Snooze Documentation

Release 1.1.0

Matthieu Simonin

July 16, 2013

CONTENTS

1 Contents: **1**
1.1 Administration Guide 1
1.2 Developper Guide 20

2 Support **35**

CHAPTER ONE

CONTENTS:

1.1 Administration Guide

1.1.1 Installation of the Command Line Interface

This page describes how to install, configure, and use the Snooze command-line interface (CLI). Note that for the time being the CLI installation tutorial is for Debian users only. Still as we also provide the CLI binary, with little efforts you should be able to run it on any distribution after following this tutorial.

Install the client

Get the latest client Debian package from the Downloads page and install it:

```
($) dpkg -i snoozeclient_X.X-X_all.deb
```

Configure the client

Open the client configuration package file */usr/share/snoozeclient/configs/snooze_client.cfg*. It is split into two parts: general and statistics

```
general.bootstrapNodes = localhost:5000,192.168.0.2:5001
general.submissionPollingInterval = 6
general.numberOfMonitoringEntries = 5
general.dumpOutputFile = /tmp/snooze_tree.xml
general.graphPollingInterval = 5

statistics.enabled = false
statistics.output.format = gnuplot
statistics.output.file = /tmp/snooze_results.dat
```

- *General*

The client application requires at least one active bootstrap node in order to discover the current group leader (GL). Consequently it will contact one of the active servers specified in the *general.bootstrapNodes* list to do so.

The VM submission requests are processed asynchronously in Snooze. The client application submits the requests to the GL and periodically polls it for a reply. The polling interval can be specified using the *general.submissionPollingInterval parameter*.

The client application offers features to retrieve VM monitoring information as well as to visualize the system hierarchy. Therefore, it needs to send requests to the group manages (GMs). The parameter *general.numberOfMonitoringEntries* specifies the number of monitoring entries client requests from the GMs.

Finally the client application offers a feature to dump the hierarchy layout in the GraphML format to the disk. Therefore, the *general.dumpOutputFile* parameter specifies the output file name.

- *Statistics*

Once enabled (*statistics.enabled*) the client application will write statistics (e.g. submission time, number of failed submissions) to the disk. You can specify the statistics output format and the file name using the two parameters: *statistics.output.format*, *statistics.output.file*. Note that currently only GNUPlot output format is implemented.

Control VM life-cycle

On the shell execute the *"snoozeclient"* command. The following commands are currently supported:

```
Commands:
    define      Define virtual cluster
    undefine    Undefine virtual cluster
    add         Add virtual machine to the cluster
    remove      Remove virtual machine from the cluster
    start       Start virtual cluster/machine
    suspend     Suspend virtual cluster/machine
    resume      Resume virtual cluster/machine
    shutdown    Shutdown virtual cluster/machine
    destroy     Destroy virtual cluster/machine
    info        Shows virtual cluster/machine information
    list        List virtual clusters
    visualize   Visualize system hierarchy
    dump        Dump system hierarchy
```

You can display a help for each command by executing "snoozeclient [command] –help".

We use the notion of virtual cluster (VC) to group together virtual machines (VMs) prior submission on the client side. Consequently, in order to submit VMs you first need to define at least one VC and add some VMs to it. To define a VC execute the following command:

```
($) snoozeclient define -vcn myVC
```

Note that you can always undefine/remove the VC using the *undefine* command.

Contextualize the virtual machines

In order to add VMs to your newly created VC, first disk images and VM templates must be created. Thereby, disk images must be placed on a storage accessible to all the local controllers (LCs), otherwise they can not be located by the LCs during submission. For example, if you are using NFS and the QEMU Copy-on-Write (COW) mechanism you need to go through the following steps: (1) Copy backing file VM image to the mounted storage directory (e.g. /opt/cloud); (2) Create a COW file from the backing file image for each VM; (3) Create and modify the VM template to point to the COW image.

Please follow the How you can use qemu/kvm base images to be more productive tutorial now to create and install your backing file image (also known as base image).

Now that you have your base image installed, you must configure its networking in order to be reachable to the outside world after submission. Snooze automatically assigns IP addresses to VMs from the system administrator defined VM subnet (see networking settings in the Admin manual) during submission. This is currently done by encoding the assigned IP address into the VMs MAC address. When a VM boots it needs to decode this IP from its MAC address and configure the networking. We use init scripts to accomplish these tasks which will provide different level of contextualization. Choose one of the methods below depending on your needs.

- *Static Contextualization*

If your network parameters (gateway, netmask, nameserver . . .) are static, read *Static Contextualization*

- *Dynamic Contextualization*

If you need to have more flexibility on your network parameter, read *Dynamic Contextualization* . With this method you will be able to change the network parameters of your virtual machine more easily.

Start the Virtual Machines

The VM can now be added to the VC. Therefore you need to specify the VC name and pass the VM template describing your VM environment. In addition networking capacity constraints can be specified. For example, Snooze can be instructed to cap the VMs network capacity requirements to 10MBit/sec. The following command adds a VM those network capacity is bounded to 10MBit. Note that in case of no network capacity restrictions are given the default value is 100MBit.

```
($) snoozeclient add -vcn myVC -vmt /home/user/vmtemplates/debian1.xml -rx 12800 -tx 12800
```

Note that you can always remove a VM by simply calling: *snoozeclient remove -vcn myVC -vmn myVM*

You can now either add more VMs and start all of them at once or trigger individual VMs submissions by executing one of the following commands:

```
($) snoozeclient start -vcn myVC (starts all VMs belonging to myVC)
or
($) snoozeclient start -vcn myVC -vmn myVM (starts myVM)
```

If everything went well you should see a similar output.

```
Name        VM address       GM address    LC address    Status
---------------------------------------------------------------
debian1     192.168.122.5     10.0.0.2     10.0.0.2      RUNNING
```

Please see the FAQ for possible problem resolutions (e.g. ERRORs in submission). Otherwise use the user mailing list or IRC channel to ask questions.

Finally, the client offers a variety of commands to control the VM execution. For example, it is possible to suspend, resume, shutdown, or destroy VCs/VMs by simply calling:

```
($) snoozeclient suspend/resume/shutdown or destroy -vcn myVC (all VMs belonging to myVC)
($) snoozeclient suspend/resume/shutdown or destroy -vcn myVC -vmn myVM (only myVM)
```

Last but not least, VM resource (i.e. CPU, memory, network Rx, network Tx) usage information can be displayed using the info command either for the entire VC or a single VM. You should see a similar output:

```
Name       CPU      Memory     Rx/Tx         VM            GM         LC         Status
           usage    usage      usage         address       address    address
-------------------------------------------------------------------------------------
debian1    0.09     596992     0.12/0.1   192.168.122.5   10.0.0.2   10.0.0.2   RUNNING
```

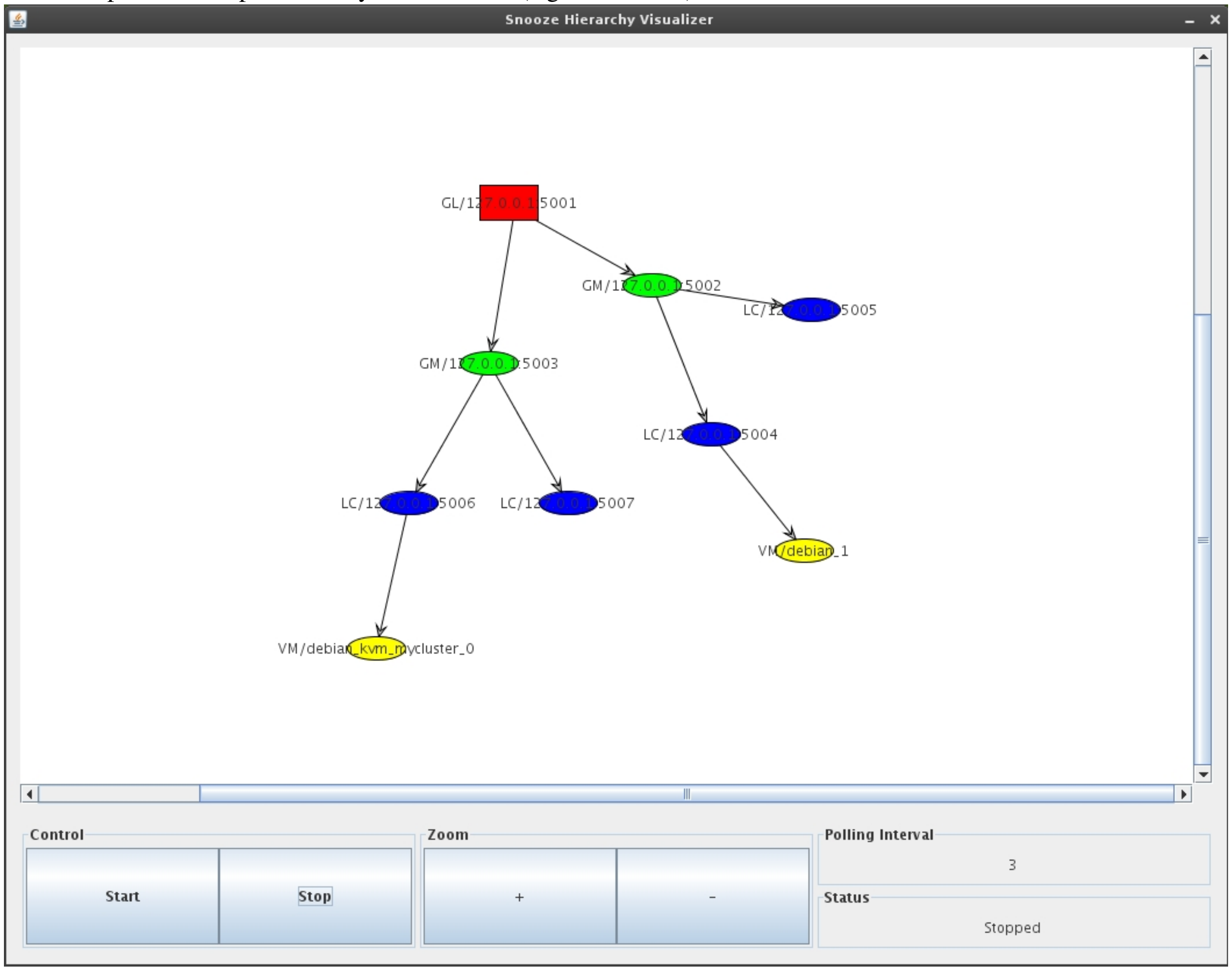
Note that it takes some time to propagate the initial VM monitoring information data. During this time the client will display the “UNKNOWN” message in the usage fields.

System hierarchy visualization and dump

You can use the client to visualize the current hierarchy state (GL, GMs, LCs, VMs) or dump it in GraphML format to the disk. Visualization requires either a running XServer or X11 forwarding. You can activate visualization by using the appropriate client command:

```
($) snoozeclient visualize
```

If everything works out a GUI will appear in which you will be able to specify the polling interval and start/stop the visualization process. Moreover, you will be able to zoom in and out the hierarchy state. The polling interval specifies the time period in which the client will request system repository information and redraw the hierarchy. This allows to visualize the system self-organization and healing as well as VM live migrations (e.g. during relocation and consolidation). The following figure shows an example system visualization with one GL, 2 GMs, 4 LCs, and 2 VMs. Each components is represented by its own color (e.g. red for GL).



1.1.2 Build new VM images

Depending on your need, you can choose between two contextualization methods.

Static Contextualization

Create the base image

Get the init script from our Download page. It was tested on the Debian as well as Ubuntu operating system. Please move it to your /etc/init.d/ directory and run the following command to update the init script links.

```
($) update-rc.d vmcontext defaults
```

Your base image should be now ready!

Generate the template

Assumed that the name of your backing file is debian-base.raw, you will need to execute the following command to create a COW image debian1.qcow2

```
($) qemu-img create -b debian-base.raw -f qcow2  debian1.qcow2
```

You can now prepare the VM template. Currently, the client accepts native libvirt templates. It is important that you choose different names and UUIDs (by the way UUID can be omitted) for your VMs . Otherwise, subsequent submissions will fail due to naming conflicts. Moreover, disk image path and bridge name must be set correctly. An example libvirt VM template (debian1.xml) for the KVM hypervisor could look as follows:

```
<domain type='kvm'>
  <name>debian1</name>
  <uuid>0f476e56-67ea-11e1-858e-00216a972a36</uuid>
  <memory>597152</memory>
  <currentMemory>597152</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-0.12'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/opt/cloud/debian1.qcow2'/>
      <target dev='vda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </disk>
    <controller type='ide' index='0'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
    </controller>
    <interface type='bridge'>
      <mac address='52:54:00:83:25:2b'/>
      <source bridge='virbr0'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <graphics type='vnc' port='-1' autoport='yes' listen='0.0.0.0'/>
    <input type='tablet' bus='usb'/>
    <input type='mouse' bus='ps2'/>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

Note that the bridge name virbr0 must correspond to the bridge configured on the LCs!

Dynamic Contextualization

Principles

Dynamic contextualization mechanism works as the following :

- Test for the presence of a CD in the CD drive of the VM,
- if it exists, mount the CD, test the presence of a script post-install, and run it as root,
- if it does not exist, use dhcp on the first network interface.

Contextualization

Copy the following script in your virtual machine under /usr/local/bin/contextualization.

```
#!/bin/bash

DEVICE=
[ -b /dev/hdb ] && DEVICE=/dev/hdb
[ -b /dev/sdb ] && DEVICE=/dev/sdb
[ -b /dev/vdb ] && DEVICE=/dev/vdb
[ -b /dev/xvdb ] && DEVICE=/dev/xvdb
[ -b /dev/sr0 ] && DEVICE=/dev/sr0

if [ -b "$DEVICE" ];then
/bin/mount -t iso9660 $DEVICE /mnt 2> /dev/null

if [ -f /mnt/post-install ]; then
bash /mnt/post-install
fi

umount /mnt 2> /dev/null
else
ifup eth0
fi

exit 0
```

Open /etc/rc.local and call this script :

```
/usr/local/bin/contextualization
```

Your base image should be now ready!

Generate the iso context file

You can download the context files from the Download page. Fill the files common/network and common/routes with your parameters. This parameters will be applied at boot time on the virtual machines. Generate the context.iso file :

```
($) genisoimage -RJ -o context.iso context
```

Generate the virtual machine template

Assumed that the name of your backing file is debian-base.raw, you will need to execute the following command to create a COW image debian1.qcow2.

```
($) qemu-img create -b debian-base.raw -f qcow2  debian1.qcow2
```

You can now prepare the VM template. Currently, the client accepts native libvirt templates. It is important that you choose different names and UUIDs (by the way UUID can be omitted) for your VMs. Otherwise, subsequent submissions will fail due to naming conflicts. Moreover, disk image path and bridge name must be set correctly. An example libvirt VM template (debian1.xml) for the KVM hypervisor could look as follows:

```
<domain type='kvm'>
<name>debian1</name>
<memory>524288</memory>
<vcpu>1</vcpu>
<os>
   <type arch="x86_64">hvm</type>
</os>
<clock sync="localtime"/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<devices>
   <emulator>/usr/bin/kvm</emulator>
   <disk type='file' device='disk'>
   <driver name='qemu' type='qcow2'/>
      <source file='/opt/cloud/debian1.qcow2'/>
      <target dev='vda' bus='virtio'/>
   </disk>
   <disk type='file' device='cdrom'>
     <source file='/opt/cloud/context.iso'/>
     <target dev='vdb' bus='virtio'/>
     <readonly/>
   </disk>
   <interface type='bridge'>
     <mac address='00:16:c0:A8:7a:11'/>
     <source bridge='virbr0'/>
   </interface>
   <serial type='pty'>
     <source path='/dev/ttyS0'/>
     <target port='0'/>
   </serial>
   <console type='pty'>
     <source path='/dev/ttyS0'/>
     <target port='0'/>
   </console>
</devices>
</domain>
```

Note that the bridge name virbr0 must correspond to the bridge configured on the LCs! Note that the context.iso file must be accessible from your LCs.

1.1.3 Deployment

Single Machine Setup

Local deployment involves running all the Snooze system components: Bootstrap Node (BN), Group Leader (GL), Group Managers (GMs), and Local Controller (LCs) as well as dependencies: Apache ZooKeeper and libvirtd on the same machine (e.g. your laptop).

Hardware and OS requirement

- Networking connection for multicast
- Hardware supporting the KVM hypervisor
- Debian wheezy or Ubuntu (12.04 or newer)

Prepare the environment

Install the Java runtime environment, Apache ZooKeeper, and QEMU/KVM.

```
$ apt-get install openjdk-6-jre zookeeper zookeeperd qemu-kvm libvirt-bin libvirt-dev
$ groupadd snooze
$ useradd -s /bin/false snoozeadmin -g snooze
```

Add User *snoozeadmin* the libvirt group.

```
(Debian)$ adduser snoozeadmin libvirt
(Ubuntu)$ adduser snoozeadmin libvirt
```

Install Snooze

Get the latest Snooze client and node Debian packages from the Downloads page and install them.

```
dpkg -i snoozeclient_X.X-X_all.deb
dpkg -i snoozenode_X.X-X_all.deb
```

Use the local deployment script

The local deployment script is made to facilitative the start of a local cluster. Particularly, it automatically creates all the configuration files needed to run multiple Snooze and libvirt daemons on a single machine. Moreover it launches the daemons and the required dependencies (e.g. the Apache ZooKeeper service).

You can get the local deployment script from the Downloads page (or from the Github repository). Unpack the archive and configure your Snooze deployment scenario. In other words, you need to inform the script how many BNs, GMs, and LCs you would like to run on your host. Therefore open the ./scripts/settings.sh file in localcluster subdirectory and look for the following lines.

```
number_of_bootstrap_nodes=1
number_of_group_managers=3
number_of_local_controllers=2
```

Adopt them to your needs and you are almost done. You are now ready to instruct the script to launch the libvirtd and Snooze daemons. Note that in case you already have a libvirt and snoozenode daemon running (default installation), they needs to be stopped to avoid possible port collisions.

```
$ /etc/init.d/libvirt-bin stop
$ /etc/init.d/snoozenode stop
```

To list the available local deployment script options start the following command

```
$ ./start_local_cluster.sh

Options:
-l Start libvirt
-d Stop libvirt
-s Start cluster
-k Kill cluster
```

Run the script with "-l" and "-s" options to start the libvirt and Snooze daemons

```
$ ./start_local_cluster.sh -l
$ ./start_local_cluster.sh -s
```

You now should have your local Snooze cluster up and running. The debug outputs of the Snooze components are stored in the /tmp/ directory under the file names: snooze_node_bn.log, snooze_node_gm1.log, etc. In case you need to stop the cluster use the "-d" and "-k" options.

Cluster

This page describes how to deploy Snooze on a traditional cluster with one management node hosting all the core system services (i.e. *NFS, Apache ZooKeeper, Bootstrap node, Group leader, and one Group manager*) and the rest of the nodes serving as compute resources (i.e. *local controllers*).

Note that our preferred Linux distribution is Debian. Consequently, the documentation below requires a working Debian installation. However, with some efforts you should be able to install Snooze on any Linux distribution by using the provided Binary (see Downloads section).

If you have managed to install Snooze on your favorite distribution please do not hesitate to contact us with your documentation. We will review and publish it.

Management Node

Management node require :

- *Multicast-routing enabled switch*
- *Hardware supporting the KVM/XEN hypervisor*
- *Debian Squeeze (Sid pinned) or wheezy operating system*

Prepare the environment

```
$ apt-get install nfs-server ipmitool sudo sun-java6-jre sun-java6-bin zookeeper zookeeperd
$ mkdir -p /opt/cloud
$ echo "/opt/cloud *(rw,sync,no_root_squash,no_subtree_check)" >> /etc/exports
$ /etc/init.d/nfs-kernel-server restart
$ groupadd snooze
$ useradd -d /opt/cloud -s /bin/false snoozeadmin -g snooze
$ "visudo" and add: "snoozeadmin ALL = NOPASSWD: /usr/bin/ipmitool" to the user privilege section
```

Install and configure Snooze Download and install the latest snoozenode package. You can find the latest snoozenode package on the Downloads page.

:: $ dpkg -i snoozenode_X.X-X_all.deb

Replicate Snooze and logger configuration files for BN, GL, and GM:

```
$ cp /usr/share/snoozenode/configs/snooze_node.cfg /usr/share/snoozenode/configs/snooze_node_bn.cfg
$ cp /usr/share/snoozenode/configs/snooze_node.cfg /usr/share/snoozenode/configs/snooze_node_gm_1.cfg
$ cp /usr/share/snoozenode/configs/snooze_node.cfg /usr/share/snoozenode/configs/snooze_node_gm_2.cfg

$ cp /usr/share/snoozenode/configs/log4j.xml /usr/share/snoozenode/configs/log4j_bn.xml
$ cp /usr/share/snoozenode/configs/log4j.xml /usr/share/snoozenode/configs/log4j_gm_1.xml
$ cp /usr/share/snoozenode/configs/log4j.xml /usr/share/snoozenode/configs/log4j_gm_2.xml

$ cp /etc/init.d/snoozenode /etc/init.d/snoozenode_bn
$ cp /etc/init.d/snoozenode /etc/init.d/snoozenode_gm_1
$ cp /etc/init.d/snoozenode /etc/init.d/snoozenode_gm_2
```

Modify the snoozenode configuration files: snooze_node_bn.cfg, snoozenode_gm_1.cfg,snoozenode_gm_2.cfg. In the bootstrap node configuration file set:

```
node.role = bootstrap
```

In the first group manager configuration file set:

```
node.role = groupmanager
network.listen.controlDataPort = 5001
network.listen.monitoringDataPort = 6001
network.virtualMachineSubnet = XXX.XXX.XXX.0/24 (your VM subnet)
```

In the second group manager configuration file set:

```
node.role = groupmanager
network.listen.controlDataPort = 5002
network.listen.monitoringDataPort = 6002
network.virtualMachineSubnet = XXX.XXX.XXX.0/24 (your VM subnet)
```

Modify the logger configuration files: log4j_bn.xml, log4j_gm_1.xml, and log4j_gm_2.xml

change the output file name "snooze_node.log" to "snooze_node_bn.log", "snoozenode_gm_1.log", and "snoozenode_gm_2.log" respectively

Modify the snoozenode init scripts: snoozenode_bn, snoozenode_gm_1, and snoozenode_gm_2 Change the following lines in the same manner as the logger configuration files:

```
# Provides: snoozenode -> Change to snoozenode_bn/gm_1/gm_2
SCRIPT_NAME="snoozenode" -> Change to snoozenode_bn/gm_1/gm_2
PID_FILE="/var/run/snoozenode.pid" ->  Change to snoozenode_bn/gm_1/gm_2.pid
SYSTEM_CONFIGURATION_FILE="$CONFIGS_DIRECTORY/snooze_node.cfg" -> Change to snooze_node_bn/gm_1/gm_2
LOGGER_CONFIGURATION_FILE="$CONFIGS_DIRECTORY/log4j.xml" -> Change to log4j_bn/gm_1/gm_2.xml
```

Update the init script links:

```
$ update-rc.d -f snoozenode remove
$ update-rc.d snoozenode_bn defaults
$ update-rc.d snoozenode_gm_1 defaults
$ update-rc.d snoozenode_gm_2 defaults

$ /etc/init.d/snoozenode_bn start
$ /etc/init.d/snoozenode_gm_1 start
$ /etc/init.d/snoozenode_gm_2 start
```

You now should have one bootstrap node, group leader, and group manager running. Please refer to logs (/tmp/snooze_node_*.log) to see if all components are running. For example bootstrap node should be receiving group leader heartbeats.

On all compute nodes

Prepare the environment Install the dependencies

```
$ apt-get install kvm qemu-kvm libvirt-bin libvirt-dev bridge-utils nfs-common sudo sun-java6-jre sur
$ apt-get install ipmitool pm-utils
```

Modify the /etc/hosts file

```
127.0.0.1  localhost.localdomain localhost
XXX.XXX.XXX.XXX  yourHostName
```

Make sure your set the right IP and hostname of your main network interface. Enable bride networking /etc/network/interfaces

If your main network interface is *eth0* modify as follows:

```
auto lo
iface lo inet loopback

auto br0
iface br0 inet dhcp
bridge_ports eth0
bridge_stp on
bridge_maxwait 0
```

Restart networking:

```
$ /etc/init.d/networking restart
```

Please see http://wiki.debian.org/BridgeNetworkConnections for more details on bridge network connections. Ensure NFS is remounted in case the node gets rebooted: modify /etc/rc.local and add

```
mount -a
```

Add snooze group and snoozeadmin user to the system

```
$ groupadd snooze
$ useradd -d /opt/cloud -s /bin/false snoozeadmin -g snooze -G kvm
```

Configure the libvirt daemon by editing the following files libvirtd.conf, libvirt-bin, and qemu.conf

In /etc/libvirt/libvirtd.conf: disable TLS and enable TCP listing, disable TCP authentication:

```
listen_tls = 0
listen_tcp = 1
auth_tcp = "none"
```

In /etc/default/libvirt-bin

```
libvirtd_opts="-d -l"
```

In /etc/libvirt/qemu.conf enter the snooze user and group information:

```
user = "snoozeadmin"
group = "snooze"
```

Restart the libvirt daemon:

```
$ /etc/init.d/libvirt-bin restart
```

Configure the NFS server

```
$ mkdir /opt/cloud
$ echo "<MANAGEMENT_NODE_IP>:/opt/cloud /opt/cloud nfs rw 0 0" >> /etc/fstab
$ mount -a
```

If you need power management features "visudo" and add to the user privilege section:

```
snoozeadmin ALL = NOPASSWD: /sbin/shutdown
snoozeadmin ALL = NOPASSWD: /usr/bin/ipmitool
snoozeadmin ALL = NOPASSWD: /usr/sbin/pm-suspend
snoozeadmin ALL = NOPASSWD: /usr/sbin/pm-hibernate
snoozeadmin ALL = NOPASSWD: /usr/sbin/pm-suspend-hybrid
snoozeadmin ALL = NOPASSWD: /usr/sbin/s2ram
snoozeadmin ALL = NOPASSWD: /usr/sbin/s2disk
snoozeadmin ALL = NOPASSWD: /usr/sbin/s2both
```

Install and configure Snooze Download and install the latest snoozenode package.You can find the latest snoozenode package on the Downloads page.

```
$ dpkg -i snoozenode_X.X-X_all.deb
```

Configure the snoozenode daemon. Open the configuration file: /usr/share/snoozenode/configs/snooze_node.cfg

Set the node.role

```
node.role = localcontroller
```

If you require power management, specify how this LC should be woken up. In this example the node is woken up using IPMI. Note that depending on your hardware different options must be passed to the IPMI driver. Currently, IPMI driver internally calls the ipmitool to enforce the commands. Note that you must verify that IPMI wake up works prior entering any information here, otherwise wake ups will fail.

```
# Wakeup driver (IPMI, WOL, kapower3, test)
energyManagement.drivers.wakeup = IPMI

# Wakeup driver specific options
# For IPMI
energyManagement.drivers.wakeup.options = -I lanplus -H <BMC_IP> -U <user> -P <password>
```

Start the snoozenode daemon

```
$ /etc/init.d/snoozenode start
```

Your Snooze cluster should be now up and running. Please see the log file (/tmp/snooze_node.log) to verify that everything works as it should and install more LCs if needed. You can also use the CLI to visualize the current system state. Please refer to the User manual for more information.

Note that Snooze deployment is not limited to the scenario described above. For example you could imagine installing all the system components (Apache ZooKeeper, BN, and more GMs) on dedicated servers to enhance scalability and fault tolerance. On the other hand you could also have a configuration in which servers act as management as well as compute nodes (i.e. having GM and LC on the same host). You should be able to realize such configurations with little effort after following this tutorial.

Multi-Nodes Cluster on Grid‘5000

Installation and Usage

- Make a reservation

```
(frontend)$ oargridsub -t deploy -w 1:00:00 rennes:rdef="{\\\\\\\"type='kavlan-global'\\\\\\\"}/vlan=
```

NB1 : On a single site you don't need to reserve a vlan. The reservation could be :

```
(frontend)$ oarsub -I -t deploy -l slash_22=1,nodes=10,walltime=8
```

NB2 : With a multisite reservation, the file oargrid.out is used by the script so it must be placed in your home directory.

If everything is fine this file looks like :

```
rennes:rdef={\\\"type='kavlan-global'\\\"}/vlan=1+/slash_22=1,lyon:rdef=/nodes=3,sophia:rdef=/nodes=
[OAR_GRIDSUB] [rennes] Date/TZ adjustment: 0 seconds
[OAR_GRIDSUB] [rennes] Reservation success on rennes : batchId = 471354
[OAR_GRIDSUB] [lyon] Date/TZ adjustment: 0 seconds
[OAR_GRIDSUB] [lyon] Reservation success on lyon : batchId = 599979
[OAR_GRIDSUB] [sophia] Date/TZ adjustment: 0 seconds
[OAR_GRIDSUB] [sophia] Reservation success on sophia : batchId = 530702
[OAR_GRIDSUB] Grid reservation id = 42581
[OAR_GRIDSUB] SSH KEY : /tmp/oargrid//oargrid_ssh_key_msimonin_42581
You can use this key to connect directly to your OAR nodes with the oar user.
```

- Connect to your job (with the kavlan reservation):

```
(frontend)$ oarsub -C 471354
```

- Configure the proxy :

```
(frontend)$ export https_proxy="http://proxy:3128"
```

- Clone the git repository :

```
(frontend)$ git clone https://github.com/snoozesoftware/snooze-deploy-grid5000.git
(frontend)$ git checkout 1.1.0
```

- Download latest version of debian package (snoozenode is require, snoozeclient is optional) :

```
(frontend)$ cd ~/snooze-deploy-grid5000/deb_packages/
(frontend)$ wget https://ci.inria.fr/snooze-software/job/maint-1.1.0-snoozenode/ws/distributions/deb-
(frontend)$ wget https://ci.inria.fr/snooze-software/job/maint-1.1.0-snoozeclient/ws/distributions/de
```

Other packages could be found in https://ci.inria.fr/snooze-software/.

Configure and launch the deployment

- Configure the number of nodes in the **settings.sh** and the deployment type :

```
(frontend)$ cd ~/snooze-grid5000-multisite/deployscript/
(frontend)$ vi scripts/settings.sh
```

```
multisite=true|false
storage_type="nfs|local"
number_of_bootstrap_nodes=1
number_of_group_managers=2
number_of_local_controllers=5
```

NB : Since we use a "service node" for the deployment you will get a Snooze cluster running with n-1 nodes. NB2 : If you set storage type to "local", the VMs base images will be propagated by scp-tsunami, you have to install it in /opt of the first bootstrap (see settings.sh of experiments scripts).

- Retrieve VMs base images in **~/vmimages/**

You can get my debian base image in /home/msimonin/vmimages in Rennes (see Snooze documentation to know how to generate a new image)

```
(frontend)$ scp -r <yourlogin>@rennes:/home/msimonin/vmimages ~/.
```

- Launch the automatic script :

```
(frontend)$ ./snooze_deploy.sh -a
```

Use the system

- Connection to the first bootstrap :

```
(frontend)$ cat tmp/bootstrap_nodes.txt
(frontend)$ ssh -l root <first bootstrap>
```

- Launching VMs :

The first bootstrap node hosts some helper to launch VMs.

```
(bootstrap)$ cd /tmp/snooze/experiments
(bootstrap)$ ./experiments -c test 5
(bootstrap)$ snoozeclient start -vcn test
```

These commands will create and start 5 VMs.

- Visualizing the system :

Make a tunnel (or export your display) from your laptop to the bootstrap through the grid'5000 frontend on port 5000. If snoozeclient is installed on your PC, you can launch :

(PC) snoozeclient visualize.

1.1.4 Snooze Config File

Snooze system configuration is done using its main configuration file /usr/share/snoozenode/configs/snooze_node.cfg. You should be able to find it on each node after installation. It is partitioned into several categories. you can find the description of each categorie below.

Node

- *node.role*

Each node can play the role of either a bootstrap, groupmanager, or localcontroller. It is up to the system administrator to decide on the role and the number of nodes dedicated to a role during the system setup.

- *node.networkCapacity.Rx* and *node.networkCapacity.Tx*

  Available networking capacity (Rx and Tx) must be configured for each local controller (LC). It is set to 1 Gb (= 131072 kB) by default.

Networking

- *network.listen.address*

Binds the Snooze service to the specified address. By default it listen on all interfaces.

- *network.listen.controlDataPort*

Control data port is required for the REST communication on each node.

- *network.listen.monitoringDataPort*

Monitoring data port is used by the GMs and the GL to receive monitoring data from LCs (resp. GMs).

- *network.multicast.address*

Multicast address is used to periodically propagate heartbeat information.

- *network.multicast.groupManagerHeartbeatPort*

The multicast port on which a GM announces its presence (must be distinct on each GM).

- *network.virtualMachineSubnet*

Coma separated list of virtual machine subnets from which Snooze will assign IPs to VMs.

Httpd

- *httpd.maxNumberOfThreads*

Adjust the maximum number of threads in the pool to handle incoming RESET requests.

- *httpd.maxNumberOfConnections*

The maximum number of allowed connections.

- *httpd.minThreads*

Minimum number of active threads.

- *httpd.lowThreads*

Low number of active threads.

- *httpd.maxThreads*

Maximum number of active threads.

- *httpd.maxQueued*

Max Idle Time (0 for infinite)

Hypervisor

- *hypervisor.driver*

The name of the underlying hypervisor driver. It is set to qemu by default.

- *hypervisor.transport*

The transfer method between the hypervisors (i.e. libvirt daemons). It is set to tcp by default. Note that the system has only been tested using the tcp transport for now! You will need to configure libvirtd accordingly to use other transport methods.

- *hypervisor.port*

The local libvirtd listen port. LCs will use it to connect to contact libvirt.

- *hypervisor.migration.method*

You can choose different migration methods as supported by libvirt (see virDomainMigrateFlags for more details). Currently, Snooze supports live migration, migration with non-shared storage with full disk copy, and migration with non-shared storage with incremental copy. The latter is the default setting.

- *hypervisor.migration.timeout*

In some hypervisors (e.g. KVM) migration can last forever if the number of pages that got dirty is larger than the number of pages that got transferred to the destination LC during the last transfer period. Snooze implements a watchdog mechanisms to force convergence in such situations. Therefore, a timeout must be specified above which the system will suspend the VMs and thus let them finish migrating (i.e. no dirty pages anymore).

Fault Tolerance

- *faultTolerance.zookeeper.hosts*

You must give a list with at least one ZooKeeper host which will be used to coordinate the group leader (GL) election.

- *faultTolerance.zookeeper.sessionTimeout*

The amount of time above with the ZooKeeper session is considered terminated.

- *faultTolerance.heartbeat.interval*

Heartbeat messages are periodically sent from the GL to the GMs and from GMs to the LCs. You can tweak the sending intervals using this parameter.

- *faultTolerance.heartbeat.timeout*

The amount of time above which the heartbeat is considered lost.

Database

System management meta-data as well as monitoring information is stored at the GL as well as at the GMs. The database settings allow you to tweak some of the storage parameters.

- *database.type*

System information can be either stored in-memory or on some persistent storage (e.g. MySQl, MongoDB, etc.). Currently only in-memory storage is supported.

- *database.numberOfEntriesPerGroupManager*

In order to avoid running out of storage capacity you need to specify the maximum number of entries per group manager. For example, if set to 20 the GL will store 20 monitoring entries per GM and start overwriting the oldest ones in case the limit has been reached. In other words, the storage is implemented as a circular buffer.

- *database.numberOfEntriesPerVirtualMachine*

Similarly to the previous setting the GM must be instructed to respect a certain upper bound on the number of monitoring entries per VM.

Monitoring

- *monitoring.interval*

Controls the time interval at which monitoring data is sent from the LCs to GMs and GMs to GL.

- *monitoring.timeout*

The amount of time above which monitoring data is considered as lost. Note that the monitoring timeout is also used to detect GM and LC failures. For example, if GM monitoring data is lost the GL considers it as failed. Similarly, when LC monitoring data is lost it is considered as failed by the GM in charge.

- *monitoring.numberOfMonitoringEntries*

Overload and underload anomaly detection is performed based on aggregates. Particularly, each LC first collects a certain amount of monitoring data entries per VM period starting the anomaly detection. You can control this amount using this parameter.

- *monitoring.thresholds.*

For each resource (i.e. CPU, memory, and network) Snooze defines three thresholds (MIN, MID, and MAX). When the aggregated utilization in one of the resources falls below the MIN threshold the LC is considered underloaded. Similarly, if the utilization crosses the MAX threshold the LC is considered overloaded. The MID threshold is used to cap the max allowed used resource capacity. This allows to keep a buffer of spare resources to compensate during periods of high resource contention. For example if set to 0.5 at max 50% of the available resource capacity will be available to host VMs.

Estimation

- *estimator.static*

True if estimations should be based on static values only. Particularly, if your VM requests 3 GB of RAM during submission but uses on average 2 GB only according to the collected monitoring data, the estimator would still consider the requested capacity when requested to do estimations.

- *estimator.sortNorm*

Sorting VMs requires their resource usage vectors to be mapped to scalar values. Therefore different vector norms (e.g. L1, Euclid, Max) can be used.

- *estimator.numberOfMonitoringEntries*

  The maximum number of monitoring entries per VM to consider in estimations. For example, it is possible to instruct the system to use the most recent 15 monitoring entries per VM in its estimations.

- *estimator.policy.*

You can implement different estimators for each resource and choose between them using this parameter. Currently, estimations are based on averages of estimator.numberOfMonitoringEntries most recent values.

Group Leader Scheduler

- *groupLeaderScheduler.assignmentPolicy*

  When a LC attempts to join the hierarchy it needs to know which GM to join. The assignment policy is in charge of selecting the GM. Currently two assignment policies are implemented: RoundRobin and FirstFit. You can integrate you own assignment policies by implementing the provided assignment interface.

- *groupLeaderScheduler.dispatchingPolicy*

When a client attempts to submit a VC, its VMs need to be dispatched to GMs. The dispatching policy makes the GM choice according to the aggregated GM monitoring data. Currently two dispatching policies are implemented: RoundRobinSingleGroupManager and FirstFitSingleGroupManager.

Note that aggregated information might be not sufficient to take exact dispatching decisions. For instance, when a client submits a VM requesting 2GB of memory and a GM reports 4GB available it does not necessary mean that the VM can be finally placed on this GM as its available memory could be distributed among multiple LCs (e.g. 4 LCs with each 1GB of RAM). Consequently, a list of candidate GMs can be returned by the dispatching policies. Based on this list, the GL performs a linear search by issuing VM placement requests to the GMs.

Existing dispatching policies return a list with a single element (i.e. GM). Consequently when the submission fails on the selected GM, no other will be tried. However, similarly to the assignment policies different dispatching policies can be integrated by implementing the appropriate dispatching interface.

Group manager scheduler

- *groupManagerScheduler.placementPolicy*

The placement policy is used to do initial assignment's of VMs to LCs upon submission. Currently, two placement policies exist RoundRobin and FirstFit

- *groupManagerScheduler.relocation.*

The overload and underload policies are triggered to resolve overload (resp. underload) anomaly situation. Both policies return a migration plan which specifies which VMs and to which LCs they need to be migrated to resolve the anomaly situations.

- *groupManagerScheduler.reconfiguration.enabled*

Complementary to the relocation mechanisms, reconfiguration can be enabled to periodically optimize the VM placement of moderately loaded VMs.

- *groupManagerScheduler.reconfiguration.policy*

You can implement any reconfiguration policy. However, currently Snooze implements a modified version of the Sercon consolidation algorithm. Please refer to Publications for more details.

- *groupManagerScheduler.reconfiguration.interval*

A cron expression which allows to provide a very flexible configuration of the reconfiguration interal (e.g. every night at 1 AM).

Submission

- *submission.dispatching.*

When the client application attempts to submit a VC to the the GL, GL instructs the selected GMs to start VMs. If a GM is busy (e.g. it is in relocation or reconfiguration state), the submission requests are rejected by the GMs state machine. The GL implements retry logic to resent submission requests a predefined number of times within a predefined interval until it considers the VM submission request as failed. You can tune the retry behavior using the numberOfRetries and retryInterval parameters.

- *submission.collection.*

Collection parameters control the VC submission response gathering. Particularly, as soon VMs have been accepted by a GM for submission, the GL will poll the GMs involved in the submission to retrieve the VM submission responses. Note that accepted for submission solely means that it has been added to the to be scheduled queue on the GM. The actual VM placement can take some time depending on the resource availability. For example, if LCs in a deep power saving state (e.g. shutdown) need to be woken up, typically several minutes are required until they become available and can be considered in the VM placement. It is absolutely crucial to set the numberOfRetries and retryInterval parameters carefully and we strongly advise you to keep the default values as they are. Setting both parameters too low will result in the client receiving failed VM submission responses.

Similarly to dispatching two parameters exist to control the response gather behavior: numberOfRetries and retryInterval. The number of retries parameter makes sure that polling terminates in case when the responses never become available due to internal errors while the retry interval specifies the polling period.

- *submission.packingDensity.*

  You can define a packing density for each resource. It will be considered during initial VM placement and allow the VM to be hosted on a LC despite existing MID capping. For example, of a LC has 4 PCORES and the CPU MID threshold (monitoring.thresholds.cpu) is set to 0.5 it is only possible to load it for up to 2 PCORES, keeping two other cores as buffered capacity. If a VM is now submitted which requires 4 VCORES it impossible to place it on the LC. However, with the packing density set to 0.4 it will be considered as a VM requiring only 1.6 VCORES thus allowing it to be placed. Note that packing density < 1 facilitates resource overcommit and thus can lead to serious performance problems. We suggest to keep it at the default value (=1) in case performance is important.

Energy Management

- *energyManagement.enabled*

Enables/disables the energy saving module. Once enabled it will periodically observe the LCs and transitioned idle (= not hosting any VMs) LCs into a lower power-state.

- *energyManagement.numberOfReservedNodes*

You can set the number of reserved nodes using this parameter. In case of a positive value the GM will keep the predefined number of LCs always online.

- *energyManagement.powerSavingAction*

  In case the decision has been made to transition an idle LC into a lower power state the power saving action will be triggered. You can choose between: shutdown, suspendToRam, suspendToDisk, and suspendToBoth. Note that you have to make sure that your hardware supports the selected action.

- *energyManagement.drivers.*

  You can choose different drivers to shutdown and wake up the system. For example, shutdown can be achieved using the native system shutdown command or IPMI. Similarly, various Linux scripts exist to trigger suspend actions (e.g. pmutils, uswsusp). Finally, a several wake up methods such as IPMI and Wake-On-Lan (WOL) exist with each of them requiring different arguments. You can use the options parameter to pass additional arguments (e.g. MAC address for WOL or authentication data for IPMI).

You can select the driver you think is supported in your environment or implement your own driver by implementing the appropriate driver interface (see Developers documentation for more details). Note that you have to configure your system properly by installing the appropriate tools required by the drivers and give them enough system access rights to operate correctly. Please see the Deployment documentation for more details.

- *energyManagement.thresholds.idleTime*

The time interval in seconds used to observe the LC load (e.g. every 120 seconds).

- *energyManagement.thresholds.wakeupTime*

The time to wait in seconds until a LC is considered active. Note that it is crucial for the proper functioning of the system to set this interval as accurate as possible. For example, if your system needs approximately 5 minutes to come back online (which is actually the right value for some modern Blade HP servers) after being shutdown, set wake up time to 300 seconds!

- *energyManagement.commandExecutionTimeout*

Sometimes the implemented drivers commands can be fragile and block the system if they do not terminate correctly. You can use the command execution timeout to force termination after a predefined number of seconds. Note that it is important to keep this value high enough in order to prevent situations where driver commands are aborted too early.

1.2 Developper Guide

1.2.1 Python driver

Apache Libcloud (http://libcloud.apache.org/) driver will allow you to interact with snooze from a python script.

You need the code of the apache-libcloud project :

```
git clone https://github.com/apache/libcloud.git
git checkout 0.12.1
```

Next, install the specific files for Snooze driver from : https://github.com/msimonin/snooze-libcloud

You need to add three files :

```
libcloud/compute/providers.py
libcloud/compute/drivers/snooze.py
libcloud/compute/types.py
```

Use it

You can copy/paste the following code in the python interpreter or in a script file.

- First you need to import the requires libraries.

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver
```

- Next, connect to the driver. Depending on your configuration pass one bootstrap address to the driver, here we assume that a bootstrap is listenning on *127.0.0.1:5000*.

```
Snooze = get_driver(Provider.SNOOZE)
driver = Snooze("127.0.0.1","5000")
```

Now you are ready to create a virtual machine and check its state.

```
n1 = driver.create_node(libvirt_templates=["vmtemplate.xml"], tx=12800, rx=12800)
print "VM %s status %s"%(n1.name,n1.state)
```

1.2.2 API Reference

Bootstrap

Methods Summary	
getGroupLeaderDescription	
getCompleteHierarchy	

In this section we assume that a *bootstrap* node is running on *localhost*.

getGroupLeaderDescription

Method GET

Returns The group manager description object of the group leader.

Using it

- Curl

```
($) curl localhost:5000/bootstrap?getGroupLeaderDescription

{
  "id":"f5cfb585-800e-4ff8-a97f-5681d0610892",
  "hostname":"mafalda",
  "listenSettings":{
     "controlDataAddress":{
        "address":"127.0.0.1",
        "port":5001
     },
     "monitoringDataAddress":{
        "address":"127.0.0.1",
        "port":6000
     }
  },
  "localControllers":{
  },
  "heartbeatAddress":{
     "address":"UNKNOWN",
     "port":-1
  },
  "summaryInformation":{
  },
  "virtualMachines":[
  ]
}
```

- Java

```
import org.inria.myriads.snoozecommon.communication.NetworkAddress;
import org.inria.myriads.snoozecommon.communication.groupmanager.GroupManagerDescription;
import org.inria.myriads.snoozecommon.communication.rest.CommunicatorFactory;
import org.inria.myriads.snoozecommon.communication.rest.api.BootstrapAPI;

[...]
```

```
NetworkAddress bootstrapAddress = new NetworkAddress();
bootstrapAddress.setAddress("localhost");
bootstrapAddress.setPort(5000);
BootstrapAPI bootstrapCommunicator = CommunicatorFactory.newBootstrapCommunicator(bootstrapAddress);
GroupManagerDescription groupLeaderDescription = bootstrapCommunicator.getGroupLeaderDescription();
```

- Python (libcloud)

```
>> from libcloud.compute.types import Provider
>> from libcloud.compute.providers import get_driver

>> Snooze = get_driver(Provider.SNOOZE)
>> driver = Snooze("127.0.0.1","5000")

>> data = driver.get_groupleader()

>> print data
```

```
{
   "summaryInformation": {},
   "localControllers": {},
   "hostname": "mafalda",
   "heartbeatAddress": {
      "port": -1,
      "address": "UNKNOWN"
   },
   "virtualMachines": [],
   "listenSettings": {
      "controlDataAddress": {
         "port": 5001,
         "address": "127.0.0.1"
      },
      "monitoringDataAddress": {
         "port": 6000,
         "address": "127.0.0.1"
      }
   },
   "id": "f5cfb585-800e-4ff8-a97f-5681d0610892"
}
```

getCompleteHierarchy

Method GET

Returns The complete hierarchy of the systems.

Using it

- Curl (2 GMs - 1 LC)

```
($) curl http://localhost:5000/bootstrap?getCompleteHierarchy
```

```
{
  "groupManagerDescriptions":[
     {
```

```
"id":"20fb798c-5d72-47d7-b80e-a613a81dc603",
"listenSettings":{
   "controlDataAddress":{
      "address":"127.0.0.1",
      "port":5002
   },
   "monitoringDataAddress":{
      "address":"127.0.0.1",
      "port":6001
   }
},
"localControllers":{
   "ddb07acb-643e-4f9d-87e3-23ae4b629509":{
      "id":"ddb07acb-643e-4f9d-87e3-23ae4b629509",
      "controlDataAddress":{
         "address":"127.0.0.1",
         "port":5003
      },
      "status":"ACTIVE",
      "hypervisorSettings":{
         "port":16509,
         "driver":"qemu",
         "transport":"tcp",
         "migration":{
            "method":"live",
            "timeout":60
         }
      },
      "totalCapacity":[
         4.0,
         3958348.0,
         131072.0,
         131072.0
      ],
      "wakeupSettings":{
         "driver":"IPMI",
         "options":"-I lanplus -H BMC_IP -U user -P password"
      },
      "hostname":"mafalda",
      "virtualMachineMetaData":{

      },
      "assignedVirtualMachines":[

      ]
   }
},
"heartbeatAddress":{
   "address":"225.4.5.6",
   "port":10001
},
"hostname":"mafalda",
"summaryInformation":{
   "1373888959514":{
      "timeStamp":1373888959514,
      "usedCapacity":[
         0.0,
         0.0,
            0.0,
            0.0
         ],
         "requestedCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "localControllers":[

         ],
         "activeCapacity":[
            4.0,
            3958348.0,
            131072.0,
            131072.0
         ],
         "passiveCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "legacyIpAddresses":[

         ]
      },
      "1373888956509":{
         "timeStamp":1373888956509,
         "usedCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "requestedCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "localControllers":[

         ],
         "activeCapacity":[
            4.0,
            3958348.0,
            131072.0,
            131072.0
         ],
         "passiveCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "legacyIpAddresses":[
      ]
   },
   "1373888953504":{
      "timeStamp":1373888953504,
      "usedCapacity":[
         0.0,
         0.0,
         0.0,
         0.0
      ],
      "requestedCapacity":[
         0.0,
         0.0,
         0.0,
         0.0
      ],
      "localControllers":[

      ],
      "activeCapacity":[
         4.0,
         3958348.0,
         131072.0,
         131072.0
      ],
      "passiveCapacity":[
         0.0,
         0.0,
         0.0,
         0.0
      ],
      "legacyIpAddresses":[

      ]
   },
   "1373888950498":{
      "timeStamp":1373888950498,
      "usedCapacity":[
         0.0,
         0.0,
         0.0,
         0.0
      ],
      "requestedCapacity":[
         0.0,
         0.0,
         0.0,
         0.0
      ],
      "localControllers":[

      ],
      "activeCapacity":[
         4.0,
         3958348.0,
         131072.0,
         131072.0
      ],
            "passiveCapacity":[
               0.0,
               0.0,
               0.0,
               0.0
            ],
            "legacyIpAddresses":[

            ]
         },
         "1373888947493":{
            "timeStamp":1373888947493,
            "usedCapacity":[
               0.0,
               0.0,
               0.0,
               0.0
            ],
            "requestedCapacity":[
               0.0,
               0.0,
               0.0,
               0.0
            ],
            "localControllers":[

            ],
            "activeCapacity":[
               4.0,
               3958348.0,
               131072.0,
               131072.0
            ],
            "passiveCapacity":[
               0.0,
               0.0,
               0.0,
               0.0
            ],
            "legacyIpAddresses":[

            ]
         },
         "1373888944487":{
            "timeStamp":1373888944487,
            "usedCapacity":[
               0.0,
               0.0,
               0.0,
               0.0
            ],
            "requestedCapacity":[
               0.0,
               0.0,
               0.0,
               0.0
            ],
            "localControllers":[
         ],
         "activeCapacity":[
            4.0,
            3958348.0,
            131072.0,
            131072.0
         ],
         "passiveCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "legacyIpAddresses":[

         ]
      },
      "1373888941482":{
         "timeStamp":1373888941482,
         "usedCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "requestedCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "localControllers":[

         ],
         "activeCapacity":[
            4.0,
            3958348.0,
            131072.0,
            131072.0
         ],
         "passiveCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
         "legacyIpAddresses":[

         ]
      },
      "1373888938477":{
         "timeStamp":1373888938477,
         "usedCapacity":[
            0.0,
            0.0,
            0.0,
            0.0
         ],
        "requestedCapacity":[
           0.0,
           0.0,
           0.0,
           0.0
        ],
        "localControllers":[

        ],
        "activeCapacity":[
           4.0,
           3958348.0,
           131072.0,
           131072.0
        ],
        "passiveCapacity":[
           0.0,
           0.0,
           0.0,
           0.0
        ],
        "legacyIpAddresses":[

        ]
     },
     "1373888935472":{
        "timeStamp":1373888935472,
        "usedCapacity":[
           0.0,
           0.0,
           0.0,
           0.0
        ],
        "requestedCapacity":[
           0.0,
           0.0,
           0.0,
           0.0
        ],
        "localControllers":[

        ],
        "activeCapacity":[
           4.0,
           3958348.0,
           131072.0,
           131072.0
        ],
        "passiveCapacity":[
           0.0,
           0.0,
           0.0,
           0.0
        ],
        "legacyIpAddresses":[

        ]
     },
           "1373888932467":{
              "timeStamp":1373888932467,
              "usedCapacity":[
                 0.0,
                 0.0,
                 0.0,
                 0.0
              ],
              "requestedCapacity":[
                 0.0,
                 0.0,
                 0.0,
                 0.0
              ],
              "localControllers":[

              ],
              "activeCapacity":[
                 4.0,
                 3958348.0,
                 131072.0,
                 131072.0
              ],
              "passiveCapacity":[
                 0.0,
                 0.0,
                 0.0,
                 0.0
              ],
              "legacyIpAddresses":[

              ]
           }
        },
        "virtualMachines":[

        ]
     }
  ]
}
```

- Java

```
import org.inria.myriads.snoozecommon.communication.NetworkAddress;
import org.inria.myriads.snoozecommon.communication.groupmanager.repository.GroupLeaderRepositoryInfo
import org.inria.myriads.snoozecommon.communication.rest.CommunicatorFactory;
import org.inria.myriads.snoozecommon.communication.rest.api.BootstrapAPI;

[...]

NetworkAddress bootstrapAddress = new NetworkAddress();
bootstrapAddress.setAddress("localhost");
bootstrapAddress.setPort(5000);
BootstrapAPI bootstrapCommunicator = CommunicatorFactory.newBootstrapCommunicator(bootstrapAddress);
GroupLeaderRepositoryInformation hierarchy = bootstrapCommunicator.getCompleteHierarchy();
```

- Python (libcloud)

Not implemented

Group Leader

Methods Summary	
startVirtualCluster	
getVirtualClusterResponse	

In this section we assume that the group leader node is running on localhost.

startVirtualCluster

Method POST

Input Data You must provide a JSON encoded hash in the body of your request that must contain a parameter *virtualMachineTemplates* which value is an array of virtualMachineTemplate.

virtualMachineTemplate must contain the following parameters:

- libVirtTemplate : xml string representing the virtual machine to start
- networkCapacityDemand : hash which parameters are rxBytes and txBytes.

A example is given below with one virtual machine:

```
{
   "virtualMachineTemplates":[
      {
         "libVirtTemplate":"<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?><domain type=
         "networkCapacityDemand":{
            "rxBytes":12800.0,
            "txBytes":12800.0
         }
      }
   ]
}
```

Return The assigned task identifier.

Using it

- Curl

```
($) curl -H "Content-Type: application/json" -X POST -d "$template" http://localhost:5001/groupmanage

a4fba1c6-4c8f-4691-a530-01c067e8deb2
```

- Java

```
// Gets the leader address.
NetworkAddress bootstrapAddress = new NetworkAddress();
bootstrapAddress.setAddress("localhost");
bootstrapAddress.setPort(5000);
BootstrapAPI bootstrapCommunicator = CommunicatorFactory.newBootstrapCommunicator(bootstrapAddress);
GroupManagerDescription groupLeaderDescription = bootstrapCommunicator.getGroupLeaderDescription();

// Construct the virtual machine templates array.
VirtualClusterSubmissionRequest virtualClusterDescription = new VirtualClusterSubmissionRequest();
ArrayList<VirtualMachineTemplate> virtualMachineTemplates = new ArrayList<VirtualMachineTemplate>();
```

```
VirtualMachineTemplate virtualMachineTemplate = new VirtualMachineTemplate();
virtualMachineTemplate.setLibVirtTemplate("[...]");
NetworkDemand networkDemand = new NetworkDemand();
networkDemand.setRxBytes(12800);
networkDemand.setTxBytes(12800);
virtualMachineTemplate.setNetworkCapacityDemand(networkDemand);

virtualMachineTemplates.add(virtualMachineTemplate);
virtualClusterDescription.setVirtualMachineTemplates(virtualMachineTemplates);

// Call to the startVirtualCluster Method.
NetworkAddress groupLeaderAddress = groupLeaderDescription.getListenSettings().getControlDataAddress
GroupManagerAPI groupLeaderCommunicator = CommunicatorFactory.newGroupManagerCommunicator(groupLeade
String taskIdentifier = groupLeaderCommunicator.startVirtualCluster(virtualClusterDescription);
```

- Python (libcloud)

When using this driver, polling for the response is wrapped in the *create_node* method of the driver.

```
>> from libcloud.compute.types import Provider
>> from libcloud.compute.providers import get_driver
>> Snooze = get_driver(Provider.SNOOZE)
>> driver = Snooze("127.0.0.1","5000")

>> driver.create_node(libvirt_templates=["/home/msimonin/Images-VM/Snooze-images/vmtemplates/debian1

Out << [<Node: uuid=b69042832fd33b092eb5d0211554d3c983c80ddf, name=debian1, state=RUNNING, public_ip
```

getVirtualClusterResponse

Method POST

Input Data You must provide a JSON encoded hash in the body of your request that must contain a string identifying the cluster submission request.

Return The virtual cluster response or empty if the response isn't available yet.

Using it

- Curl

```
($) curl  -H "Content-Type: application/json" -X POST -d "a4fba1c6-4c8f-4691-a530-01c067e8deb2" http
```

```
{
   "errorCode":null,
   "virtualMachineMetaData":[
      {
         "status":"RUNNING",
         "virtualMachineLocation":{
            "virtualMachineId":"debian1",
            "localControllerId":"ddb07acb-643e-4f9d-87e3-23ae4b629509",
            "groupManagerId":"20fb798c-5d72-47d7-b80e-a613a81dc603",
            "groupManagerControlDataAddress":{
               "address":"127.0.0.1",
               "port":5002
            },
            "localControllerControlDataAddress":{
               "address":"127.0.0.1",
               "port":5003
            }
         },
         "usedCapacity":{

         },
         "requestedCapacity":[
            1.0,
            200000.0,
            12800.0,
            12800.0
         ],
         "ipAddress":"192.168.122.6",
         "errorCode":"UNKNOWN",
         "groupManagerControlDataAddress":{
            "address":"127.0.0.1",
            "port":5002
         },
         "xmlRepresentation":"<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?><domain typ
      }
   ]
}
```

- Java

```
// This code can follow the code from above.
GroupManagerAPI groupLeaderCommunicator = CommunicatorFactory.newGroupManagerCommunicator(networkAddr
virtualClusterResponse = groupLeaderCommunicator.getVirtualClusterResponse(taskIdentifier);
```

Group Manager

External Methods	Internal Methods
destroyVirtualMachine	
rebootVirtualMachine	
resumeVirtualMachine	
shutdownVirtualMachine	
suspendVirtualMachine	

In this section we assume that the group leader node is running on localhost.

destroyVirtualMachine

Method POST

Input Data You must provide a JSON encoded hash in the body of your request which correspond to the Virtual-MachineMachineLocation java object.

Return boolean.

Using it

- Python (libcloud)

The mechanism of getting the location of the virtual machine is wrapped in the call.

```
>> from libcloud.compute.types import Provider
>> from libcloud.compute.providers import get_driver
>> Snooze = get_driver(Provider.SNOOZE)
>> driver = Snooze("127.0.0.1","5000")

>> l = driver.create_node(libvirt_templates=["/home/msimonin/Images-VM/Snooze-images/vmtemplates/debi

Out << [<Node: uuid=b69042832fd33b092eb5d0211554d3c983c80ddf, name=debian1, state=RUNNING, public_ips

>> driver.destroy(l[0])
```

rebootVirtualMachine

Method POST

Input Data You must provide a JSON encoded hash in the body of your request which correspond to the Virtual-MachineMachineLocation java object.

Return boolean.

Using it

- Python (libcloud)

```
>> driver.reboot(l[0])
```

resumeVirtualMachine

Method POST

Input Data You must provide a JSON encoded hash in the body of your request which correspond to the Virtual-MachineMachineLocation java object.

Return boolean.

Using it

- Python (libcloud)

```
>> driver.resume(l[0])
```

shutdownVirtualMachine

Method POST

Input Data You must provide a JSON encoded hash in the body of your request which correspond to the VirtualMachineMachineLocation java object.

Return boolean.

Using it

- Python (libcloud)

```
>> driver.shutdown(l[0])
```

suspendVirtualMachine

Method POST

Input Data You must provide a JSON encoded hash in the body of your request which correspond to the VirtualMachineMachineLocation java object.

Return boolean.

Using it

- Python (libcloud)

```
>> driver.suspend(l[0])
```

Local Controller

Exposed Methods	
getGroupLeaderDescription	
getCompleteHierarchy	
startVirtualCluster	
getVirtualClusterResponse	
destroyVirtualMachine	
rebootVirtualMachine	
resumeVirtualMachine	
shutdownVirtualMachine	
suspendVirtualMachine	

CHAPTER

TWO

SUPPORT

- *search*
- mailing list
- issue tracker
- contact form