JOB_SCHEDULER_GE_HOME/scheduling/.logs* folder. There, you can find per-job,

per-task, per-service and per-node log files, you might check at your needs or, as *best practise*, periodically.

```
bash-4.2$ ls -R $JOB_SCHEDULER_GE_HOME/scheduling/.logs/
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
.logs/:
derby.log       jobs                        Node-local-
LocalNodes-1.log  Node-local-LocalNodes-3.log  RM-stdout.log
       Scheduler-stdout.log
Jetty-stdout.log  Node-local-LocalNodes-0.log  Node-local-
LocalNodes-2.log  RM.log                    Scheduler.log

/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
.logs/jobs:
1  10000  10001  10002      10003  10004  10005  10006  10007
```

9.5.1 Resource availability

At first, in order to deploy a truly healthy GE instance, we recommend you to avoid the default node source deployment (offered in the release as the tool to get started), by relying on truly distributed nodes for computation.

Then, as you may notice at *List of running processes* sub-section, both the RM and the Scheduler Service requires *-Xms128m -Xmx1048m*, whose meaning is explained at ORACLE "-X Command-line Options" and which justifies what already stated at *Minimal System Requirements*:

- 3GB RAM for Test Environment - 5+GB RAM for Production Environment
- 5GB free on the HDD for Test Environment - 30+GB free on the HDD for Production Environment

Whereas the last requirement is due to accomplish the *Download & Unzip* sub-section (that requires 3 GB more or less) and give a reasonable and indicative margin to use it in test/production environment. In fact, such requirement might be seen more as a *best practise*, which might be subjected to modification according to the amount of the data you need to process.

About the CPU, a good computation power, such as a CPU with 4+ cores, might be required.

About the NIC performance, being the Job Scheduler Generic Enabler the brain of a distributed system, where a lot of I/O operations might be required to accomplish the jobs, is reasonable to have a 1Gb/s NIC.

9.5.2 Remote Service Access

Our *ProActive Cloud Job Scheduler* is *pluggable* against the *Identity Management GEi*s and the *IaaS Data Center Resource Manager GEi*s, which are OpenStack-compliant by design. So, here we suggest a unique scenario to test both of them.

At first, it is required you have installed and run an OpenStack infrastructure or two of those GEis. If not available, for testing purposes only, we recommend to deploy it

inside a single VM, by leverging DevStack. Once the deployment is accomplished with all the OpenStack services up, proceed in the further steps:

1. retrieve the Keystone endpoint URL related to API v2.0 (in our case *http://172.16.0.254:5000/v2.0*), coming out from the last logs after the *./stack.sh* script execution.
2. Make sure that the Keystone service is reachable, by typing

```
$ telnet <KEYSTONE_IP_OR_HOSTNAME> <PORT>

# Expected output

Trying 172.16.0.254...
Connected to 172.16.0.254.
Escape character is '^]'.
```

afterwards, create a user *os_only_user* on the Keystone AAA system, as follows:

```
. ./git/devstack/openrc

# Default DevStack admin details
export OS_USERNAME=admin
export OS_PASSWORD=password

# Create a user existing only in the Keystone authentication
system (used specifically for testing the authentication login
chain)
keystone user-create --name os_only_user --pass password
```

and bind him to *demo* tenant.

According to your case, update your OpenStack properties at *$JOB_SCHEDULER_GE_HOME/scheduling/config/openstack/openstack.ini*, such as

```
...
# At the moment, the integration between ProActive
Authentication Authorization Accounting and OpenStack Keystone
concerns just the authentication phase.
# So that, the OpenStack Keystone roles/groups (important for
authorization) are overwritten (see
config/authentication/jaas.config for better understanding)
pa.openstack.keystone.authorization.special.admin.usernames=os
_only_user
pa.openstack.keystone.authorization.special.admin.groups=admin
,nsadmins
```

```
pa.openstack.keystone.authorization.default.groups=user,provid
ers


#-------------------------------------------------------
#----   OPENSTACK KEYSTONE PROPERTIES   ---------------
#-------------------------------------------------------


# OpenStack Keystone absolute endpoint uri: it is the prefix
for access REST API resources
pa.openstack.keystone.absolute.endpoint.uri=http://172.16.0.25
4:5000/v2.0
..
```

Now, it is required to create your own template image and the related *id* (*3e5095ed-6e97-4282-b78e-e088fabbb1cf* in our case), you might leverage for computation, where you must include our *$JOB_SCHEDULER_GE_HOME/scheduling* folder content, together with the -still obviously needed- JDK 6+. In particular, be sure that your image configuration matches with the *metadata* fetched at runtime at *$JOB_SCHEDULER_GE_HOME/scheduling/addons/org/ow2/proactive/iaas/nova/start-proactive-node*.

Then, we can launch the server by using the most generic protocol to avoid any networking issue:

```
bash-4.2$ $JAVA_HOME/bin/jrunscript -
Duser.dir=$JOB_SCHEDULER_GE_HOME/scheduling/bin -
Dproactive.net.interface=virbr4 -
Dproactive.communication.protocol=pamr
$JOB_SCHEDULER_GE_HOME/scheduling/bin/start-server.js
```

Finally, we can check the integrations by submitting the request for creating an infrastructure (at *demo* tenant) of just *1* instance at *demo*, having *flavor_id* equal to 2 (the small one):

```
bash-4.2$ curl -v -X POST -H "sessionid:`curl -X POST -d
"username=os_only_user&password=password"
http://localhost:8080/rest/rest/rm/login`" \
 -d "nodeSourceName=NOVA_STATIC" \
 -d
"infrastructureType=org.ow2.proactive.iaas.nova.NovaInfrastruc
ture" \
 -d
"policyType=org.ow2.proactive.resourcemanager.nodesource.polic
y.StaticPolicy" \
 -d "policyParameters=ALL" \
 -d "policyParameters=ME" \
 -d "infrastructureParameters=pamr://0/" \
```

```
 -d "infrastructureParameters=1" \
 -d "infrastructureParameters=http://172.16.0.254:5000/v2.0" \
 -d "infrastructureParameters=monitoringEnabled" \
 -d "infrastructureParameters=os_only_user" \
 -d "infrastructureParameters=password" \
 -d "infrastructureParameters=demo" \
 -d "infrastructureParameters=3e5095ed-6e97-4282-b78e-
e088fabbb1cf" \
 -d "infrastructureParameters=2" \
 --data-urlencode "infrastructureFileParameters=`cat
$JOB_SCHEDULER_GE_HOME/scheduling/config/authentication/rm.cre
d`" \
 http://localhost:8080/rest/rest/rm/nodesource/create
  % Total    % Received % Xferd  Average Speed   Time    Time
Time  Current
                                 Dload  Upload   Total   Spent
Left  Speed
100  1067    0  1028  100    39    750     28  0:00:01
0:00:01 --:--:--   750
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /rest/rest/rm/nodesource/create HTTP/1.1
> User-Agent: curl/7.21.7 (x86_64-redhat-linux-gnu)
libcurl/7.21.7 NSS/3.13.5.0 zlib/1.2.5 libidn/1.22
libssh2/1.2.7
> Host: localhost:8080
> Accept: */*
> sessionid:MIIC-
wYJKoZIhvcNAQcCoIIC8DCCAuwCAQExCTAHBgUrDgMCGjCCAVUGCSqGSIb3DQE
HAaCCAUYEggFCeyJhY2Nlc3MiOiB7InRva2VuIjogeyJpc3N1ZWRfYXQiOiAiM
jAxMy0xMi0xOVQxNjo0MDoxMS41NjM4NzgiLCAiZXhwaXJlcyI6ICIyMDEzLTE
yLTIwVDE2OjQwOjExWiIsICJpZCI6ICJwbGFjZWhvbGRlciJ9LCAic2VydmljZ
UNhdGFsb2ciOiBbXSwgInVzZXIiOiB7InVzZXJuYW1lIjogIm9zX29ubHlfdXN
lciIsICJyb2xlc19saW5rcyI6IFtdLCAiaWQiOiAiYzk2ZTEzNDRiYTU5NGI2M
Tk0YjUzMTI3NDkzZmIyMjIiLCAicm9sZXMiOiBbXSwgIm5hbWUiOiAib3Nfb25
seV91c2VyIn0sICJtZXRhZGF0YSI6IHsiaXNfYWRtaW4iOiAwLCAicm9sZXMiO
iBbXX19fTGCAYEwggF9AgEBMFwwVzELMAkGA1UEBhMCVVMxDjAMBgNVBAgMBVV
uc2V0MQ4wDAYDVQQHDAVVbnNldDEOMAwGA1UECgwFVW5zZXQxGDAWBgNVBAMMD
3d3dy5leGFtcGxlLmNvbQIBATAHBgUrDgMCGjANBgkqhkiG9w0BAQEFAASCAQC
odIu9aKCSm7sS-C7+OeMvMl8Qrw3-e-ouNSG2uNmTR5uB0WaEm7r8-
BsC24wGmoRpiHvQqkpw1f2QTke6o1ZVB18GZLMj+-
uGWwmVDokr0wPxGiMCkbNZ4WikQ3sTpfM9dHYxpbr7k0kjRMu7-
Jj5d7xCbDmtQyYXxiJ0Wd72cUZmGLdZASqKrk6BAIPOR9i4Jq1ohPBzPmCHoky
uIw9M9oyQPxg7EPK8MI8eqINMEHHPIO2JYfWTaBiZ-
```

```
6VMt2gOxuq1dvuyBiCBv0Zl6f8AupTPkABBgGGGPXTQfxZR-
xozBvEbEHJEDPHuJYMXBipwiCoOeh67qz5j9c2uE7Gu
> Content-Length: 1043
> Content-Type: application/x-www-form-urlencoded
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
< HTTP/1.1 200 OK
< Content-Type: application/json
< Transfer-Encoding: chunked
< Server: Jetty(6.1.18)
<
* Connection #0 to host localhost left intact
* Closing connection #0
true
```

As you can see, the authentication is done through the Keystone (note the longest string of the *sessionid*, representing the OpenStack *unscoped token*). Moreover, the *true* returned states that the nodesource creation has been submitted successfully. Finally, periodically check the OpenStack VM instance logs and the related node availability, such as:

```
bash-4.2$ curl -v -X GET -H sessionid:`curl -X POST -d
"username=os_only_user&password=password"
http://localhost:8080/rest/rest/rm/login`
http://localhost:8080/rest/rest/rm/state | python -mjson.tool
  % Total    % Received % Xferd  Average Speed   Time    Time
Time  Current
                                 Dload  Upload   Total   Spent
Left  Speed
100  1067    0  1028  100    39   1135     43 --:--:-- --:--:-
- --:--:--  1135
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1...   % Total    % Received % Xferd
Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent
Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-
- --:--:--     0connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /rest/rest/rm/state HTTP/1.1
> User-Agent: curl/7.21.7 (x86_64-redhat-linux-gnu)
libcurl/7.21.7 NSS/3.13.5.0 zlib/1.2.5 libidn/1.22
libssh2/1.2.7
```

```
> Host: localhost:8080
> Accept: */*
> sessionid:MIIC-
wYJKoZIhvcNAQcCoIIC8DCCAuwCAQExCTAHBgUrDgMCGjCCAVUGCSqGSIb3DQE
HAaCCAUYEggFCeyJhY2Nlc3MiOiB7InRva2VuIjogeyJpc3N1ZWRfYXQiOiAiM
jAxMy0xMi0xOVQxNzozODo1NS4zNDI0MzQiLCAiZXhwaXJlcyI6ICIyMDEzLTE
yLTIwVDE3OjM4OjU1WiIsICJpZCI6ICJwbGFjZWhvbGRlciJ9LCAic2VydmljZ
UNhdGFsb2ciOiBbXSwgInVzZXIiOiB7InVzZXJuYW1lIjogIm9zX29ubHlfdXN
lciIsICJyb2xlc19saW5rcyI6IFtdLCAiaWQiOiAiYzk2ZTEzNDRiYTU5NGI2M
Tk0YjUzMTI3NDkzZmIyMjIiLCAicm9sZXMiOiBbXSwgIm5hbWUiOiAib3Nfb25
seV91c2VyIn0sICJtZXRhZGF0YSI6IHsiaXNfYWRtaW4iOiAwLCAicm9sZXMiO
iBbXX19fTGCAYEwggF9AgEBMFwwVzELMAkGA1UEBhMCVVMxDjAMBgNVBAgMBVV
uc2V0MQ4wDAYDVQQHDAVVbnNldDEOMAwGA1UECgwFVW5zZXQxGDAWBgNVBAMMD
3d3dy5leGFtcGxlLmNvbQIBATAHBgUrDgMCGjANBgkqhkiG9w0BAQEFAASCAQA
6drQ6yIULUk9AMPy568+M7quPs-6m-
3Y+GRMkzNv7z6B8NAS3kSyqX4cUIjixQc3DAaPR7HFFsIhJErBPOR1T3Mh2k3+
8g5f47qpflZTXl1dWwvIng8jd8qv4foTCJohBoTXuN9d1af1KYEAgCR4zA69bb
eEcv9YjuBqRad7-0xHaqtJ+g25R4Cfk2aVMVaJnvX1nBIP-
xE4l4LjucIrJBi9D2f42q9aqlvv4NBj5a-
VnFylRcCuygnbXVxYD+D5n996gNcBz4GBEVDPqP04GvnvL53GIQQUpYIwKlYgL
HyYm2nybpbBiAJAaoNRQ526T3DnuEREw-yBjgNxIPxFz
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Transfer-Encoding: chunked
< Server: Jetty(6.1.18)
<
{ [data not shown]
100    68    0    68    0     0   1086      0 --:--:-- --:--:-
- --:--:--  1114
* Connection #0 to host localhost left intact
* Closing connection #0
{
    "freeNodesNumber": 5,
    "totalAliveNodesNumber": 5,
    "totalNodesNumber": 5
}
```

where we registered the increasing of the default number of nodes from 4 to 5. Enjoy the computation, targeting your fresh new created infrastructure having only one node, as required, but could hundreds!

9.5.3 Resource consumption

N/A: Resource consumption may vary according to the type of deployment and of jobs submitted, so we invite to refer you to *Minimal System Requirements* sub-section.

9.5.4 I/O flows

By default, the port where the *ProActive Cloud Job Scheduler* REST Server listens to is the *8080*; while, for back-end nodes communications, the port *1099*, *1100* and *1101* are required to be opened, due to the *rmiregistry* each service listens to.

Anyway, since the Job Scheduler has the possibility to communicate with several protocols, both secure and unsecure, we list the complete list of the ports, whose TCP configuration you might be aware:

```
bash-4.2$ java -jar
$JOB_SCHEDULER_GE_HOME/scheduling/dist/lib/ProActive.jar |
grep port
    INTEGER    proactive.communication.rmissh.port [null]
    INTEGER    proactive.http.port [null]
    INTEGER    proactive.rmi.port [1099]
    INTEGER    proactive.runtime.broadcast.port [4554]
    INTEGER    proactive.pnps.port [0]
    INTEGER    proactive.amqp_federation.ssh.port [null]
    INTEGER
proactive.communication.amqp_federation.broker.port [5672]
    INTEGER    proactive.pamr.router.port [33647]
    INTEGER    proactive.pamrssh.port [null]
    INTEGER    proactive.amqp.ssh.port [null]
    INTEGER    proactive.communication.amqp.broker.port [5672]
    INTEGER    proactive.pnp.port [0]
```

Finally, for a truly understanding of the possible I/O flows, we recommends the following links:

- ProActive Network Configuration
- Using SSH tunneling for RMI or HTTP communications

9.6 Appendix - ProActive JVM Properties & Configuration

ProActive JVM (that is a JVM launched with *ProActive.jar* library) comes with a great amount of tunable properties according to your needs and to ProActive Configuration documentation. Here follows the list of the most up-to-date ones (default values are between square brackets):

```
bash-4.2$ java -jar
$JOB_SCHEDULER_GE_HOME/scheduling/dist/lib/ProActive.jar
...
Available properties:
```

```
From class
org.objectweb.proactive.core.ssh.proxycommand.ProxyCommandConf
ig
        STRING  proactive.communication.ssh.proxy.gateway
[null]
        STRING  proactive.communication.ssh.proxy.out_gateway
[null]
        BOOLEAN proactive.communication.ssh.try_proxy_command
[false]
From class org.objectweb.proactive.extensions.pamr.PAMRConfig
        INTEGER proactive.pamr.agent.id [null]
        STRING  proactive.pamr.agent.magic_cookie [null]
        INTEGER proactive.pamr.connect_timeout [3000]
        STRING  proactive.pamr.router.address [null]
        INTEGER proactive.pamr.router.port [33647]
        STRING  proactive.pamr.socketfactory [plain]
        INTEGER proactive.pamrssh.connect_timeout [null]
        INTEGER proactive.pamrssh.gc_idletime [null]
        INTEGER proactive.pamrssh.gc_period [null]
        STRING  proactive.pamrssh.key_directory [null]
        STRING  proactive.pamrssh.known_hosts [null]
        INTEGER proactive.pamrssh.port [null]
        STRING  proactive.pamrssh.username [null]
From class
org.objectweb.proactive.extensions.pnpssl.PNPSslConfig
        BOOLEAN proactive.pnps.authenticate [false]
        INTEGER proactive.pnps.default_heartbeat [9000]
        INTEGER proactive.pnps.idle_timeout [60000]
        STRING  proactive.pnps.keystore [null]
        INTEGER proactive.pnps.port [0]
From class org.objectweb.proactive.extensions.amqp.AMQPConfig
        STRING  proactive.amqp.ssh.key_directory [null]
        STRING  proactive.amqp.ssh.known_hosts [null]
        INTEGER proactive.amqp.ssh.port [null]
        STRING  proactive.amqp.ssh.username [null]
        STRING  proactive.communication.amqp.broker.address
[localhost]
        STRING  proactive.communication.amqp.broker.password
[guest]
```

```
        INTEGER proactive.communication.amqp.broker.port
[5672]

        STRING  proactive.communication.amqp.broker.user
[guest]

        STRING  proactive.communication.amqp.broker.vhost [/]

        STRING
proactive.communication.amqp.discover_exchange_name
[proactive.remoteobject.amqp_discover_exchange]

        STRING  proactive.communication.amqp.rpc_exchange_name
[proactive.remoteobject.amqp_rpc_exchange]

        LONG    proactive.communication.amqp.rpc_timeout
[10000]

        STRING  proactive.communication.amqp.socketfactory
[plain]

From class
org.objectweb.proactive.extensions.amqp.federation.AMQPFederat
ionConfig

        STRING  proactive.amqp_federation.ssh.key_directory
[null]

        STRING  proactive.amqp_federation.ssh.known_hosts
[null]

        INTEGER proactive.amqp_federation.ssh.port [null]

        STRING  proactive.amqp_federation.ssh.username [null]

        STRING
proactive.communication.amqp_federation.broker.address
[localhost]

        STRING
proactive.communication.amqp_federation.broker.mapping_file
[null]

        STRING
proactive.communication.amqp_federation.broker.password
[guest]

        INTEGER
proactive.communication.amqp_federation.broker.port [5672]

        STRING
proactive.communication.amqp_federation.broker.user [guest]

        STRING
proactive.communication.amqp_federation.broker.vhost [/]

        STRING
proactive.communication.amqp_federation.discover_exchange_name
[proactive.remoteobject.amqp_federation_discover_exchange]

        LONG
proactive.communication.amqp_federation.ping_timeout [5000]

        STRING
proactive.communication.amqp_federation.rpc_exchange_name
[proactive.remoteobject.amqp_federation_rpc_exchange]
```

```
        STRING
proactive.communication.amqp_federation.rpc_reply_exchange_nam
e [proactive.remoteobject.amqp_federation_rpc_reply_exchange]
        LONG
proactive.communication.amqp_federation.rpc_timeout [10000]
        STRING
proactive.communication.amqp_federation.socketfactory [plain]
From class
org.objectweb.proactive.core.config.CentralPAPropertyRepositor
y
        STRING  catalina.base [null]
        INTEGER components.creation.timeout [10000]
        STRING  fractal.provider [null]
        STRING  gcm.provider
[org.objectweb.proactive.core.component.Fractive]
        BOOLEAN java.net.preferIPv4Stack [null]
        BOOLEAN java.net.preferIPv6Addresses [null]
        STRING  java.rmi.server.codebase [null]
        STRING  java.security.auth.login.config [null]
        STRING  java.security.policy [null]
        STRING  javax.xml.transform.TransformerFactory [null]
        STRING  log4j.configuration [null]
        BOOLEAN proactive.classloading.useHTTP [true]
        STRING  proactive.codebase [null]
        STRING  proactive.communication.additional_protocols
[null]
        STRING  proactive.communication.benchmark.class
[org.objectweb.proactive.core.remoteobject.benchmark.Selection
Only]
        STRING  proactive.communication.benchmark.parameter
[null]
        STRING  proactive.communication.protocol [rmi]
        STRING  proactive.communication.protocols.order [null]
        INTEGER proactive.communication.rmissh.connect_timeout
[null]
        INTEGER proactive.communication.rmissh.gc_idletime
[null]
        INTEGER proactive.communication.rmissh.gc_period
[null]
        STRING  proactive.communication.rmissh.key_directory
[null]
        STRING  proactive.communication.rmissh.known_hosts
[null]
```

```
        INTEGER proactive.communication.rmissh.port [null]
        BOOLEAN
proactive.communication.rmissh.try_normal_first [false]
        STRING  proactive.communication.rmissh.username [null]
        BOOLEAN proactive.components.use_shortcuts [null]
        STRING  proactive.configuration [null]
        STRING  proactive.dataspaces.scratch_path [null]
        STRING  proactive.dataspaces.scratch_url [null]
        BOOLEAN proactive.debug [null]
        BOOLEAN proactive.dgc [false]
        INTEGER proactive.dgc.tta [null]
        INTEGER proactive.dgc.ttb [null]
        BOOLEAN proactive.exit_on_empty [false]
        INTEGER proactive.filetransfer.blocks_number [8]
        INTEGER proactive.filetransfer.blocks_size_kb [512]
        INTEGER proactive.filetransfer.buffer_size_kb [256]
        INTEGER proactive.filetransfer.services_number [16]
        BOOLEAN proactive.future.ac [true]
        LONG    proactive.future.synchrequest.timeout [0]
        INTEGER proactive.futuremonitoring.ttm [null]
        STRING  proactive.gcmd.unix.shell [/bin/sh]
        STRING  proactive.home [null]
        STRING  proactive.hostname [null]
        INTEGER proactive.http.connect_timeout [null]
        STRING  proactive.http.jetty.connector [null]
        STRING  proactive.http.jetty.xml [null]
        INTEGER proactive.http.port [null]
        BOOLEAN proactive.implicitgetstubonthis [false]
        BOOLEAN proactive.legacy.parser [null]
        STRING  proactive.locationserver
[org.objectweb.proactive.ext.locationserver.LocationServer]
        STRING  proactive.locationserver.rmi
[//localhost/LocationServer]
        STRING  proactive.log4j.appender.provider
[org.objectweb.proactive.core.util.log.remote.ThrottlingProvid
er]
        STRING  proactive.log4j.collector [null]
        BOOLEAN proactive.masterworker.compresstasks [false]
        INTEGER proactive.masterworker.pingperiod [10000]
```

```
        INTEGER proactive.mixedlocation.maxMigrationNb [-1]
        INTEGER proactive.mixedlocation.maxTimeOnSite [-1]
        INTEGER proactive.mixedlocation.ttl [-1]
        BOOLEAN proactive.mixedlocation.updatingForwarder
[false]
        STRING  proactive.mop.generatedclassesdir [null]
        BOOLEAN proactive.mop.writestubondisk [false]
        BOOLEAN proactive.net.disableIPv6 [true]
        STRING  proactive.net.interface [null]
        STRING  proactive.net.netmask [null]
        BOOLEAN proactive.net.nolocal [null]
        BOOLEAN proactive.net.noprivate [null]
        STRING  proactive.net.secondaryNames [null]
        STRING  proactive.os [null]
        INTEGER proactive.rmi.connect_timeout [null]
        INTEGER proactive.rmi.port [1099]
        BOOLEAN proactive.runtime.broadcast [false]
        STRING  proactive.runtime.broadcast.address
[230.0.1.1]
        STRING  proactive.runtime.broadcast.callback.class
[org.objectweb.proactive.core.runtime.broadcast.BTCallbackDefa
ultImpl]
        INTEGER proactive.runtime.broadcast.port [4554]
        STRING  proactive.runtime.domain.url [null]
        STRING  proactive.runtime.name [null]
        BOOLEAN proactive.runtime.ping [true]
        STRING  proactive.runtime.ping.url
[http://pinging.activeeon.com/ping.php]
        STRING  proactive.runtime.security [null]
        BOOLEAN proactive.runtime.stayalive [true]
        BOOLEAN proactive.securitymanager [true]
        STRING  proactive.ssl.cipher.suites
[SSL_DH_anon_WITH_RC4_128_MD5]
        BOOLEAN proactive.stack_trace [false]
        BOOLEAN proactive.tag.dsf [false]
        INTEGER proactive.tagmemory.lease.max [60]
        INTEGER proactive.tagmemory.lease.period [21]
        BOOLEAN proactive.test [null]
        INTEGER proactive.test.perf.duration [30000]
```

```
        INTEGER proactive.test.timeout [300000]
        STRING  proactive.timit.activation [null]
        BOOLEAN proactive.useIPaddress [null]
        INTEGER
proactive.vfsprovider.server.stream_autoclose_checking_millis
[null]
        INTEGER
proactive.vfsprovider.server.stream_open_maximum_period_millis
[null]
        BOOLEAN proactive.webservices.elementformdefault
[false]
        STRING  proactive.webservices.framework [cxf]
        BOOLEAN schema.validation [true]
From class org.objectweb.proactive.extensions.pnp.PNPConfig
        INTEGER proactive.pnp.default_heartbeat [9000]
        INTEGER proactive.pnp.idle_timeout [60000]
        INTEGER proactive.pnp.port [0]
```

9.7 Appendix B - Configuration and Launching via shell/batch scripts

9.7.1 Configuration

```
bash-4.2$ sh $JOB_SCHEDULER_GE_HOME/scheduling/bin/unix/rm-
start -h


usage: rm-start [-h] [-ln] [-t <timeout>]
 -h,--help                to display this help
 -ln,--localNodes         start the resource manager deploying
default 4 local nodes
 -t,--timeout <timeout>   Timeout used to start the nodes
(only usefull with local nodes, defaul: 30000ms)


 Notice : Without argument, the resource manager starts
without any computing node.



bash-4.2$ sh
$JOB_SCHEDULER_GE_HOME/scheduling/bin/unix/scheduler-start -h


usage: scheduler-start [-h] [-p <policy>] [-u <rmURL>]
```

```
 -h,--help                to display this help
 -p,--policy <policy>     the complete name of the scheduling
policy to use (default

org.ow2.proactive.scheduler.policy.DefaultPolicy)
 -u,--rmURL <rmURL>       the resource manager URL (default
localhost)


bash-4.2$ sh $JOB_SCHEDULER_GE_HOME/scheduling/bin/unix/jetty-
launcher -h
/user/lcantelm/home/ProActiveCloudJobScheduler_3.2/scheduling/
bin/unix/jetty-launcher <arguments> [options]


Paths for Web Applications (REST Server and both UI) point to
either
a valid .war file, or the extraction directory of a valid .war
file.


Arguments:
-A PATH   REST Server


Options:
-R PATH   RM Web UI
-S PATH   Scheduler Web UI
-r URL    RM Server URL
-s URL    Scheduler Server URL
-p PORT   HTTP server port
-q        quiet output
-v        Output jetty logs on stdout instead of tmp file
-Dxxx     JVM option
-h        print this message
```

As shown in the last script help, the JVM properties overriding can be achieve by simply appending the *-Dxxx=yyy* pattern after the script, where *xxx* is the JVM option and *yyy* is its value. About the other scripts, such a possibility is omitted since a more elegant solution involves the loading of dedicated *settings.ini* files for both the Resource Manager and the Scheduler Service, placed in their respective *$JOB_SCHEDULER_GE_HOME/scheduling/config* sub-folders.

9.7.2 Launching

Now we launch the RM with the default *LocalNodes* infrastructure in order to offer you a tool to get started.

```
bash-4.2$ sh $JOB_SCHEDULER_GE_HOME/scheduling/bin/unix/rm-
start -ln -
Dpa.scheduler.home=$JOB_SCHEDULER_GE_HOME/scheduling -
Dpa.rm.home=$JOB_SCHEDULER_GE_HOME/scheduling

Starting the resource manager...

The resource manager with 4 local nodes created on
rmi://172.16.0.1:1099/
```

and the Scheduler Service:

```
bash-4.2$ sh
$JOB_SCHEDULER_GE_HOME/scheduling/bin/unix/scheduler-start -
Dpa.scheduler.home=$JOB_SCHEDULER_GE_HOME/scheduling -
Dpa.rm.home=$JOB_SCHEDULER_GE_HOME/scheduling

Starting the scheduler...

Connected to the existing resource manager at
rmi://172.16.0.1:1099/

The scheduler created on rmi://172.16.0.1:1099/
```

In order that Jetty server can be launched without problems, make sure that ports 8080 and 5900 are free, leveraging *netstat*, as shown previously.

```
bash-4.2$ sh $JOB_SCHEDULER_GE_HOME/scheduling/bin/unix/jetty-
launcher -A
$JOB_SCHEDULER_GE_HOME/scheduling/dist/war/rest.war -R
$JOB_SCHEDULER_GE_HOME/scheduling/dist/war/rm.war -S
$JOB_SCHEDULER_GE_HOME/scheduling/dist/war/scheduler.war -s
rmi://172.16.0.1:1099/ -r rmi://172.16.0.1:1099/ -
Dpa.scheduler.home=$JOB_SCHEDULER_GE_HOME/scheduling -
Dpa.rm.home=$JOB_SCHEDULER_GE_HOME/scheduling

Deploying REST Server in
/tmp/proactive5816236884566515427.tmp/rest/

Deploying RM UI in /tmp/proactive3553440868177471094.tmp/rm/

Deploying Scheduling UI in
/tmp/proactive3409231045490516331.tmp/sched/

Jetty Launcher logs to /tmp/proactive59002354363286432257.tmp

Deployed application: http://localhost:8080/rest

Deployed application: http://localhost:8080/rm

Deployed application: http://localhost:8080/sched
```

10 Edgelets - Installation and Administration Guide

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

10.1 Introduction

Welcome to the Edgelets installation and administration guide.

This guide provides information for running the release 3.3 of the Edglets GE, and concentrates on the installation of the Edgelet manager, which is the a master node used to control, manage and monitor the distributed edgelet devices and services.

The specific deployment of a new device to the Edgelet network as well as the instanciation of a service are explainer in the **Edgelets - Installation and Administration Guide**.

10.1.1 Minimal system requirements

In order to install the edgelet master, you will need a linux machine running Ubuntu 12.04, for which .deb packages are available. It is possible to install it on other linux distributions, but this will need some adaptations (you will either have to recreate the packages or adapt them to your distribution). For the rest of this tutorial, we will assume that you are using Ubuntu 12.04.

The edgelet master is quite heavy and you will need to meet the following requirements:

- 30GB free on the HDD
- 4GB memory

10.2 System installation

The installation is separated in several steps which are explained afterwards. The basic idea is to build the edgelet master using edgelet tools. You will act as a node, and use a fake edgelet master in order to install your own instance of the edgelet master.

10.2.1 Step 1: installation of the edgelet node package

Retrieve the .deb package (chose the right version according to your system architecture, packets are compiled for Ubuntu 12.04)

- https://forge.fi-ware.org/frs/download.php/1416/CLOUD-Edgelets-client-3.3.0-amd64.deb
- https://forge.fi-ware.org/frs/download.php/1418/CLOUD-Edgelets-client-3.3.0-i386.deb

Then install it

```
# dpkg -i <file.deb>
```

Then install the dependencies

```
# apt-get install -f
```

Your machine is now capable of acting as an edgelet node.

10.2.2 Step 2: Client configuration

Edit /opt/slapos/slapos.cfg with the following content (be sure to adapt parameters like the interface name):

```
[slapos]
software_root = /opt/slapgrid
instance_root = /srv/slapgrid
master_url = http://127.0.0.1:5000/
# use a name of your convenience
computer_id = myedgeletmaster

[slapformat]
# Replace by your network interface like eth0, eth1,
slapbr0...
interface_name = interfacename
create_tap = false
partition_amount = 20
computer_xml = /opt/slapos/slapos.xml
log_file = /opt/slapos/log/slapos-node-format.log
partition_base_name = slappart
user_base_name = slapuser
tap_base_name = slaptap
# You can choose any other local network which does not
conflict with your
# current machine configuration
ipv4_local_network = 10.0.0.0/16
# Comment this if you are using native IPv6 and don't want to
use SlapOS tunnel
ipv6_interface = tapVPN

[slapproxy]
host = 127.0.0.1
port = 5000
database_uri = /opt/slapos/proxy.db
```

Note that the Edgelets GE is based on IPv6. In order to be sure to have this connectivity, you can configure an openvpn configuration to have IPv6 with the "tapVPN" interface, which currently access remote servers from slapos.org (whose software this GE is based on) to obtain IPv6 (ipv6_interface configuration item). You can disable it if you have native ipv6. Be sure that the openvpn daemon is started if you don't have native ipv6.

```
# service openvpn start
```

10.2.3 Step 3: run a fake local master

Slapproxy is a fake and light master, with minimal functionalities, used to bootstrap the real one.

```
# bin/slapproxy -vc /opt/slapos/slapos.cfg
```

Slapproxy will run in the foreground on port 5000, be sure to leave it running until the end of the installation procedure.

10.2.4 Step 4: local modification

Edit the file /opt/slapos/eggs/slapos.core-XXX/slapos/format.py and remove the two lines that refer to certificate_repository_path (around line 1100). This is because of a conflict on the use of certificates between the slapproxy and the real edgelet master.

```
if getattr(config, 'certificate_repository_path'):
 mkdir_p(config.certificate_repository_path, mode=0o700)
```

10.2.5 Step 5: slapformat

This tool wll create some system users, directories, network interfaces

```
# bin/slapformat -c /opt/slapos/slapos.cfg
```

10.2.6 Step 6: slapgrid

Slapgrid is the client daemon. It will query our fake master and will be responsible for installing the software, the instances and push back the client status to the master.

```
# bin/slapgrid -c /opt/slapos/slapos.cfg
```

You can check that it correctly connects to our fake master and that there is nothing to do at that time

10.2.7 Step 7: request installation of the master

slapconsole is a small python interpreter that allows to talk to slapproxy

```
$ bin/slapconsole /opt/slapos/slapos.cfg


import slapos.slap.slap
slap = slapos.slap.slap()
# Connect to slapproxy
slap.initializeConnection('http://127.0.0.1:5000/')
# Request to slapproxy the installation of a new software.
Here the url refers
# to the edgelet master software release
slap.registerSupply().supply(
    'https://raw.githubusercontent.com/fiware-
edgelets/slapos/master/software/erp5/software.cfg',
    computer_guid='myedgeletmaster')
```

10.2.8 Step 8: compile the software

Now that we instructed slapproxy that the master software should be installed on our computer, we run slapgrid to actually install the software (in short slapgrid will ask what to do to slapproxy, which will answer to installe the edglet master software)

```
# bin/slapgrid -c /opt/slapos/slapos.cfg
```

Check the slapproxy log and wait until they show that the software release is ready.

10.2.9 Step 9: request an instance of the edgelet master

From the slapconsole:

```
$ bin/slapconsole
import json
true = True
parameter_json = {
  "site-id": "erp5",
  "erp5-ca": {
    "country-code": "FR",
    "city": "Lille",
    "state": "Nord-Pas-de-Calais",
    "company": "ViFiB SARL",
    "email": "admin@vifib.org"
```

```
  },
  "zeo": {
    "Zeo-Server-1": [{
      "zodb-name": "main",
      "serialize-path": "/%(site-id)s/account_module/",
      "mount-point": "/",
      "zope-cache-size": "2000",
      "storage-name": "main",
      "zeo-cache-size": "400MB"
    }]
  },
  "activity": {
    "zopecount": 1,
    "timerservice": true
  },
  "timezone": "Europe/Paris",
  "backend": {
    "shacacheupload": {
      "zopecount": 1,
      "no-timeout": true,
      "maxconn": 1,
      "backend-path": "/%(site-id)s/web_site_module",
      "access-control-string": "all",
      "scheme": ["https"],
      "thread-amount": 1,
      "ssl-authentication": true
    },
    "login": {
      "zopecount": 1,
      "access-control-string": "all",
      "scheme": ["https"],
      "maxconn": 1,
      "thread-amount": 1
    },
    "erp5": {
      "zopecount": 1,
      "access-control-string": "all",
      "scheme": ["https"],
```

```
      "maxconn": 1,
      "thread-amount": 1
    },
    "shacachedownload": {
      "zopecount": 1,
      "no-timeout": true,
      "http-cache": true,
      "backend-path": "/%(site-id)s/web_site_module",
      "access-control-string": "all",
      "thread-amount": 1,
      "scheme": ["http"],
      "maxconn": 1
    },
    "service": {
      "zopecount": 2,
      "no-timeout": true,
      "http-cache": true,
      "backend-path": "/%(site-id)s/portal_slap",
      "access-control-string": "all",
      "thread-amount": 1,
      "scheme": ["https"],
      "maxconn": 1,
      "ssl-authentication": true
    }
  }
}
partition_parameter_kw = {
  'json': json.dumps(parameter_json),
}

# Choose a title
title = "My master Instance"
target_node = 'myedgeletmaster'

import slapos.slap.slap
slap = slapos.slap.slap()
slap.initializeConnection('http://127.0.0.1:5000/')
```

```
mypartition = slap.registerOpenOrder().request(
  'https://raw.githubusercontent.com/fiware-
edgelets/slapos/master/software/erp5/software.cfg',
  title,
  partition_parameter_kw=partition_parameter_kw,
  filter_kw=dict(computer_guid=target_node),
  software_type='production',
)
# Please note somewhere the following partition id, you will
use it later to
# fetch informations
master_computer_partition.getId()
```

10.2.10 Step 10: instantiate the edgelet master

Run slapgrid to request the intantiation of the edgelet master

```
# bin/slapgrid-cp -c /opt/slapos/slapos.cfg
```

Again check the slapproxy logs to check that everything is fine and wait until it is finished. You should see messages like

10.2.11 Step 11: use the services

Once ready, retrieve some endpoints for accessing the software running the edgelet master (you will get errors if it is not ready, then retry a bit later)

```
$ bin/slapconsole
import slapos.slap.slap
slap = slapos.slap.slap()
slap.initializeConnection('http://127.0.0.1:5000/')

# The second parameter of the following function is the result
of
# erp5_instance.getId() in step 9.
# It is usually 'slappart0' with slapproxy.
partition = slap.registerComputerPartition('myedgeletmaster',
'slappart0')
partition.getConnectionParameterDict()
```

Run the last command on slappart0, slappart1 etc until you find one with url-service and url-login: these are the parameters needed for the next step

Restart your node

```
# slapos node restart all
```

10.2.12 Step 12: Edgelet master configuration

Connect to the address corresponding to url-service/erp5. You might need some time before it is available.

You can then login with user "zope" and password "insecure"

On the menu "my favourites", select "Check site consistency". Then select all items in the list, and chose "Fix site configuration".

It can take few minutes, but some modules should be showed on front page. Otherwise you can repeat the process by going to "Check Site Consistency"

Check /portal_activities/manageActivities and wait until there is no activity left. You can periodically reload the page to see the status.

When done, restart your master instance

```
# slapos node restart all
```

Solve again consistency if any consistency error (except cloudooo) is still present and restart again the services.

As soon "Configure Your Site" is showed on menu (and make sure you don't have activities running) use the "My favourites" Menu to select "Configure your Site".

Select the entry available and start the configuration. You can see a progress bar showing the configuration process advancement. This can take a while.

When done, restart your master instance

```
# slapos node restart all
```

You can now safely stop the slapproxy instance.

10.2.13 API and portal installation and launch

On your machine go to user home and create a new directory

```
$ mkdir edgelet-portal
```

Then download the portal source code from FI-WARE download page https://forge.fi-ware.org/frs/?group_id=7 and run the installation script

```
$ install_portal.sh
```

The script will ask for ome information (like the url used before). Answer accordingly.

In order to launch the portal, use the launch_portal.sh script

10.3 Sanity check procedures

- The edgelet master installation procedure is quite long and as it is downloading lot of dependencies, you should be patient.
- You can control the correct output of each step of the installation procedure
- For the first steps, looking at the log output from slapproxy will give you information on the status of the software and instance installation, since the local node will periodically send information on its status
- Once you can navigate to url-service/erp5 you are sure that the master has been installed correctly
- One progress bar is indicating the status of the master configuration
- You can navigate to the portal and/or login to the API to check if it is running correctly with the zope login (either from localhost or using the proper ip adress)

```
http://localhost:8080/
```

10.3.1 End to End testing

Accessing to the portal or the API without specific errors shows that the master is configured correctly. For example, require the registration of a new node via the web portal in order tc check that the master is in ready state and that the portal is able to communicate with it.

```
http://<service ip>:<service_port>/
```

10.3.2 List of Running Processes

- The API and portal are using the nodejs server
- Edgelet master is based on zope

10.3.3 Network interfaces Up & Open

Following ports should be open and in use (using TCP):

```
* Nodejs port for the portal (default to 8080)
* The edgelet master runs on the port given in the url-service parameter in step 11
```

10.3.4 Databases

For this release, the portal and catalogue are using a SQLite database (File data.db). You can test a query like "SELECT COUNT(*) FROM tokens" to test if it is correctly launched.

The slapmaster itself is using mariadb in a specific container, and shouldn't be accessed from the outside

10.4 Diagnosis Procedures

10.4.1 Resource availability

The resource availability should be at least 1Gb of RAM and 1GB of Hard disk in order to prevent enabler's bad performance.

10.4.2 Remote Service Access

You should be able to connect to the portal or the API

10.4.3 Resource consumption

The edgelet master is quite heavy and you will need to meet the following requirements:

- 30GB free on the HDD
- 4GB memory

10.4.4 I/O flows

The applictaion is running on port 5000 and 8080 if you are using the default ports. Please refer to the installation steps in order to know exactly which ports are used