Technical document on ConPaaS services: ConPaaS MySQL Server

Aleš Černivec

March 6, 2012

Contents

1 Introduction

Currently you can add and remove agent nodes, query for status of the agent nodes, configure users, upload mysql database.

2 Architecture

ConPaaS MySQL Manager node has to be run manually or by the usage of ConPaaS web front-end. ConPaaS Servers can also be managed directly through web front-end (see section 5) or by the use of the script provided under `trunk/conpaas/conpaas-mysql/scripts/manager/run-manager-client.sh`.

When the manager VM starts, it fetches the fresh package of **conpaas-mysql** sources from the Contrail SVN with anonymous username. There is no need to install anything into the VM before running the conpaas-mysql manager. The template for creating the manager, contains details on installation script.

When obtaining access point of the manager, new agents can be provisioned by issuing HTTP POST `add_nodes` command on resource `sql-manager-host:/`:

```
POST / HTTP/1.1
Accept: */*
Content-Type: application/json
Content-Length: 67
{"params": {"function": "agent"}, "method": "add_nodes", "id": "1"}
```

Parameter `function` will soon support more than just creating new agent nodes (e.g. cluster manager, cluster agent, cluster). Parameter `method` designates command `add_nodes` and `id` equals 1.

3 Configuration

Configuration of the service consists of configuration file of the service describing **iaas section**, **manager section** and **onevm_agent_template section**. In the appendix we give `init.sh` of the manager which initializes missing variables. For the provisioning we need also a template to be used by the iaas create command.

3.1 Initialisation file `init.sh`

Frontend or the user manually has to set up these variables:

- `[iaas] DRIVER`
- `[iaas] OPENNEBULA_URL`
- `[iaas] OPENNEBULA_USER`
- `[iaas] OPENNEBULA_PASSWORD`
- `[iaas] OPENNEBULA_IMAGE_ID`
- `[iaas] OPENNEBULA_NETWORK_ID`

- `[onevm_agent_template] IMAGE_ID` – The same as in `[iaas] OPENNEBULA_IMAGE_ID`
- `[onevm_agent_template] NETWORK_ID` – There are two variables. They should be the same as in `[iaas] OPENNEBULA_NETWORK_ID`.

An example of the configuration file:

```
[iaas]
DRIVER=OPENNEBULA_XMLRPC
OPENNEBULA_URL=http://10.30.1.14:2633/RPC2
OPENNEBULA_USER=oneadmin
OPENNEBULA_PASSWORD=oneadmin
OPENNEBULA_IMAGE_ID = 221
OPENNEBULA_SIZE_ID = 1
OPENNEBULA_NETWORK_ID = 205
OPENNEBULA_NETWORK_GATEWAY=$IP_GATEWAY
OPENNEBULA_NETWORK_NAMESERVER=$NAMESERVER

[manager]
find_existing_agents = true
logfile=/var/log/conpaassql-stdout.log

[onevm_agent_template]
FILENAME=/root/conpaassql/scripts/one-scripts/agent/agent.template
USERDATA=/root/conpaassql/scripts/one-scripts/agent/init.sh
NAME=conpaassql_server
CPU=0.2
MEM_SIZE=256
DISK=readonly=no
OS=arch=x86_64,boot=hd
IMAGE_ID=221
NETWORK_ID=205
CONTEXT=vmid=$VMID,vmname=$NAME,ip_public="$NIC[IP, NETWORK_ID=205]", \
 ip_gateway="$IP_GATEWAY",netmask="$NETMASK",nameserver="$NAMESERVER", \
 userdata=$AGENT_USER_DATA
```

The file resirdes on the SVN:
`trunk/conpaas/conpaas-mysql/scripts/one-scripts/manager/init.sh`.

3.2 Manager template file `manager.template`

After the `init.sh` has been modified, we need to use a manager template to initiate the manager service. Here, we give a template used to provision the manager. Variable that need to be modified by the user or by the frontend, are these:

- `image_id`
- `network_id`
- `hostname`

- ip_gateway
- netmask
- nameserver
- userdata This is hexdump, therefore we need an external programm to generate this for us (see Appendix B).

```
NAME = conpaas-sql-manager CPU = 0.5 MEMORY = 128
   OS     = [ boot="hd", arch="x86_64" ]
DISK   = [
   image_id = 221,
   readonly = "no"
]
NIC    = [
   NETWORK_ID = 205
   ]
GRAPHICS = [
  type="vnc"
  ]
CONTEXT = [
 hostname="ConPaaSSQL-01",
 ip_public  = "$NIC[IP, NETWORK_ID=205]",
 ip_gateway  = "10.1.0.254",
 netmask = "255.255.255.0",
 nameserver = "192.168.122.1",
 userdata = $USERDATA
]
RANK = "- RUNNING_VMS"
```

Variable USERDATA needs to be filled with beforehand created manager's init.sh. In Appendix B we give you an example python script (provided in the SVN under trunk/conpaas/conpaas-mysql/bin/create_one_template.py. The usage is python create_one_template.py manager.template init.sh. It takes the manager.template file and fills in the USERDATA variable with hexdump of the init.sh. On the output we can see the generated template (save it as newtemplate) which can be used with the command for creating the VM: onevm create newtemple. After this the manager is up and running and available to be used with the client program.

4 Client programm run-manager-client.sh

The script resides on the SVN:
trunk/conpaas/conpaas-mysql/scripts/manager/run-manager-client.sh. Some examples of running the service:

```
#./run-manager-client.sh 10.1.0.47 50000 list_nodes
#./run-manager-client.sh 10.1.0.47 50000 add_nodes 2
#./run-manager-client.sh 10.1.0.47 50000 remove_nodes count 2
```

```
./run-manager-client.sh 10.1.0.47 50000 send_mysqldump all dbtest
./run-manager-client.sh 10.1.0.47 50000 configure_user all ales ales
 485  ./run-manager-client.sh 10.1.0.47 50000 remove_nodes count 2
```

5 ConPaaS MySQL Server Web front-end

ConPaaS MySQL Server Web front-end can easily be installed after the server already runs:

```
# apt-get install python-setuptools python-dev python-pycurl -y
# easy_install pip
# pip install oca apache-libcloud
# pip install --extra-index-url http://eggs.contrail.xlab.si \
conpaassql-manager-gui
# mkdir -p /etc/conpaassql
# cat > /etc/conpaassql/manager-gui.conf << EOF
# MANAGER_HOST = 'localhost'
# EOF
# screen -S conpaasssql-manager-gui -d -m conpaassql_manager_gui
```

It also is not mandatory that the web front-end is installed on the same server as the SQL Server already runs.

6 ConPaaS MySQL Server Manager API

Module `conpaas.mysql.server.manager.internals` contains internals of the ConPaaS MySQL Server. ConPaaS MySQL Server consists of several nodes with different roles.

- Manager node
- Agent node(s)
  - Master
  - Slave(s)

platform: Linux, Debian Squeeze, tested also within Ubuntu 10.10 (there should be no problem when using later distributions).
synopsis: Internals of ConPaaS MySQL Servers.
moduleauthor: Ales Cernivec <ales.cernivec@xlab.si>

`conpaas.mysql.server.manager.internals.add_nodes(kwargs)`

Description:

HTTP POST method. Creates new node and adds it to the list of existing nodes

in the manager. A role of new node can be one of: agent, manager. Currently only agent is supported. It makes internal call to `createServiceNodeThread()`.

Parameters:

kwargs – string describing a function (agent).

Returns:

HttpJsonResponse - JSON response with details about the node.

Raises:

ManagerException

Example

```
POST / HTTP/1.1
Accept: */*
Content-Type: application/json

Body content: {"params": {"function": "agent"}, "method": /
"add_nodes", "id": "1"}
```

`conpaas.mysql.server.manager.internals.remove_nodes(params)`

Description:

HTTP POST method. Deletes specific node from a pool of agent nodes. Node deleted is given by `{'serviceNodeId':id}`.

Parameters:

kwargs –string identifying a node.

Returns:

HttpJsonResponse - HttpJsonResponse - JSON response with details about the node. OK if everything went well.

Raises:

ManagerException if something went wrong. It contains a detailed description about the error.

Example

```
POST / HTTP/1.1
Accept: */*
Content-Type: application/json

Body content: {"params": {"serviceNodeId": "12"}, "method": /
"remove_nodes", "id": "1"}
```

`conpaas.mysql.server.manager.internals.list_nodes()`

Description:

HTTP GET method. Uses `IaaSClient.listVMs()` to get list of all service nodes. For each service node it checks if it is in servers list. If some of them are missing they are removed from the list. Returns list of all service nodes.

Parameters:

Returns:

HttpJsonResponse - JSON response with the list of services: `{ 'serviceNode': [<a list of ids>]})`

Raises:

HttpErrorResponse

Example

```
GET /?method=list_nodes&id=1 HTTP/1.1
Accept: */*
Content-Type: application/json
```

`conpaas.mysql.server.manager.internals.get_node_info()`

Description:

HTTP GET method. Gets info of a specific node.

Parameters:

param (str) – serviceNodeId is a VMID of an existing service node.

Returns:

HttpJsonResponse - JSON response with details about the node: : `{'serviceNode':{'id': serviceNode.vmid,'ip': serviceNode.ip,'isRunningMySQL': serviceNode.isRunningMySQL}}`.

Raises:

ManagerException

Example (note the VMID inside the command)

```
GET /?params=%7B%22serviceNodeId%22%3A+%22<VMID>%22%7D&method=get_node_info&id=1 HTTP/1.1
Accept: */*
Content-Type: application/json
```

`conpaas.mysql.server.manager.internals.get_service_info()`

Description:

HTTP GET method. Returns the current state of the manager.

Parameters:

param (str) – serviceNodeId is a VMID of an existing service node.

Returns:

HttpJsonResponse - JSON response with the description of the state.

Raises:

ManagerException

Example

```
GET /?method=get_service_info&id=1 HTTP/1.1
Accept: */*
Content-Type: application/json
```

`conpaas.mysql.server.manager.internals.set_up_replica_master()`

Description:

HTTP POST method. Sets up a replica master node.

Parameters:

id – new replica master id.

Returns:

HttpJsonResponse - JSON response with details about the new node. ManagerException if something went wrong.

Raises:

ManagerException

`conpaas.mysql.server.manager.internals.set_up_replica_slave()`

Description:

HTTP POST method. Sets up a replica master node.

Parameters:

id – new replica slave id.

Returns:

HttpJsonResponse - JSON response with details about the new node. ManagerException if something went wrong.

Raises:

ManagerException

`conpaas.mysql.server.manager.internals.shutdown()`

Description:

HTTP POST method. Shuts down the manager service.

Parameters:

id – new replica slave id.

Returns:

HttpJsonResponse - JSON response with details about the status of a manager

node: :py:attr‘S_EPILOGUE‘. ManagerException if something went wrong.

Raises:

ManagerException

`conpaas.mysql.server.manager.internals.get_service_performance()`

Description:

HTTP GET method. Placeholder for obtaining performance metrics.

Parameters:

kwargs (dict) – Additional parameters.

Returns:

HttpJsonResponse – returns metrics

Raises:

Example

```
GET /?method=get_service_performance&id=1 HTTP/1.1
Accept: */*
Content-Type: application/json
```

7 Conclusion

8 Appendix A

In this appendix the content of manager's `init.sh` is given.

```
#!/bin/bash

if [ -f /mnt/context.sh ]; then
  . /mnt/context.sh
fi

ifconfig eth0 $IP_PUBLIC netmask $NETMASK
route add default gw $IP_GATEWAY eth0
echo nameserver $NAMESERVER > /etc/resolv.conf

# installation
apt-get update
apt-get install -y unzip python python-mysqldb python-pycurl \
        python-dev python-setuptools python-pip subversion

cd /root
svn co svn://svn.forge.objectweb.org/svnroot/contrail/trunk/conpaas/conpaas-mysql conpaas
cd conpaassql
python setup.py bdist_egg
easy_install dist/conpaas*

cat > /root/conpaassql/src/conpaas/mysql/server/manager/configuration.cnf << EOF
[iaas]
DRIVER=OPENNEBULA_XMLRPC
OPENNEBULA_URL=http://10.30.1.14:2633/RPC2
OPENNEBULA_USER=oneadmin
OPENNEBULA_PASSWORD=oneadmin
OPENNEBULA_IMAGE_ID = 221
OPENNEBULA_SIZE_ID = 1
OPENNEBULA_NETWORK_ID = 205
OPENNEBULA_NETWORK_GATEWAY=$IP_GATEWAY
OPENNEBULA_NETWORK_NAMESERVER=$NAMESERVER

[manager]
find_existing_agents = true
logfile=/var/log/conpaassql-stdout.log

[onevm_agent_template]
FILENAME=/root/conpaassql/scripts/one-scripts/agent/agent.template
USERDATA=/root/conpaassql/scripts/one-scripts/agent/init.sh
NAME=conpaassql_server
CPU=0.2
MEM_SIZE=256
DISK=readonly=no
OS=arch=x86_64,boot=hd
IMAGE_ID=221
```

```
NETWORK_ID=205
CONTEXT=vmid=$VMID,vmname=$NAME,ip_public="$NIC[IP, NETWORK=\"private-lan\"]", \
        ip_gateway="$IP_GATEWAY",netmask="$NETMASK",nameserver="$NAMESERVER", \
        userdata=$AGENT_USER_DATA
EOF

nohup python /root/conpaassql/src/conpaas/mysql/server/manager/server.py -p 50000 \
        -b $IP_PUBLIC -c /root/conpaassql/src/conpaas/mysql/server/manager/configuration.
        /var/log/conpaassql-stdout.log 2> /var/log/conpaassql-err.log &
```

9 Appendix B

In this appendix the content of script for the manager's provisioning. The usage is `python create_one_template.py manager.template init.sh`. It takes the `manager.template` file and fills in the `USERDATA` variable with hex-dump of the `init.sh`.

```
#!/usr/bin/python

import sys, string

def read_template(file):
        fin = open(file, "r")
        templatestr = fin.read()
        fin.close()
        return string.Template(templatestr)

def read_userdata(file):
        fin = open(file, "r")
        templatestr = fin.read()
        fin.close()
        return templatestr

if __name__ == '__main__':
    file = None
    user_data = None
    if sys.argv.__len__() == 3:
        file = str(sys.argv[1])
        user_data = str(sys.argv[2])
    else:
        print "Need input [template_file] [user_data]"
        exit(1)
    template = read_template (file)
    hex_user_data=read_userdata(user_data).encode('hex')
    template = template.substitute(USERDATA= hex_user_data, NIC = "$NIC")
    print template
```