ConPaaS Documentation

Release 1.5.0

The ConPaaS team <info@conpaas.eu>

May 21, 2015

CHAPTER 1

Introduction

ConPaaS (http://www.conpaas.eu) is an open-source runtime environment for hosting applications in the cloud which aims at offering the full power of the cloud to application developers while shielding them from the associated complexity of the cloud.

ConPaaS is designed to host both high-performance scientific applications and online Web applications. It runs on a variety of public and private clouds, and is easily extensible. ConPaaS automates the entire life-cycle of an application, including collaborative development, deployment, performance monitoring, and automatic scaling. This allows developers to focus their attention on application-specific concerns rather than on cloud-specific details.

ConPaaS is organized as a collection of services, where each service acts as a replacement for a commonly used runtime environment. For example, to replace a MySQL database, ConPaaS provides a cloud-based MySQL service which acts as a high-level database abstraction. The service uses real MySQL databases internally, and therefore makes it easy to port a cloud application to ConPaaS. Unlike a regular centralized database, however, it is self-managed and fully elastic: one can dynamically increase or decrease its processing capacity by requesting it to reconfigure itself with a different number of virtual machines.

CHAPTER 2

Installation

The central component of ConPaaS is called the *ConPaaS Director* (**cpsdirector**). It is responsible for handling user authentication, creating new applications, handling their life-cycle and much more. **cpsdirector** is a web service exposing all its functionalities via an HTTP-based API.

ConPaaS can be used either via a command line interface (called **cpsclient**) or through a web frontend (**cpsfrontend**). Recently a new experimental command line interface called **cps-tools** has become available (note: **cps-tools** requires Python 2.7). This document explains how to install and configure all the aforementioned components.

ConPaaS's **cpsdirector** and its two clients, **cpsclient** and **cpsfrontend**, can be installed on your own hardware or on virtual machines running on public or private clouds. If you wish to install them on Amazon EC2, the Official Debian Wheezy, Ubuntu 12.04 or Ubuntu 14.04 images are known to work well.

ConPaaS services are designed to run either in an *OpenStack* or *OpenNebula* cloud installation or in the *Amazon Web Services* cloud.

Installing ConPaaS requires to take the following steps:

1. Choose a VM image customized for hosting the services, or create a new one. Details on how to do this vary depending on the choice of cloud where ConPaaS will run. Instructions on how to find or create a ConPaaS image suitable to run on Amazon EC2 can be found in *ConPaaS on Amazon EC2*. The section *ConPaaS on OpenStack* describes how to create a ConPaaS image for OpenStack and section *ConPaaS on OpenNebula* describes how to create an image for OpenNebula.
2. Install and configure **cpsdirector** as explained in *Director installation*. All system configuration takes place in the director.
3. Install and configure **cpsclient** as explained in *Installing and configuring cpsclient.py*.
4. Install and configure **cps-tools** as explained in *Installing and configuring cps-tools*.
5. Install **cpsfrontend** and configure it to use your ConPaaS director as explained in *Frontend installation*.

2.1 Director installation

The ConPaaS Director is a web service that allows users to manage their ConPaaS applications. Users can create, configure and terminate their cloud applications through it. This section describes the process of setting up a ConPaaS director on a Debian/Ubuntu GNU/Linux system. Although the ConPaaS director might run on other distributions, only Debian versions 6.0 (Squeeze) and 7.0 (Wheezy), and Ubuntu versions 12.04 (Precise Pangolin) and 14.04 (Trusty Tahr) are officially supported. Also, only official APT repositories should be enabled in `/etc/apt/sources.list` and `/etc/apt/sources.list.d/`.

cpsdirector is available here: http://www.conpaas.eu/dl/cpsdirector-1.x.x.tar.gz. The tarball includes an installation script called `install.sh` for your convenience. You can either run it as root or follow the installation procedure outlined below in order to setup your ConPaaS Director installation.

1. Install the required packages:

```
    $ sudo apt-get update
    $ sudo apt-get install build-essential python-setuptools python-dev
    $ sudo apt-get install apache2 libapache2-mod-wsgi libcurl4-openssl-dev
```

2. Make sure that your system's time and date are set correctly by installing and running **ntpdate**:

```
    $ sudo apt-get install ntpdate
    $ sudo ntpdate 0.us.pool.ntp.org

    >> If the NTP socket is in use, you can type:
    $ sudo service ntp stop
    >> and again
    $ sudo ntpdate 0.us.pool.ntp.org
```

3. Download http://www.conpaas.eu/dl/cpsdirector-1.x.x.tar.gz and uncompress it
4. Run **`make install`** as root
5. After all the required packages are installed, you will get prompted for your hostname. Please provide your **public** IP address / hostname
6. Edit `/etc/cpsdirector/director.cfg` providing your cloud configuration. Among other things, you will have to choose an Amazon Machine Image (AMI) in case you want to use ConPaaS on Amazon EC2, an OpenStack image if you want to use ConPaaS on OpenStack, or an OpenNebula image if you want to use ConPaaS on OpenNebula. Section *ConPaaS on Amazon EC2* explains how to use the Amazon Machine Images provided by the ConPaaS team, as well as how to make your own images if you wish to do so. A description of how to create an OpenStack image suitable for ConPaaS is available in *ConPaaS on OpenStack* and *ConPaaS on OpenNebula* contains instructions for OpenNebula.

The installation process will create an *Apache VirtualHost* for the ConPaaS director in `/etc/apache2/sites-available/conpaas-director.conf` for Apache 2.4 or `/etc/apache2/sites-available/conpaas-director` for older versions of Apache. There should be no need for you to modify such a file, unless its defaults conflict with your Apache configuration.

Run the following commands as root to start your ConPaaS director for the first time:

```
$ sudo a2enmod ssl
$ sudo a2enmod wsgi
$ sudo a2ensite conpaas-director
$ sudo service apache2 restart
```

If you experience any problems with the previously mentioned commands, it might be that the default VirtualHost created by the ConPaaS director installation process conflicts with your Apache configuration. The Apache Virtual Host documentation might be useful to fix those issues: http://httpd.apache.org/docs/2.2/vhosts/.

Finally, you can start adding users to your ConPaaS installation as follows:

```
$ sudo cpsadduser.py
```

2.1.1 SSL certificates

ConPaaS uses SSL certificates in order to secure the communication between you and the director, but also to ensure that only authorized parties such as yourself and the various component of ConPaaS can interact with the system.

It is therefore crucial that the SSL certificate of your director contains the proper information. In particular, the *commonName* field of the certificate should carry the **public hostname of your director**, and it should match the *hostname* part of `DIRECTOR_URL` in `/etc/cpsdirector/director.cfg`. The installation procedure takes care of setting up such a field. However, should your director hostname change, please ensure you run the following commands:

```
$ sudo cpsconf.py
$ sudo service apache2 restart
```

2.1.2 Director database

The ConPaaS Director uses a SQLite database to store information about registered users and running services. It is not normally necessary for ConPaaS administrators to directly access such a database. However, should the need arise, it is possible to inspect and modify the database as follows:

```
$ sudo apt-get install sqlite3
$ sudo sqlite3 /etc/cpsdirector/director.db
```

If you have an existing installation (version 1.4.0 and earlier) you should upgrade your database to contain the extra `uuid` field needed for external IdP usage (see next topic) and the extra `openid` field needed for OpenID support:

```
$ sudo add-user-columns-to-db.sh
```

This script will warn you when you try to upgrade an already upgraded database.

On a fresh installation the database will be created on the fly.

2.1.3 Contrail IdP and SimpleSAML

ConPaaS can optionally delegate its user authentication to an external service. For registration and login through the Contrail Identification Provider you have to install the SimpleSAML package simplesamlphp-1.11.0 as follows:

```
$ wget http://simplesamlphp.googlecode.com/files/simplesamlphp-1.11.0.tar.gz
$ tar xzf simplesamlphp-1.11.0.tar.gz
$ cd simplesamlphp-1.11.0
$ cd cert ; openssl req -newkey rsa:2048 -new -x509 -days 3652 -nodes -out saml.crt -keyout saml.
```

Edit file `../metadata/saml20-idp-remote.php` and replace the `$metadata array` by the code found in the simpleSAMLphp flat file format part at the end of the browser output of https://multi.contrail.xlab.si/simplesaml/saml2/idp/metadata.php?output=xhtml .

Modify the authentication sources to contain the following lines (do not copy the line numbers):

```
$ cd ../config ; vi authsources.php
25                  // 'idp' => NULL,
26                  'idp' => 'https://multi.contrail.xlab.si/simplesaml/saml2/idp/metadata.php',

32                  //  next lines added by (your name)
33                  'privatekey' => 'saml.pem',
34                  'certificate' => 'saml.crt',
```

Copy your SimpleSAML tree to `/usr/share`

```
$ cd ../../
$ tar cf - simplesamlphp-1.11.0 | ( cd /usr/share ; sudo tar xf - )
```

Change ownerships:

```
$ cd /usr/share/simplesamlphp-1.11.0
$ sudo chown www-data www log
$ sudo chgrp www-data www log
```

Now edit `/etc/apache2/sites-enabled/default-ssl.conf` to contain the following lines (line numbers may vary depending on your current situation):

```
5          Alias /simplesaml /usr/share/simplesamlphp-1.11.0/www

18         <Directory /usr/share/simplesamlphp-1.11.0/www>
19                 Options Indexes FollowSymLinks MultiViews
20                 AllowOverride None
```

```
                Order allow,deny
                allow from all
        </Directory>
```

And the last thing to do: **register** your director domain name or IP at *contrail@lists.xlab.si*. This will enable you to use the federated login service provided by the Contrail project.

2.1.4 Multi-cloud support

ConPaaS services can be created and scaled on multiple heterogeneous clouds.

In order to configure **cpsdirector** to use multiple clouds, you need to set the `OTHER_CLOUDS` variable in the **[iaas]** section of `/etc/cpsdirector/director.cfg`. For each cloud name defined in `OTHER_CLOUDS` you need to create a new configuration section named after the cloud itself. Please refer to `/etc/cpsdirector/director.cfg.multicloud-example` for an example.

2.1.5 Virtual Private Networks with IPOP

Network connectivity between private clouds running on different networks can be achieved in ConPaaS by using IPOP (IP over P2P). This is useful in particular to deploy ConPaaS instances across multiple clouds. IPOP adds a virtual network interface to all ConPaaS instances belonging to an application, allowing services to communicate over a virtual private network as if they were deployed on the same LAN. This is achieved transparently to the user and applications - the only configuration needed to enable IPOP is to determine the network's base IP address, mask, and the number of IP addresses in this virtual network that are allocated to each service.

VPN support in ConPaaS is per-application: each application you create will get its own isolated IPOP Virtual Private Network. VMs running in the same application will be able to communicate with each other.

In order to enable IPOP you need to set the following variables in `/etc/cpsdirector/director.cfg`:

- `VPN_BASE_NETWORK`
- `VPN_NETMASK`
- `VPN_SERVICE_BITS`

Unless you need to access 172.16.0.0/12 networks, the default settings available in `/etc/cpsdirector/director.cfg.example` are probably going to work just fine.

The maximum number of services per application, as well as the number of agents per service, is influenced by your choice of `VPN_NETMASK` and `VPN_SERVICE_BITS`:

```
services_per_application = 2^VPN_SERVICE_BITS
agents_per_service = 2^(32 - NETMASK_CIDR - VPN_SERVICE_BITS) - 1
```

For example, by using 172.16.0.0 for `VPN_BASE_NETWORK`, 255.240.0.0 (/12) for `VPN_NETMASK`, and 5 `VPN_SERVICE_BITS`, you will get a 172.16.0.0/12 network for each of your applications. Such a network space will be then logically partitioned between services in the same application. With 5 bits to identify the service, you will get a maximum number of 32 services per application (2^5) and 32767 agents per service (2^(32-12-5)-1).

Optional: specify your own bootstrap nodes. When two VMs use IPOP, they need a bootstrap node to find each other. IPOP comes with a default list of bootstrap nodes from PlanetLab servers which is enough for most use cases. However, you may want to specify your own bootstrap nodes (replacing the default list). Uncomment and set `VPN_BOOTSTRAP_NODES` to the list of addresses of your bootstrap nodes, one address per line. A bootstrap node address specifies a protocol, an IP address and a port. For example:

```
VPN_BOOTSTRAP_NODES =
    udp://192.168.35.2:40000
    tcp://192.168.122.1:40000
    tcp://172.16.98.5:40001
```

2.1.6 Troubleshooting

If for some reason your Director installation is not behaving as expected, here are a few frequent issues and their solutions.

If you cannot create services, try to run this on the machine holding your Director:

1. Run the **cpscheck.py** command as root to attempt an automatic detection of possible misconfigurations.
2. Check your system's time and date settings as explained previously.
3. Test network connectivity between the director and the virtual machines deployed on the cloud(s) you are using.
4. Check the contents of `/var/log/apache2/director-access.log` and `/var/log/apache2/director-error.log`.

If services get created, but they fail to startup properly, you should try to ssh into your manager VM as root and:

1. Make sure that a ConPaaS manager process has been started:

```
root@conpaas:~# ps x | grep cpsmanage[r]
  968 ?        Sl     0:02 /usr/bin/python /root/ConPaaS/sbin/manager/php-cpsmanager -c /root
```

2. If a ConPaaS manager process has **not** been started, you should check if the manager VM can download a copy of the ConPaaS source code from the director. From the manager VM:

```
root@conpaas:~# wget --ca-certificate /etc/cpsmanager/certs/ca_cert.pem \
    `awk '/BOOTSTRAP/ { print $3 }' /root/config.cfg`/ConPaaS.tar.gz
```

   The URL used by your manager VM to download the ConPaaS source code depends on the value you have set on your Director in `/etc/cpsdirector/director.cfg` for the variable `DIRECTOR_URL`.

3. See if your manager's port **443** is open *and* reachable from your Director. In the following example, our manager's IP address is 192.168.122.15 and we are checking if *the director* can contact *the manager* on port 443:

```
root@conpaas-director:~# apt-get install nmap
root@conpaas-director:~# nmap -p443 192.168.122.15
Starting Nmap 6.00 ( http://nmap.org ) at 2013-05-14 16:17 CEST
Nmap scan report for 192.168.122.15
Host is up (0.00070s latency).
PORT    STATE SERVICE
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

4. Check the contents of `/root/manager.err`, `/root/manager.out` and `/var/log/cpsmanager.log`.
5. If the Director fails to respond to requests and you receive errors such as `No ConPaaS Director at the provided URL: HTTP Error 403:  Forbidden` or `403 Access Denied`, you need to allow access to the root file system, which is denied by default in newer versions of **apache2**. You can fix this by modifying the file `/etc/apache2/apache2.conf`. In particular, you need to replace these lines:

```
<Directory />
        Options FollowSymLinks
        AllowOverride all
        Order deny,allow
        Allow from all
</Directory>
```

   with these others:

```
    <Directory />
            Options Indexes FollowSymLinks Includes ExecCGI
            AllowOverride all
            Order deny,allow
            Allow from all
    </Directory>
```

2.2 Command line tool installation

There are two command line clients: an old one called `cpsclient.py` and a more recent one called `cps-tools`.

2.2.1 Installing and configuring cpsclient.py

The command line tool `cpsclient` can be installed as root or as a regular user. Please note that libcurl development files (binary package `libcurl4-openssl-dev` on Debian/Ubuntu systems) need to be installed on your system.

As root:

```
$ sudo easy_install http://www.conpaas.eu/dl/cpsclient-1.x.x.tar.gz
```

(do not forget to replace 1.x.x with the exact number of the ConPaaS release you are using)

Or, if you do not have root privileges, `cpsclient` can also be installed in a Python virtual environment if `virtualenv` is available on your machine:

```
$ virtualenv conpaas # create the 'conpaas' virtualenv
$ cd conpaas
$ source bin/activate # activate it
$ easy_install http://www.conpaas.eu/dl/cpsclient-1.x.x.tar.gz
```

Configuring `cpsclient.py`:

```
$ cpsclient.py credentials
Enter the director URL: https://your.director.name:5555
Enter your username: xxxxx
Enter your password:
Authentication succeeded
```

2.2.2 Installing and configuring cps-tools

The command line `cps-tools` is a more recent command line client to interact with ConPaaS. It has essentially a modular internal architecture that is easier to extend. It has also “object-oriented” arguments where “ConPaaS” objects are services, users, clouds and applications. The argument consists in stating the “object” first and then calling a sub-command on it. It also replaces the command line tool `cpsadduser.py`.

`cps-tools` requires:

- Python 2.7
- Python argparse module
- Python argcomplete module

If these are not yet installed, first follow the guidelines in *Installing Python2.7 and virtualenv*.

Installing `cps-tools`:

```
$ tar -xaf cps-tools-1.x.x.tar.gz
$ cd cps-tools-1.x.x
$ ./configure --sysconf=/etc
$ sudo make install
```

or:

```
$ make prefix=$HOME/src/virtualenv-1.11.4/ve install |& tee my-make-install.log
$  cd ..
$  pip install simplejson |& tee sjson.log
$  apt-get install libffi-dev |& tee libffi.log
$  pip install cpslib-1.x.x.tar.gz |& tee my-ve-cpslib.log
```

Configuring `cps-tools`:

```
$ mkdir -p $HOME/.conpaas
$ cp /etc/cps-tools.conf $HOME/.conpaas/
$ vim $HOME/.conpaas/cps-tools.conf
>> update 'director_url' and 'username'
>> do not update 'password' unless you want to execute scripts that must retrieve a certificate w
$ cps-user get_certificate
>> enter you password
>> now you can use cps-tools commands
```

2.2.3 Installing Python2.7 and virtualenv

Recommended installation order is first `python2.7`, then `virtualenv` (you will need about 0.5GB of free disk space). Check if the following packages are installed, and install them if not:

```
apt-get install gcc
apt-get install libreadline-dev
apt-get install -t squeeze-backports libsqlite3-dev libsqlite3-0
apt-get install tk8.4-dev libgdbm-dev libdb-dev libncurses-dev
```

Installing `python2.7`:

```
$ mkdir ~/src        (choose a directory)
$ cd ~/src
$ wget --no-check-certificate http://www.python.org/ftp/python/2.7.2/Python-2.7.2.tgz
$ tar xzf Python-2.7.2.tgz
$ cd Python-2.7.2
$ mkdir $HOME/.localpython
$ ./configure --prefix=$HOME/.localpython |& tee my-config.log
$ make |& tee my-make.log
>> here you may safely ignore complaints about missing modules: bsddb185   bz2   dl   imageop   s
$ make install |& tee my-make-install.log
```

Installing `virtualenv` (here version 1.11.4):

```
$ cd ~/src
$ wget --no-check-certificate http://pypi.python.org/packages/source/v/virtualenv/virtualenv-1.11
$ tar xzf virtualenv-1.11.4.tar.gz
$ cd virtualenv-1.11.4
$ $HOME/.localpython/bin/python setup.py install     (install virtualenv using P2.7)

$ $HOME/.localpython/bin/virtualenv ve -p $HOME/.localpython/bin/python2.7
New python executable in ve/bin/python2.7
Also creating executable in ve/bin/python
Installing setuptools, pip...done.
Running virtualenv with interpreter $HOME/.localpython/bin/python2.7
```

Activate `virtualenv`:

```
$ alias startVE='source $HOME/src/virtualenv-1.11.4/ve/bin/activate'
$ alias stopVE='deactivate'
$ startVE
(ve)$ python -V
Python 2.7.2
(ve)$
```

Install python argparse and argcomplete modules:

```
(ve)$ pip install argparse
(ve)$ pip install argcomplete
(ve)$ activate-global-python-argcomplete
```

2.3 Frontend installation

As for the Director, only Debian versions 6.0 (Squeeze) and 7.0 (Wheezy), and Ubuntu versions 12.04 (Precise Pangolin) and 14.04 (Trusty Tahr) are officially supported, and no external APT repository should be enabled. In a typical setup Director and Frontend are installed on the same host, but such does not need to be the case.

The ConPaaS Frontend can be downloaded from http://www.conpaas.eu/dl/cpsfrontend-1.x.x.tar.gz.

After having uncompressed it you should install the required packages:

```
$ sudo apt-get install libapache2-mod-php5 php5-curl
```

Copy all the files contained in the `www` directory underneath your web server document root. For example:

```
$ sudo cp -a www/ /var/www/
```

Copy `conf/main.ini` and `conf/welcome.txt` in your ConPaaS Director configuration folder (`/etc/cpsdirector`). Modify those files to suit your needs:

```
$ sudo cp conf/{main.ini,welcome.txt} /etc/cpsdirector/
```

Create a `config.php` file in the web server directory where you have chosen to install the frontend. `config-example.php` is a good starting point:

```
$ sudo cp www/config-example.php /var/www/config.php
```

Note that `config.php` must contain the `CONPAAS_CONF_DIR` option, pointing to the directory mentioned in the previous step

By default, PHP sets a default maximum size for uploaded files to 2Mb (and 8Mb to HTTP POST requests). However, in the web frontend, users will need to upload larger files (for example, a WordPress tarball is about 5Mb, a MySQL dump can be tens of Mb). To set higher limits, set the properties *post_max_size* and *upload_max_filesize* in file `/etc/php5/apache2/php.ini`. Note that property *upload_max_filesize* cannot be larger than property *post_max_size*.

Enable SSL if you want to use your frontend via https, for example by issuing the following commands:

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Details about the SSL certificate you want to use have to be specified in `/etc/apache2/sites-available/default-ssl`.

As a last step, restart your Apache web server:

```
$ sudo service apache2 restart
```

At this point, your front-end should be working!

2.4 Creating A ConPaaS Services VM Image

Various services require certain packages and configurations to be present in the VM image. ConPaaS provides facilities for creating specialized VM images that contain these dependencies. Furthermore, for the convenience of users, there are prebuilt Amazon AMIs that contain the dependencies for *all* available services. If you intend to run ConPaaS on Amazon EC2 and do not need a specialized VM image, then you can skip this section and proceed to *ConPaaS on Amazon EC2*.

2.4.1 Configuring your VM image

The configuration file for customizing your VM image is located at *conpaas-services/scripts/create_vm/create-img-script.cfg*.

In the **CUSTOMIZABLE** section of the configuration file, you can define whether you plan to run ConPaaS on Amazon EC2, OpenStack or OpenNebula. Depending on the virtualization technology that your target cloud uses, you should choose either KVM or Xen for the hypervisor. Note that for Amazon EC2 this variable needs to be set to Xen. Please do not make the recommended size for the image file smaller than the default. The *optimize* flag enables certain optimizations to reduce the necessary packages and disk size. These optimizations allow for smaller VM images and faster VM startup.

In the **SERVICES** section of the configuration file, you have the opportunity to disable any service that you do not need in your VM image. If a service is disabled, its package dependencies are not installed in the VM image. Paired with the *optimize* flag, the end result will be a minimal VM image that runs only what you need.

Note that te configuration file contains also a **NUTSHELL** section. The settings in this section are explained in details in *ConPaaS in a Nutshell*. However, in order to generate a regular customized VM image, make sure that both *container* and *nutshell* flags in this section are set to *false*.

Once you are done with the configuration, you should run this command in the *create_vm* directory:

```
$ python create-img-script.py
```

This program generates a script file named *create-img-conpaas.sh*. This script is based on your specific configurations.

2.4.2 Creating your VM image

To create the image you can execute *create-img-conpaas.sh* in any 64-bit Debian or Ubuntu machine. Please note that you will need to have root privileges on such a system. In case you do not have root access to a Debian or Ubuntu machine please consider installing a virtual machine using your favorite virtualization technology, or running a Debian/Ubuntu instance in the cloud.

1. Make sure your system has the following executables installed (they are usually located in `/sbin` or `/usr/sbin`, so make sure these directories are in your `$PATH`): *dd parted losetup kpartx mkfs.ext3 tune2fs mount debootstrap chroot umount grub-install*
2. It is particularly important that you use Grub version 2. To install it:

```
    sudo apt-get install grub2
```

3. Execute *create-img-conpaas.sh* as root.

The last step can take a very long time. If all goes well, the final VM image is stored as *conpaas.img*. This file is later registered to your target IaaS cloud as your ConPaaS services image.

2.4.3 If things go wrong

Note that if anything fails during the image file creation, the script will stop and it will try to revert any change it has done. However, it might not always reset your system to its original state. To undo everything the script has done, follow these instructions:

1. The image has been mounted as a separate file system. Find the mounted directory using command `df -h`. The directory should be in the form of `/tmp/tmp.X`.
2. There may be a `dev` and a `proc` directories mounted inside it. Unmount everything using:

```
sudo umount /tmp/tmp.X/dev /tmp/tmp.X/proc /tmp/tmp.X
```

3. Find which loop device you are using:

```
sudo losetup -a
```

4. Remove the device mapping:

```
sudo kpartx -d /dev/loopX
```

5. Remove the binding of the loop device:

```
sudo losetup -d /dev/loopX
```

6. Delete the image file
7. Your system should be back to its original state.

2.5 ConPaaS on Amazon EC2

ConPaaS is capable of running over the Elastic Compute Cloud (EC2) of Amazon Web Services (AWS). This section describes the process of configuring an AWS account to run ConPaaS. You can skip this section if you plan to install ConPaaS over OpenStack or OpenNebula.

If you are new to EC2, you will need to create an account on the Amazon Elastic Compute Cloud. A very good introduction to EC2 is Getting Started with Amazon EC2 Linux Instances.

2.5.1 Pre-built Amazon Machine Images

ConPaaS requires the usage of an Amazon Machine Image (AMI) to contain the dependencies of its processes. For your convenience we provide a pre-built public AMI, already configured and ready to be used on Amazon EC2, for each availability zone supported by ConPaaS. The AMI IDs of said images are:

- `ami-7a565912` United States East (Northern Virginia)
- `ami-b7dd31f3` United States West (Northern California)
- `ami-e57f49d5` United States West (Oregon)
- `ami-7f7e1108` Europe West (Ireland)
- `ami-3a0bc83a` Asia Pacific (Tokyo)
- `ami-fcdde1ae` Asia Pacific (Singapore)
- `ami-0b473b31` Asia Pacific (Sydney)
- `ami-a154d0bc` South America (Sao Paulo)

You can use one of these values when configuring your ConPaaS director installation as described in *Director installation*.

2.5.2 Registering your custom VM image to Amazon EC2

Using pre-built Amazon Machine Images is the recommended way of running ConPaaS on Amazon EC2, as described in the previous section. However, you can also create a new Amazon Machine Image yourself, for example in case you wish to run ConPaaS in a different Availability Zone or if you prefer to use a custom services image. If this is the case, you should have already created your VM image (*conpaas.img*) as explained in *Creating A ConPaaS Services VM Image*.

Amazon AMIs are either stored on Amazon S3 (i.e. S3-backed AMIs) or on Elastic Block Storage (i.e. EBS-backed AMIs). Each option has its own advantages; S3-backed AMIs are usually more cost-efficient, but if you plan to use t1.micro (free tier) your VM image should be hosted on EBS.

For an EBS-backed AMI, you should either create your *conpaas.img* on an Amazon EC2 instance, or transfer the image to one. Once *conpaas.img* is there, you should execute *register-image-ec2-ebs.sh* as root on the EC2 instance to register your AMI. The script requires your **EC2_ACCESS_KEY** and **EC2_SECRET_KEY** to proceed. At the end, the script will output your new AMI ID. You can check this in your Amazon dashboard in the AMI section.

For a S3-backed AMI, you do not need to register your image from an EC2 instance. Simply run *register-image-ec2-s3.sh* where you have created your *conpaas.img*. Note that you need an EC2 certificate with private key to be able to do so. Registering an S3-backed AMI requires administrator privileges. More information on Amazon credentials can be found at About AWS Security Credentials.

2.5.3 Security Group

An AWS security group is an abstraction of a set of firewall rules to limit inbound traffic. The default policy of a new group is to deny all inbound traffic. Therefore, one needs to specify a whitelist of protocols and destination ports that are accessible from the outside. The following ports should be open for all running instances:

- TCP ports 80, 443, 5555, 8000, 8080 and 9000 – used by the Web Hosting service
- TCP ports 3306, 4444, 4567, 4568 – used by the MySQL service with Galera extensions
- TCP ports 8020, 8021, 8088, 50010, 50020, 50030, 50060, 50070, 50075, 50090, 50105, 54310 and 54311 – used by the Map Reduce service
- TCP ports 4369, 14194 and 14195 – used by the Scalarix service
- TCP ports 2633, 8475, 8999 – used by the TaskFarm service
- TCP ports 32636, 32638 and 32640 – used by the XtreemFS service

AWS documentation is available at http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/index.html?using-network-security.html.

2.6 ConPaaS on OpenStack

ConPaaS can be deployed over an OpenStack installation. This section describes the process of configuring the DevStack version of OpenStack to run ConPaaS. You can skip this section if you plan to deploy ConPaaS over Amazon Web Services or OpenNebula.

In the rest of this section, the command-line examples assume that the user is authenticated and able to run OpenStack commands (such as `nova list`) on the controller node. If this is not the case, please refer first to the OpenStack documentation: http://docs.openstack.org/openstack-ops/content/lay_of_the_land.html.

2.6.1 Getting the OpenStack API access credentials

ConPaaS talks with an OpenStack deployment using the EC2 API, so first make sure that EC2 API access is enabled for the OpenStack deployment and note down the EC2 Access Key and EC2 Secret Key.

Using Horizon (the OpenStack dashboard), the EC2 access credentials can be recovered by navigating to the *Project* > *Compute* > *Access & Security* menu in the left pane of the dashboard and then selecting the *API Access* tab. The EC2 Access Key and EC2 Secret key can be revealed by pressing the *View Credentials* button located on the right side of the page.

Using the command line, the same credentials can be obtained by interrogating Keystone (the OpenStack identity manager service) using the following command:

```
$ keystone ec2-credentials-list
```

For testing the EC2 API or obtaining necessary information, it is very often useful to install the Eucalyptus client API tools (euca2ools). On a Debian / Ubuntu system, this can be done using the following command:

```
$ sudo apt-get install euca2ools
```

Before executing any commands from this package, you must first export the **EC2_URL**, **EC2_ACCESS_KEY** and **EC2_SECRET_KEY** environment variables, using the values obtained by following the instructions above.

Alternatively, OpenStack provides a script that, when sourced, automatically exports all the required environment variables. Using the Horizon dashboard, this script can be found by navigating to the *Project* > *Compute* > *Access & Security* menu in the left pane and then selecting the *API Access* tab. An archive containing this script (named `ec2rc.sh`) can be downloaded by pressing the *Download EC2 Credentials* button.

An easy way to check that euca2ools commands work is by listing all the active instances using:

```
$ euca-describe-instances
```

2.6.2 Registering your ConPaaS image to OpenStack

This section assumes that you already have created a ConPaaS services image as explained in *Creating A ConPaaS Services VM Image* and uploaded it to your OpenStack controller node. To register this image with OpenStack, you may use either Horizon or the command line client of Glance (the OpenStack image management service).

In Horizon, you can register the ConPaaS image by navigating to the *Project* > *Compute* > *Images* menu in the left pane and then pressing the *Create Image* button. In the next form, you should fill-in the image name, select *Image File* as the image source and then click the *Choose File* button and select your image (i.e. *conpaas.img*). The image format should be set to *Raw*.

Alternatively, using the command line, the ConPaaS image can be registered in the following way:

```
$ glance image-create --name <image_name> \
    --is-public true \
    --disk-format raw \
    --container-format bare \
    --file <conpaas.img>
```

In both cases, you need to obtain the AMI ID associated with the image in order to allow ConPaaS to refer to it when using the EC2 API. To do this, you need to execute the following command:

```
$ euca-describe-images
```

The AMI ID appears in the second column of the output.

2.6.3 Security Group

As in the case of Amazon Web Services deployments, OpenStack deployments use security groups to limit the the network connections allowed to an instance. The list of ports that should be opened for every instance is the same as in the case of Amazon Web Services and can be consulted here: *Security Group*.

For more details on creating and editing a security group, please refer to the OpenStack documentation available at http://docs.openstack.org/openstack-ops/content/security_groups.html.

2.7 ConPaaS on OpenNebula

ConPaaS is capable of running over an OpenNebula installation. This section describes the process of configuring OpenNebula to run ConPaaS. You can skip this section if you plan to deploy ConPaaS over Amazon Web Services or OpenStack.

2.7.1 Registering your ConPaaS image to OpenNebula

This section assumed that you already have created a ConPaaS services image as explained in *Creating A ConPaaS Services VM Image*. Upload your image (i.e. *conpaas.img*) to your OpenNebula headnode. The headnode is where OpenNebula services are running. You need have a valid OpenNebula account on the headnode (i.e. onevm list works!). Although you have a valid account on OpenNebula, you may have a problem similar to this:

/usr/lib/one/ruby/opennebula/client.rb:119:in 'initialize': ONE_AUTH file not present (RuntimeError)

You can fix it setting the ONE_AUT variable like follows:

```
$ export ONE_AUTH="/var/lib/one/.one/one_auth"
```

To register your image, you should execute *register-image-opennebula.sh* on the headnode. *register-image-opennebula.sh* needs the path to *conpaas.img* as well as OpenNebula's datastore ID and architecture Type.

To get the datastore ID, you should execute this command on the headnode:

```
$ onedatastore list
```

The output of *register-image-opennebula.sh* will be your ConPaaS OpenNebula image ID.

2.7.2 Make sure OpenNebula is properly configured

OpenNebula's OCCI daemon is used by ConPaaS to communicate with your OpenNebula cluster. The OCCI daemon is included in OpenNebula only up to version 4.6 (inclusive), so later versions of OpenNebula are not officially supported at the moment.

1. The OCCI server should be configured to listen on the correct interface so that it can receive connections from the managers located on the VMs. This can be achieved by modifying the "host" IP (or FQDN - fully qualified domain name) parameter from `/etc/one/occi-server.conf` and restarting the OCCI server.

2. Ensure the OCCI server configuration file `/etc/one/occi-server.conf` contains the following lines in section instance_types:

```
    :custom:
      :template: custom.erb
```

3. At the end of the OCCI profile file `/etc/one/occi_templates/common.erb` from your OpenNebula installation, append the following lines:

```
    <% @vm_info.each('OS') do |os| %>
          <% if os.attr('TYPE', 'arch') %>
            OS = [ arch = "<%= os.attr('TYPE', 'arch').split('/').last %>" ]
          <% end %>
    <% end %>
    GRAPHICS = [type="vnc",listen="0.0.0.0"]
```

These new lines adds a number of improvements from the standard version:

- The match for `OS TYPE:arch` allows the caller to specify the architecture of the machine.
- The last line allows for using VNC to connect to the VM. This is very useful for debugging purposes and is not necessary once testing is complete.

4. Make sure you started OpenNebula's OCCI daemon:

```
    sudo occi-server start
```

Please note that, by default, OpenNebula's OCCI server performs a reverse DNS lookup for each and every request it handles. This can lead to very poor performances in case of lookup issues. It is recommended *not* to install **avahi-daemon** on the host where your OCCI server is running. If it is installed, you can remove it as follows:

```
sudo apt-get remove avahi-daemon
```

If your OCCI server still performs badly after removing **avahi-daemon**, we suggest to disable reverse lookups on your OCCI server by editing `/usr/lib/ruby/$YOUR_RUBY_VERSION/webrick/config.rb` and replacing the line:

```
:DoNotReverseLookup => nil,
```

with:

```
:DoNotReverseLookup => true,
```

2.8 ConPaaS in a Nutshell

ConPaaS in a Nutshell is an extension to the ConPaaS project which aims at providing a cloud environment and a ConPaaS installation running on it, all in a single VM, called the Nutshell. More specifically, this VM has an all-in-one OpenStack installation running on top of LXC containers, as well as a ConPaaS installation, including all of its components, already configured to work in this environment.

The Nutshell VM can be deployed on various virtual environments, not only standard clouds such as OpenNebula, OpenStack and EC2 but also on simpler virtualization tools such as VirtualBox. Therefore, it provides a great developing and testing environment for ConPaaS without the need of accessing a cloud.

2.8.1 Creating a Nutshell image

The procedure for creating a Nutshell image is very similar to the one for creating a standard customized image described in section *Creating A ConPaaS Services VM Image*. However, there are a few settings in the configuration file which need to be considered.

Most importantly, there are two flags in the **Nutshell** section of the configuration file, *nutshell* and *container* which control the kind of image that is going to be generated. Since these two flags can take either value true of false, we distinguish four cases:

1. nutshell = false, container = false: In this case a standard ConPaaS VM image is generated and the nutshell configurations are not taken into consideration. This is the default configuration which should be used when ConPaaS is deployed on a standard cloud.
2. nutshell = false, container = true: In this case the user indicates that the image that will be generated will be a LXC container image. This image is similar to a standard VM one, but it does not contain a kernel installation.
3. nutshell = true, container = false. In this case a Nutshell image is generated and a standard ConPaaS VM image will be embedded in it. This configuration should be used for deploying ConPaaS in nested standard VMs within a single VM.
4. nutshell = true, container = true. Similar to the previous case, a Nutshell image is generated but this time a container image is embedded in it instead of a VM one. Therefore, in order to generate a Nutshell based on containers, make sure to set these flags to this configuration. This is the default configuration for our distribution of the Nutshell.

Another important setting for generating the Nutshell image is also the path to a directory containing the ConPaaS tarballs (cps*.tar.gz files). The rest of the settings specify the distro and kernel versions that the Nutshell VM would have. For the moment we have tested it only for Ubuntu 12.04 with kernel 3.5.0.

In order to run the image generating script, the procedure is almost the same as for a standard image. From the create_vm diretory run:

```
$ python create-img-script.py
$ sudo ./create-img-nutshell.sh
```

Note that if the *nutshell* flag is enabled the generated script file is called *create-img-nutshell.sh*. Otherwise, the generated script file is called *create-img-conpaas.sh* as indicated previously.

2.8.2 Creating a Nutshell image for VirtualBox

As mentioned earlier the Nutshell VM can run on VirtualBox. In order to generate a Nutshell image compatible with VirtualBox, you have to set the *cloud* value to *vbox* on the **Customizable** section of the configuration file. The rest of the procedure is the same as for other clouds. The result of the image generation script would be a *nutshell.vdi* image file which can be used as a virtual hard drive when creating a new appliance on VirtualBox.

The procedure for creating a new appliance on VirtualBox is quite standard:

1. Name and OS: You choose a custom name for the appliance but use *Linux* and *Ubuntu (64 bit)* for the type and version.
2. Memory size: Since the Nutshell runs a significant number of services and also requires some memory for the containers, we suggest to choose at least 3 GB of RAM.
3. Hard drive: Select "User an existing virtual hard drive file", browse to the location of the *nutshell.vdi* file generated earlier and press create.

2.8.3 Running the Nutshell in VirtualBox

From the 1.4.1 release though, ConPaaS is shipped together with a VirtualBox appliance containing the Nutshell VM image as well.

Before running the appliance it is strongly suggested to create a host-only network on VirtualBox in case there is not already one created. To do so from the VirtualBox GUI, go to: File>Preferences>Network>Host-only Networks and click add. Then use the File>Import appliance menu to import the image in VirtualBox.

For more information regarding the usage of the Nutshell please consult the *ConPaaS in a VirtualBox Nutshell* section in the guide.

CHAPTER 3

User Guide

ConPaaS currently contains the following services:

- **Two Web hosting services** respectively specialized for hosting PHP and JSP applications;
- **MySQL** offering a multi-master replicated load-balanced database service;
- **Scalarix service** offering a scalable in-memory key-value store;
- **MapReduce service** providing the well-known high-performance computation framework;
- **TaskFarming service** high-performance batch processing;
- **Selenium service** for functional testing of web applications;
- **XtreemFS service** offering a distributed and replicated file system;
- **HTC service** providing a throughput-oriented scheduler for bags of tasks submitted on demand;
- **Generic service** allowing the execution of arbitrary applications.

ConPaaS applications can be composed of any number of services. For example, a bio-informatics application may make use of a PHP and a MySQL service to host a Web-based frontend, and link this frontend to a MapReduce backend service for conducting high-performance genomic computations on demand.

3.1 Usage overview

3.1.1 Web-based interface

Most operations in ConPaaS can be done using the ConPaaS frontend, which gives a Web-based interface to the system. The front-end allows users to register (directly with ConPaaS or through an external Identification Provider at Contrail), create services, upload code and data to the services, and configure each service.

- The Dashboard page displays the list of services currently active in the system.
- Each service comes with a separate page which allows one to configure it, upload code and data, and scale it up and down.

3.1.2 Command line interfaces

All the functionalities of the frontend are also available using a command-line interface. This allows one to script commands for ConPaaS. The command-line interface also features additional advanced functionalities, which are not available using the front-end. (The use of external Identification Provider at Contrail is not yet available from the command-line interface.)

It exists two command line clients: `cpsclient.py` and `cps-tools`.

cpsclient.py Installation and configuration: see *Installing and configuring cpsclient.py*.

Command arguments:

```
$ cpsclient.py usage
```

Available service types:

```
$ cpsclient.py available
```

Service command specific arguments:

```
$ cpsclient.py usage <service_type>
```

Create a service:

```
$ cpsclient.py create <service_type>
```

List services:

```
$ cpsclient.py list
```

cps-tools

Installation and configuration: see *Installing and configuring cps-tools*.

Command arguments:

```
$ cps-tools --help
```

Available service types:

```
$ cps-tools service get_types
$ cps-service get-types
```

Service command specific arguments:

```
$ cps-tools <service_type> --help
$ cps-<service_type> --help
```

Create a service:

```
$ cps-tools service create <service_type>
$ cps-tools <service_type> create
$ cps-<service_type> create
```

List services:

```
$ cps-tools service list
$ cps-service list
```

List applications:

```
$ cps-tools application list
$ cps-application list
```

List clouds:

```
$ cps-tools cloud list
$ cps-cloud list
```

3.1.3 Controlling services using the front-end

The ConPaaS front-end provides a simple and intuitive interface for controlling services. We discuss here the features that are common to all services, and refer to the next sections for service-specific functionality.

Create a service. Click on "create new service", then select the service you want to create. This operation starts a new "Manager" virtual machine instance. The manager is in charge of taking care of the service, but it does not host applications itself. Other instances in charge of running the actual application are called "agent" instances.

Start a service. Click on "start", this will create a new virtual machine which can host applications, depending on the type of service.

Rename the service. By default all new services are named "New service". To give a meaningful name to a service, click on this name in the service-specific page and enter a new name.

Check the list of virtual instances. A service can run using one or more virtual machine instances. The service-specific page shows the list of instances, their respective IP addresses, and the role each instance is currently having in the service. Certain services use a single role for all instances, while other services specialize different instances to take different roles. For example, the PHP Web hosting service distinguishes three roles: load balancers, web servers, and PHP servers.

Scale the service up and down. When a service is started it uses a single "agent" instance. To add more capacity, or to later reduce capacity you can vary the number of instances used by the service. Click the numbers below the list of instances to request adding or removing servers. The system reconfigures itself without any service interruption.

Stop the service. When you do not need to run the application any more, click "stop" to stop the service. This stops all instances except the manager which keeps on running.

Terminate the service. Click "terminate" to terminate the service. At this point all the state of the service manager will be lost.

3.1.4 Controlling services using the command-line interfaces

Command-line interfaces allow one to control services without using the graphical interface. The command-line interfaces also offer additional functionalities for advanced usage of the services. See *Installing and configuring cpsclient.py* to install it.

List all options of the command-line tool.

```
$ cpsclient.py help
```

Create a service.

```
$ cpsclient.py create php
```

List available services.

```
$ cpsclient.py list
```

List service-specific options.

```
# in this example the id of our service is 1
$ cpsclient.py usage 1
```

Scale the service up and down.

```
$ cpsclient.py usage 1
$ cpsclient.py add_nodes 1 1 1 0
$ cpsclient.py remove_nodes 1 1 1 0
```

3.1.5 The credit system

In Cloud computing, resources come at a cost. ConPaaS reflects this reality in the form of a credit system. Each user is given a number of credits that she can use as she wishes. One credit corresponds to one hour of execution of one virtual machine. The number of available credits is always mentioned in the top-right corner of the front-end.

Once credits are exhausted, your running instances will be stopped and you will not be able to use the system until the administrator decides to give additional credit.

Note that every service consumes credit, even if it is in “stopped” state. The reason is that stopped services still have one “manager” instance running. To stop using credits you must completely terminate your services.

3.2 Tutorial: hosting WordPress in ConPaaS

This short tutorial illustrates the way to use ConPaaS to install and host WordPress (http://www.wordpress.org), a well-known third-party Web application. WordPress is implemented in PHP using a MySQL database so we will need a PHP and a MySQL service in ConPaaS.

1. Open the ConPaaS front-end in your Web browser and log in. If necessary, create yourself a user account and make sure that you have at least 5 credits. Your credits are always shown in the top-right corner of the front-end. One credit corresponds to one hour of execution of one virtual machine instance.
2. Create a MySQL service, start it, reset its password. Copy the IP address of the master node somewhere, we will need it in step 5.
3. Create a PHP service, start it.
4. Download a WordPress tarball from http://www.wordpress.org, and expand it in your computer.
5. Copy file `wordpress/wp-config-sample.php` to `wordpress/wp-config.php` and edit the `DB_NAME`, `DB_USER`, `DB_PASSWORD` and `DB_HOST` variables to point to the database service. You can choose any database name for the `DB_NAME` variable as long as it does not contain any special character. We will reuse the same name in step 7.
6. Rebuild a tarball of the directory such that it will expand in the current directory rather than in a `wordpress` subdirectory. Upload this tarball to the PHP service, and make the new version active.
7. Connect to the database using the command proposed by the frontend. Create a database of the same name as in step 5 using command “`CREATE DATABASE databasename;`“
8. Open the page of the PHP service, and click “access application.” Your browser will display nothing because the application is not fully installed yet. Visit the same site at URL `http://xxx.yyy.zzz.ttt/wp-admin/install.php` and fill in the requested information (site name etc).
9. That’s it! The system works, and can be scaled up and down.

Note that, for this simple example, the “file upload” functionality of WordPress will not work if you scale the system up. This is because WordPress stores files in the local file system of the PHP server where the upload has been processed. If a subsequent request for this file is processed by another PHP server then the file will not be found. The solution to that issue consists in using the shared file-system service called XtreemFS to store the uploaded files.

3.3 The PHP Web hosting service

The PHP Web hosting service is dedicated to hosting Web applications written in PHP. It can also host static Web content.

3.3.1 Uploading application code

PHP applications can be uploaded as an archive or via the Git version control system.

Archives can be either in the `tar`, `zip`, `gzip` or `bzip2` format.

Warning: the archive must expand **in the current directory** rather than in a subdirectory.

The service does not immediately use new applications when they are uploaded. The frontend shows the list of versions that have been uploaded; choose one version and click “set active” to activate it.

Note that the frontend only allows uploading archives smaller than a certain size. To upload large archives, you must use the command-line tools or Git.

The following example illustrates how to upload an archive to the service with id 1 using the `cpsclient.py` command line tool:

```
$ cpsclient.py upload_code 1 path/to/archive.zip
```

To enable Git-based code uploads you first need to upload your SSH public key. This can be done either using the command line tool:

```
$ cpsclient.py upload_key serviceid filename
```

An SSH public key can also be uploaded using the ConPaaS frontend by choosing the “checking out repository” option in the “Code management” section of your PHP service. Once the key is uploaded the frontend will show the `git` command to be executed in order to obtain a copy of the repository. The repository itself can then be used as usual. A new version of your application can be uploaded with `git push`.

```
user@host:~/code$ git add index.php
user@host:~/code$ git commit -am "New index.php version"
user@host:~/code$ git push origin master
```

3.3.2 Access the application

The frontend gives a link to the running application. This URL will remain valid as long as you do not stop the service.

3.3.3 Using PHP sessions

PHP normally stores session state in its main memory. When scaling up the PHP service, this creates problems because multiple PHP servers running in different VM instances cannot share their memory. To support PHP sessions the PHP service features a key-value store where session states can be transparently stored. To overwrite PHP session functions such that they make use of the shared key-value store, the PHP service includes a standard “phpsession.php” file at the beginning of every .php file of your application that uses sessions, i.e. in which function *session_start()* is encountered. This file overwrites the session handlers using the *session_set_save_handler()* function.

This modification is transparent to your application so no particular action is necessary to use PHP sessions in ConPaaS.

3.3.4 Debug mode

By default the PHP service does not display anything in case PHP errors occur while executing the application. This setting is useful for production, when you do not want to reveal internal information to external users. While developing an application it is however useful to let PHP display errors.

```
$ cpsclient.py toggle_debug serviceid
```

3.3.5 Adding and removing nodes

Like all ConPaaS services, the PHP service is elastic: service owner can add or remove nodes. The PHP service (like the Java service) belongs to a class of web services that deals with three types of nodes:

proxy a node that is used as an entry point for the web application and as a load balancer

web a node that deals with static pages only

backend a node that deals with PHP requests only

When a proxy node receives a request, it redirects it to a web node if it is a request for a static page, or a backend node if it is a request for a PHP page.

If your PHP service has a slow response time, increase the number of backend nodes.

On command line, you can use `cpsclient.py` to add nodes. The `add_nodes` sub-command takes 4 arguments in that order: the PHP service identifier, the number of backend nodes, the number of web nodes and the number of proxy nodes to add. It also take a 5th optional argument that specify in which cloud nodes will be created. For example, adding two backend nodes to PHP service id 1:

```
$ cpsclient.py add_nodes 1 2 0 0
```

Adding one backend node and one web node in a cloud provider called `mycloud`:

```
$ cpsclient.py add_nodes 1 1 1 0 mycloud
```

You can also remove nodes using `cpsclient.py`. For example, the following command will remove one backend node:

```
$ cpsclient.py remove_nodes 1 1 0 0
```

Warning: Initially, an instance of each node is running on one single VM. Then, when adding a backend node, ConPaaS will move the backend node running on the first VM to a new VM. So, actually, it will *not* add a new backend node the first time. Requesting for one more backend node will create a new VM that will run an additional backend.

3.3.6 Autoscaling

One of the worries of a service owner is the trade-off between the performance of the service and the cost of running it. The service owner can add nodes to improve the performance of the service, which will have more nodes to balance the load, or remove nodes from the service to decrease the cost per hour, but increase the load per node.

Adding and removing nodes as described above is interactive: the service owner has to run a command line or push some buttons on the web frontend GUI. However, the service owner is not always watching for the performance of his Web service.

Autoscaling for the PHP service will add or remove nodes according to the load on the Web service. If the load on nodes running a Web service exceeds a given threshold and the autoscaling mechanism estimates that it will last, then the autoscaling mechanism will automatically add nodes for the service to balance the load. If the load on nodes running a Web service is low and the autoscaling mechanism estimates that it will last and that removing some nodes will not increase the load on nodes beyond the given threshold, then the autoscaling mechanism will automatically remove nodes from the service to decrease the cost per hour of the service.

Autoscaling for the PHP service will also take into account the different kind of nodes that the cloud providers propose. They usually propose small instances, middle range instances and large instances. So, the autoscaling mechanism will select different kind of nodes depending on the service owner strategy choice.

To enable autoscaling for the PHP service, run the command:

```
$ cpsclient.py on_autoscaling <sid> <adapt_interval> <response_time_threshold> <strategy>
```

where:

- <sid> is the service identifier
- <adapt_interval> is the time in minutes between automatic adaptation point
- <response_time_threshold> is the desired response time in milliseconds
- <strategy> is the policy used to select instance type when adding nodes, it must be one of:
  - "low": will always select the smallest (and cheapest) instance proposed by the cloud provider

  - "medium_down"
  - "medium"
  - "medium_up"
  - "high"

For example:

```
$ cpsclient.py on_autoscaling 1 5 2000 low
```

enables autoscaling for PHP service 1, with an adaptation every 5 minutes, a response time threshold of 2000 milliseconds (2 seconds), and using the strategy low. This means that every 5 minutes, autoscaling will determine if it will add nodes, remove nodes, or do nothing, by looking at the history of the Web service response time and comparing it to the desired 2000 milliseconds. According the specified "low" strategy, if it decides to create nodes, it will always select the smallest instance from the cloud provider.

Any time, the service owner may re-run the "on_autoscaling" command to tune autoscaling with different parameters:

```
$ cpsclient.py on_autoscaling 1 10 1500 low
```

this command updates the previous call to "on_autoscaling" and changes the adaptation interval to 10 minutes, and setting a lower threshold to 15000 milliseconds.

Autoscaling may be disabled by running command:

```
$ cpsclient.py off_autoscaling <sid>
```

3.4 The Java Web hosting service

The Java Web hosting service is dedicated to hosting Web applications written in Java using JSP or servlets. It can also host static Web content.

3.4.1 Uploading application code

Applications in the Java Web hosting service can be uploaded in the form of a `war` file or via the Git version control system. The service does not immediately use new applications when they are uploaded. The frontend shows the list of versions that have been uploaded; choose one version and click "set active" to activate it.

Note that the frontend only allows uploading archives smaller than a certain size. To upload large archives, you must use the command-line tools or Git.

The following example illustrates how to upload an archive with the `cpsclient.py` command line tool:

```
$ cpsclient.py upload_code serviceid archivename
```

To upload new versions of your application via Git, please refer to section *Uploading application code*.

3.4.2 Access the application

The frontend gives a link to the running application. This URL will remain valid as long as you do not stop the service.

3.5 The MySQL Database Service

The MySQL service is a true multi-master database cluster based on MySQL-5.5 and the Galera synchronous replication system. It is an easy-to-use, high-availability solution, which provides high system uptime, no data loss and scalability for future growth. It provides exactly the same look and feel as a regular MySQL database.

Summarizing, its advanced features are:

- Synchronous replication
- Active-active multi-master topology
- Read and write to any cluster node
- Automatic membership control, failed nodes drop from the cluster
- Automatic node joining
- True parallel replication, on row level
- Both read and write scalability
- Direct client connections, native MySQL look & feel

3.5.1 The Database Nodes and Load Balancer Nodes

The MySQL service offers the capability to instantiate multiple instances of database nodes, which can be used to increase the throughput and to improve features of fault tolerance through replication. The multi-master structure allows any database node to process incoming updates, because the replication system is responsible for propagating the data modifications made by each member to the rest of the group and resolving any conflicts that might arise between concurrent changes made by different members. These features can be used to increase the throughput of the cluster.

To obtain the better performances from a cluster, it is a best practice to use it in balanced fashion, so that each node has approximatively the same load of the others. To achieve this, the service allows users to allocate special load balancer nodes (`glb_nodes`) which implement load balancing. Load balancer nodes are designed to receive all incoming database queries and automatically schedule them between the database nodes, making sure they all process equivalent workload.

3.5.2 Resetting the User Password

When a MySQL service is started, a new user “`mysqldb`” is created with a randomly-generated password. To gain access to the database you must first reset this password. Click “Reset Password” in the front-end, and choose the new password.

Note that the user password is not kept by the ConPaaS frontend. If you forget the password the only thing you can do is reset the password again to a new value.

3.5.3 Accessing the database

The frontend provides the command-line to access the database cluster. Copy-paste this command in a terminal. You will be asked for the user password, after which you can use the database as you wish. Note that, in case the service has instantiated a load balancer, the command refers to the load balancer IP and its specific port, so the load balancer can receive all the queries and distributes them across the ordinary nodes. Note, again, that the *mysqldb* user has extended privileges. It can create new databases, new users etc.

3.5.4 Uploading a Database Dump

The ConPaaS frontend allows users to easily upload database dumps to a MySQL service. Note that this functionality is restricted to dumps of a relatively small size. To upload larger dumps you can always use the regular mysql command for this:

```
$ mysql mysql-ip-address -u mysqldb -p < dumpfile.sql
```

3.5.5 Performance Monitoring

The MySQL service interface provides a sophisticated mechanism to monitor the service. The user interface, in the frontend, shows a monitoring control, called "Performance Monitor", that can be used to monitor a large cluster's behaviour. It interacts with "Ganglia", "Galera" and "MySQL" to obtain various kinds of information. Thus, "Performance Monitor" provides a solution for maintaining control and visibility of all nodes, with a monitoring dynamic data every few seconds.

It consists of three main components.

- "Cluster usage" monitors the number of incoming SQL queries. This will let you know in advance about any overload of the resources. You will also be able to spot usage trends over time so as to get insights on when you need to add new nodes, serving the MySQL database.
- The second control highlights the cluster's performance, with a table detailing the load, memory usage, CPU utilization, and network traffic for each node of the cluster. Users can use these informations in order to detect problems in their applications. The table displays the resource utilization across all nodes, and highlight the parameters which suggest an abnormality. For example if CPU utilization is high, or free memory is very low this is shown clearly. This may mean that processes on this node will start to slow down, and that it may be time to add additional nodes to the cluster. On the other hand this may indicate a malfunction of the specific node.

  In this latter case, in a multimaster system, it may be a good idea to kill the node and replace it with another one. The monitoring system also simplifies this kind of operations through buttons which allow to directly kill a specific node. Keep in mind, however, that high CPU utilization may not necessarily affect application performance.
- "Galera Mean Misalignment" draws a real-time measure of the mean misalignment across the nodes. This information is derived by Galera metrics about the average length of the receive queue since the most recent status query. If this value is noticeably larger than zero, the nodes are likely to be overloaded, and cannot apply the writesets as quickly as they arrive, resulting in replication throttling.

3.6 The Scalarix key-value store service

The Scalarix service provides an in-memory key-value store. It is highly scalable and fault-tolerant. This service deviates slightly from the organization of other services in that it does not have a separate manager virtual machine instance. Scalarix is fully symmetric so any Scalarix node can act as a service manager.

3.6.1 Accessing the key-value store

Clients of the Scalarix service need the IP address of (at least) one node to connect to the service. Copy-paste the address of any of the running instances in the client. A good choice is the first instance in the list: when scaling the service up and down, other instances may be created or removed. The first instance will however remain across these reconfigurations, until the service is terminated.

3.6.2 Managing the key-value store

Scalarix provides its own Web-based interface to monitor the state and performance of the key-value store, manually add or query key-value pairs, etc. For convenience reasons the ConPaaS front-end provides a link to this interface.

3.7 The MapReduce service

The MapReduce service provides the well-known Apache Hadoop framework in ConPaaS. Once the MapReduce service is created and started, the front-end provides useful links to the Hadoop namenode, the job tracker, and to a graphical interface which allows to upload/download data to/from the service and issue MapReduce jobs.

Warning: This service requires virtual machines with **at least** 384 MB of RAM to function properly.

3.8 The TaskFarming service

The TaskFarming service provides a bag of tasks scheduler for ConPaaS. The user needs to provide a list of independent tasks to be executed on the cloud and a file system location where the tasks can read input data and/or write output data to it. The service first enters a sampling phase, where its agents sample the runtime of the given tasks on different cloud instances. The service then based on the sampled runtimes, provides the user with a list of schedules. Schedules are presented in a graph and the user can choose between cost/makespan of different schedules for the given set of tasks. After the choice is made, the service enters the execution phase and completes the execution of the rest of the tasks according to the user's choice.

3.8.1 Preparing the ConPaaS services image

By default, the TaskFarming service can execute the user code that is supported by the default ConPaaS services image. If user's tasks depend on specific libraries and/or applications that do not ship with the default ConPaaS services image, the user needs to configure the ConPaaS services image accordingly and use the customized image ID in ConPaaS configuration files.

3.8.2 The bag of tasks file

The bag of tasks file is a simple plain text file that contains the list of tasks along with their arguments to be executed. The tasks are separated by new lines. This file needs to be uploaded to the service, before the service can start sampling. Below is an example of a simple bag of tasks file containing three tasks:

```
/bin/sleep 1 && echo "slept for 1 seconds" >> /mnt/xtreemfs/log
/bin/sleep 2 && echo "slept for 2 seconds" >> /mnt/xtreemfs/log
/bin/sleep 3 && echo "slept for 3 seconds" >> /mnt/xtreemfs/log
```

The minimum number of tasks required by the service to start sampling is depending on the number of tasks itself, but a bag with more than thirty tasks is large enough.

3.8.3 The filesystem location

The TaskFarming service uses XtreemFS for data input/output. The actual task code can also reside in the XtreemFS. The user can optionally provide an XtreemFS location which is then mounted on TaskFarming agents.

3.8.4 The demo mode

With large bags of tasks and/or with long running tasks, the TaskFarming service can take a long time to execute the given bag. The service provides its users with a progress bar and reports the amount of money spent so far. The TaskFarming service also provides a “demo” mode where the users can try the service with custom bags without spending time and money.

3.9 The XtreemFS service

The XtreemFS service provides POSIX compatible storage for ConPaaS. Users can create volumes that can be mounted remotely or used by other ConPaaS services, or inside applications. An XtreemFS instance consists of multiple DIR, MRC and OSD servers. The OSDs contain the actual storage, while the DIR is a directory service and the MRC contains meta data. By default, one instance of each runs inside the first agent virtual machine and the service can be scaled up and down by adding and removing additional OSD nodes. The XtreemFS documentation can be found at http://xtreemfs.org/userguide.php.

3.9.1 SSL Certificates

The XtreemFS service uses SSL certificates for authorization and authentication. There are two types of certificates, user-certificates and client-certificates. Both certificates can additionally be flagged as administrator certificates which allows performing administrative file-systems tasks when using them to access XtreemFS. Certificates are only valid for the service that was used to create them. The generated certificates are in P12-format.

The difference between client- and user-certificates is how POSIX users and groups are handled when accessing volumes and their content. Client-certificates take the user and group with whom an XtreemFS command is called, or a mounted XtreemFS volume is accessed. So multiple users might share a single client-certificate. On the other hand, user-certificates contain a user and group inside the certificate. So usually, each user has her personal user-certificate. Both kinds of certificate can be used in parallel. Client-certificates are less secure, since the user and group with whom files are accessed can be arbitrarily changed if the mounting user has local superuser rights. So client-certificates should only be used in trusted environments.

Using the command line client, certificates can be created like this, where <adminflag> can be “true”, “yes”, or “1” to grant administrator rights:

```
$ cpsclient.py get_client_cert <service-id> <passphrase> <adminflag> <filename.p12>
$ cpsclient.py get_user_cert <service-id> <user> <group> <passphrase> <adminflag> <filename.p12>
```

3.9.2 Accessing volumes directly

Once a volume has been created, it can be directly mounted on a remote site by using the mount.xtreemfs command. A mounted volume can be used like any local POSIX-compatible filesystem. You need a certificate for mounting (see last section). The command looks like this, where <address> is the IP of an agent running an XtreemFS directory service (usually the first agent):

```
$ mount.xtreemfs <address>/<volume> <mount-point> --pkcs12-file-path <filename.p12> --pkcs12-passp
```

The volume can be unmounted with the following command:

```
$ fusermount -u <mount-point>
```

Please refer to the XtreemFS user guide (http://xtreemfs.org/userguide.php) for further details.

3.9.3 Policies

Different aspects of XtreemFS (e.g. replica- and OSD-selection) can be customised by setting certain policies. Those policies can be set via the ConPaaS command line client (recommended) or directly via xtfsutil (see the

XtreemFS user guide). The commands are like follows, were <policy_type> is “osd_sel”, “replica_sel”, or “replication”:

```
$ cpsclient.py list_policies <service-id> <policy_type>
$ cpsclient.py set_policy <service-id> <policy_type> <volume> <policy> [factor]
```

3.9.4 Persistency

If the XtreemFS service is shut down, all its data is permanently lost. If persistency beyond the service runtime is needed, the XtreemFS service can be moved into a snapshot by using the download_manifest operation of the command line client.

Warning: This operation will automatically shut down the service and its application.

The whole application containing the service and all of its stored volumes with their data can be moved back into a running ConPaaS application by using the manifest operation.

The commands are:

```
$ cpsclient.py download_manifest <application-id> > <filename>
$ cpsclient.py manifest <filename>
```

3.9.5 Important notes

When a service is scaled down by removing OSDs, the data of those OSDs is migrated to the remaining OSDs. Always make sure there is enough free space for this operation to succeed. Otherwise you risk data loss. The download_manifest operation of the XtreemFS service will also shut the service down. This behaviour might differ from other ConPaaS services, but is necessary to avoid copying the whole filesystem (which would be a very expensive operation). This might change in future releases.

3.10 The HTC service

The HTC service provides a throughput-oriented scheduler for bags of tasks submitted on demand for ConPaaS. An initial bag of tasks is sampled generating a throughput = f(cost) function. The user is allowed at any point, including upon new tasks submission, to request the latest throughput = f(cost) function and insert his target throughput. After the first bag is sampled and submitted for execution the user is allowed to add tasks to the job with the corresponding identifier. The user is allowed at any point, including upon new tasks submission, to request the latest throughput = f(cost) function and adjust his target throughput. All tasks that are added are immediately submitted for execution using the latest configuration requested by the user, corresponding to the target throughput.

3.10.1 Available commands

`start service_id`: prompts the user to specify a mode (’real’ or ’demo’) and type (’batch’, ’online’ or ’workflow’) for the service. Starts the service under the selected context and initializes all the internal data structures for running the service.

`stop service_id`: stops and releases all running VMs that exist in the pool of workers regardless of the tasks running.

`terminate service_id`: stops and releases the manager VM along with the running algorithm and existing data structures.

`create_worker service_id type count`: adds count workers to the pool returns the worker_ids. The worker is added to the table. The manager starts the worker on a VM requested of the selected type.

`remove_worker service_id worker_id`: removes a worker from the condor pool. The worker_id is removed from the table.

`create_job service_id .bot_file`: creates a new job on the manager and returns a job_id. It uploads the .bot_file on the manager and assign a queue to the job which will contain the path of all .bot_files submitted to this job_id.

`sample service_id job_id`: samples the job on all available machine types in the cloud according to the HTC model.

`throughput service_id`: prompts the user to select a target throughput within [0,TMAX] and returns the cost for that throughput.

`configuration service_id`: prompts the user to select a target throughput within [0,TMAX] and returns the machine configuration required for that throughput. At this point the user can manually create the pool of workers using create_worker and remove_worker.

`select service_id`: prompts the user to select a target throughput within [0,TMAX] and creates the pool of workers needed to obtain that throughput.

`submit service_id job_id`: submits all the bags in this job_id for execution with the current configuration of workers.

`add service_id job_id .bot_file`: submits a .bot_file for execution on demand. The bag is executed with the existing configuration.

3.11 The Generic service

The Generic service facilitates the deployment of arbitrary server-side applications in the cloud. A Generic service may contain multiple Generic agents, each of them running an instance of the application.

The users can control the application's life cycle by installing or removing code versions, running or interrupting the execution of the application or checking the status of each of the Generic agents. New Generic agents can be added or old ones removed at any time, based on the needs of the application. Moreover, additional storage volumes can be attached to agents if additional storage space is needed.

To package an application for the Generic service, the user has to provide simple scripts that guide the process of installing, running, scaling up and down, interrupting or removing an application to/form a Generic agent.

3.11.1 Agent roles

Generic agents assume two roles: the first agent started is always a "master" and all the other agents assume the role of regular "nodes". This distinction is purely informational: there is no real difference between the two agent types, both run the same version of the application's code and are treated by the ConPaaS system in exactly the same way. This distinction may be useful, however, when implementing some distributed algorithms in which one node must assume a specific role, such as leader or coordinator.

It is guaranteed that, as long as the Generic service is running, there will always be exactly one agent with the "master" role and the same agent will assume this role until the Generic service is stopped. Adding or removing nodes will only affect the number of regular nodes.

3.11.2 Packaging an application

To package an application for the Generic service, one needs to write various scripts which are automatically called inside agents whenever the corresponding events happen. The following scripts may be used:

`init.sh` – called whenever a new code version is activated. The script is automatically called for each agent as soon as the corresponding code version becomes active. The script should contain commands that initialize the environment and prepare it for the execution of the application. It is guaranteed that this script is is called before any other scripts in a specific code version.

`notify.sh` – called whenever a new agent is added or removed. The script is automatically called whenever a new agent is added and becomes active or is removed from the Generic service. The script may configure the application to take into account the addition or removal of a specific node or group of nodes. In order to retrieve the updated list of nodes along with their IP addresses, the script may check the content of the following file, which always contains the current list of nodes in JSON format: `/var/cache/cpsagent/agents.json`. Note that when multiple nodes are added or removed in a single operation, the script will be called only once for each of the remaining nodes.

`run.sh` – called whenever the user requests to start the application. The script should start executing the application and after the execution completes, it may return an error code that will be shown to the user. It is guaranteed that the `init.sh` script already finished execution before `run.sh` is called.

`interrupt.sh` – called whenever the user requests that the application is interrupted. The script should notify the application that the interruption was requested and allow it to gracefully terminate execution. It is guaranteed that `interrupt.sh` is only called when the application is actually running.

`cleanup.sh` – called whenever the user requests that the application's code is removed from the agent. The script should remove any files that the application generated during execution and are not longer needed. After the script completes execution, a new version of the code may be activated and the `init.sh` script called again, so the agent needs to be reverted to a clean state.

To create an application's package, all the previous scripts must be added to an archive in the `tar`, `zip`, `gzip` or `bzip2` format. If there is no need to execute any tasks when a specific type of event happens, some of the previous scripts may be left empty or may even be missing completely from the application's archive.

Warning: the archive must expand **in the current directory** rather than in a subdirectory.

The application's binaries can be included in the archive only if they are small enough.

Warning: the archive is stored on the service manager instance and its contents are extracted in each agent's root file system which usually has a very limited amount of free space (usually a little more than 100 MB), so application's binaries can be included only if they are really small (a few MBs).

A better idea would be to attach an additional storage volume where the `init.sh` script can download the application's binaries from an external location for each Generic agent. This will render the archive very small as it only contains a few scripts. This is the recommended approach.

3.11.3 Uploading the archive

An application's package can be uploaded to the Generic service either as an archive or via the Git version control system. Either way, the code does not immediately become active and must be activated first.

Using the web frontend, the "Code management" section offers the possibility to upload a new archive to the Generic service. After the upload succeeds, the interface shows the list of versions that have been uploaded; choose one version and click "set active" to activate it. Note that the frontend only allows uploading archives smaller than a certain size. To upload large archives, you must use the command-line tools or Git. The web frontend also allows downloading or deleting a specific code version. Note that the active code version cannot be deleted.

Using the command-line interface, uploading and enabling a new code version is just as simple. The following example illustrates how to upload and activate an archive to the service with id 1 using the `cpsclient.py` command line tool:

```
$ cpsclient.py upload_code 1 test-code.tar.gz
Code version code-pw1LKs uploaded
$ cpsclient.py enable_code 1 code-pw1LKs
code-pw1LKs enabled
$ cpsclient.py list_uploads 1
current codeVersionId filename         description
-----------------------------------------------------
```

```
      * code-pw1LKs   test-code.tar.gz
        code-default  code-default.tar Initial version
```

To download a specific code version, the following command may be used:

```
$ cpsclient.py download_code <serviceid> <code-version>
```

The archive will be downloaded using the original name in the current directory.

Warning: if another file with the same name is present in the current directory, it will be overwritten.

The command-line client also allows deleting a code version, with the exception of the currently active version:

```
$ cpsclient.py delete_code <serviceid> <code-version>
```

It is a good idea to delete the code versions which are not needed anymore, as all the available code versions are stored in the Generic manager's file system, which has a very limited amount of available space. In contrast to the manager, the agents only store the active code version, which is replaced every time a new version becomes active.

3.11.4 Uploading the code using git

As an alternative to uploading the application's package as stated above, the Generic service also supports uploading the package's content using Git.

To enable Git-based code uploads, you first need to upload your SSH public key. This can be done either using the web frontend, in the "Code management" section, after selecting "checking out repository" or using the command-line client:

```
$ cpsclient.py upload_key <serviceid> <filename>
```

You can check that the key was successfully uploaded by listing the trusted SSH keys:

> $ cpsclient.py list_keys <serviceid>

Once the key is uploaded, the following command has to be executed in order to obtain a copy of the repository:

```
$ git clone git@<generic-manager-ip>:code
```

The repository itself can then be used as usual. A new version of your application can be uploaded with `git push`:

```
$ cd code
$ git add {init,notify,run,interrupt,cleanup}.sh
$ git commit -m "New code version"
$ git push origin master
```

The `git push` command will trigger the updating of the available code versions. To activate the new code version, the same procedure as before must be followed. Note that, when using the web frontend, you may need to refresh the page in order to see the new code version.

To download a code version uploaded using Git, you must clone the repository and checkout a specific commit. The version number represents the first part of the commit hash, so you can use that as a parameter for the `git checkout` command:

```
$ cpsclient.py list_uploads 1
current codeVersionId filename            description
---------------------------------------------------------
        git-7235de9   7235de9             Git upload
      * code-default  code-default.tar    Initial version
$ git clone git@192.168.56.10:code
$ cd code
$ git checkout 7235de9
```

Deleting a specific code version uploaded using Git is not possible.

3.11.5 Managing storage volumes

Storage volumes of arbitrary size can be attached to any Generic agent. Note that, for some clouds such as Amazon EC2 and OpenStack, the volume size must be a multiple of 1 GB. In this case, if the requested size does not satisfy this constraint, it will be rounded up to the smallest size multiple of 1 GB that is greater than the requested size.

The attach or detach operations are permitted only if there are no scripts running inside the agents. This guarantees that a volume is never in use when it is detached.

To create and attach a storage volume using the web frontend, you must click the "+ add volume" link below the instance name of the agent that should have this volume attached to. A small form will expand where you can enter the volume name and the requested size. Note that the volume name must be unique, or else the volume will not be created. The volume is created and attached after pressing the "create volume" button. Depending on the cloud in use and the volume size, this operation may take a little while. Additional volumes can be attached later to the same agent if more storage space is needed.

The list of volumes attached to a specific agent is shown in the instance view of the agent, right under the instance name. For each volume, the name of the volume and the requested size is shown. To detach and delete a volume, you can press the red X icon after the volume's size.

Warning: after a volume is detached, all data contained within it is lost forever.

Using the command-line client, a volume can be created and attached to a specific agent with the following command:

```
$ cpsclient.py create_volume <serviceid> <vol_name> <size> <agent_id>
```

Size must always be specified in MB. To find out the *agent_id* of a specific instance, you may issue the following command:

```
$ cpsclient.py list_nodes <serviceid>
```

The list of all storage volumes can be retrieved with:

```
$ cpsclient.py list_volumes <serviceid>
```

This command detaches and deletes a storage volume:

```
$ cpsclient.py delete_volume <serviceid> <agent_id>
```

3.11.6 Controlling the application's life cycle

A newly started Generic service contains only one agent with the role "master". As in the case of other ConPaaS services, nodes can be added to the service (or removed from the service) at any point in time.

In the web frontend, new Generic nodes can be added by entering the number of new nodes (in a small cell below the list of instances) and pressing the "submit" button. Entering a negative number of nodes will lead to the removal of the specified number of nodes.

On the command-line, nodes can be added with the following command:

```
$ cpsclient.py add_nodes <serviceid> <number_of_nodes>
```

Immediately after the new nodes are ready, the active code version is copied to the new nodes and the `init.sh` script is executed in each of the new nodes. All the other nodes which were already up before the execution of the command will be notified about the addition of the new nodes to the service, so `notify.sh` is executed in their case. The `init.sh` script is never executed twice for the same agent and the same code version.

Nodes can be removed with:

```
$ cpsclient.py remove_nodes <serviceid> <number_of_nodes>
```

After the command completes and the specified number of nodes are terminated, the `notify.sh` script is executed for all the remaining nodes to notify them of the change.

The Generic service also offers an easy way to run the application on every agent, interrupt a running application or cleanup the agents after the execution is completed.

In the web frontend, the `run`, `interrupt` and `cleanup` buttons are conveniently located on the top of the page, above the instances view. Pressing such a button will execute the corresponding script in all the agents. Above the buttons there is also a parameters field which allow the user to specify parameters which will be forwarded to the script during the execution.

On the command line, the following commands may be used:

```
$ cpsclient.py run <serviceid> [parameters]
$ cpsclient.py interrupt <serviceid> [parameters]
$ cpsclient.py cleanup <serviceid> [parameters]
```

The parameters are optional and, if not present, will be replaced by an empty list.

The `run` and `cleanup` commands cannot be issued if any scripts are still running inside at least one agent. In this case, if it is not desired to wait for them to complete execution, `interrupt` may be called first.

In turn, `interrupt` cannot be called if no scripts are running (there is nothing to interrupt). The `interrupt` command will execute the `interrupt.sh` script that tries to cleanly shut down the application. If the `interrupt.sh` completes execution and the application is still running, the application will be automatically killed. When `interrupt.sh` itself has to be killed, the `interrupt` command can be issued again. In this case, it will kill all the running scripts (including `interrupt.sh`). In the web frontend, this is highlighted by renaiming the `interrupt` button to `kill`.

Warning: issuing the `interrupt` command twice kills all the running scripts, including the child processes started by them!

Enabling a new code version is allowed only when no script from the current code version is currently running. If it is not desired to wait for them to complete execution, `interrupt` may be called first. When enabling a new code version, immediately after copying the new code to the agents, the new `init.sh` script is called.

3.11.7 Checking the status of the agents

The running status of the various scripts for each agent can easily be checked in both the web frontend and using the command-line interface.

In the web frontend, the instance view of each agent contains a table with the 5 scripts and each script's running status, along with a led that codes the status using colors: *light blue* when the current version of the script was never executed, *blinking green* when the script is currently running and *red* when the script finished execution. In the latter case, hovering the mouse pointer over the led will indicate the return code in a tool-tip text.

With the command-line interface, the status of the scripts for each agent can be listed using the following command:

```
$ cpsclient.py get_script_status <serviceid>
```

The Generic service also facilitates retrieving the agent's log file and the contents of standard output and error streams. In the web frontend, three links are present in the instance's view of each agent. Using the command line, the logs can be retrieved with the following command:

```
$ cpsclient.py get_script_status <serviceid> <agent_id>
```

To find out the agent_id of a specific instance, you may issue the following command:

```
$ cpsclient.py list_nodes <serviceid>
```

3.12 ConPaaS in a VirtualBox Nutshell

ConPaaS in a Nutshell is a version of ConPaaS which runs inside a single VirtualBox VM. It is the recommended way to test the system and/or to run it in a single physical machine.

3.12.1 Starting the Nutshell

The easiest way to start the Nutshell is using VirtualBox:

1. If you haven't done this already, create a host-only network on VirtualBox. To do so from the VirtualBox GUI, go to: File>Preferences>Network>Host-only Networks and click add. If you already see a host-only network (probably called *vboxnet0*), then you do not need to add another one.
2. Import the Nutshell appliance using the menu File->Import Appliance, or by simply double-clicking on the file in your file manager.
3. Once the Nutshell has been imported, you may adjust the amount of memory and the number of CPUs you want to dedicate to it by clicking on the Nutshell, then Settings->System->Motherboard/Processor. We recommend allocating at least 3 GB of RAM for the Nutshell to function properly.
4. Start the Nutshell by clicking "Start".
5. Once the Nutshell is started, you can log into it. Wait a few seconds until you see a login prompt. The login credentials are:

```
Username: stack
Password: contrail
```

6. One important piece of information which you may want to note down is the IP address assigned to the Nutshell VM. This can be used to access the web frontend directly from your machine or to SSH into the Nutshell VM in order to execute command-line interface commands or to copy files. To find it, type the following command:

```
$ ifconfig br200
```

   The IP address will appear in the second line of text.

3.12.2 Using the Nutshell via the graphical frontend

You can access the ConPaaS frontend by inserting the IP address of the Nutshell VM in your Web browser, **making sure to add https:// in front of it**:

```
https://192.168.56.xxx
```

Note that the frontend is accessible only from your local machine. Other machines will not be able to access it. A default user is available for you, its credentials are:

```
ConPaaS
Username: test
Password: password
```

You can now use the frontend in the same way as any ConPaaS system, creating applications, services etc. Note that the services are also only accessible from your local machine.

Note that also *Horizon* (the Openstack dashboard) is running on it as well. In case you are curious and you want to look inside the system, Horizon can be reached (using HTTP, not HTTPS) at the same IP address:

```
http://192.168.56.xxx
```

The credentials for Horizon are:

```
Openstack
Username: admin
Password: password
```

3.12.3 Using the Nutshell via the command-line interface

You can also use the command-line to control your Nutshell installation. You need to log in as the *stack* user directly in the VirtualBox window or using SSH to connect to the Nutshell VM's IP address.

On login, both the ConPaaS and OpenStack users will already be authenticated. You should be able to execute ConPaaS commands, for example starting a *helloworld* service can be done with:

```
$ cpsclient.py create helloworld
```

or:

```
$ cps-tools service create helloworld
```

OpenStack commands are also available. For example:

```
$ nova list
```

lists all the active instances and:

```
$ cinder list
```

lists all the existing volumes.

The Nutshell contains a *Devstack* installation of Openstack, therefore different services run and log on different tabs of a *screen* session. In order to stop, start or consult the logs of these services, connect to the screen session by executing:

```
$ /opt/stack/devstack/rejoin-stack.sh
```

Every tab in the screen session is labeled with the name of the service it belongs to. For more information on how to navigate between tabs and scroll up and down the logs, please consult the manual page for the *screen* command.

3.12.4 Changing the IP address space used by the Nutshell

The Nutshell VM uses an IP address assigned by the DHCP server of the host-only network of VirtualBox. In the default settings, the DHCP server uses a range from `192.168.56.101` to `192.168.56.254`. If you want to change this IP range, you can go to: File>Preferences>Network>Host-only Networks, select *vboxnet0* and click the edit button and then "DHCP server".

Note that ConPaaS services running inside the Nutshell VM also need to have IP addresses assigned. This is done using OpenStack's floating IP mechanism. The default configuration uses an IP range from `192.168.56.10` to `192.168.56.99`, which does not overlap with the default one used by the DHCP server of the host-only network in VirtualBox. If you want to modify this IP range, execute the following commands on the Nutshell as the *stack* user:

```
$ nova floating-ip-bulk-delete 192.168.56.0/25
$ nova floating-ip-bulk-create --pool public --interface br200 <new_range>
```

The first command removes the default IP range for floating IPs and the second adds the new range. After executing these two commands, do not forget to restart the Nutshell so the changes take effect:

```
$ sudo reboot
```

3.12.5 Using the Nutshell to host a publicly accessible ConPaaS installation

The Nutshell can also be configured to host services which are accessible from the public Internet. In this case, the floating IP pool in use by OpenStack needs to be configured with an IP range that contains public IP addresses. The procedure for using such an IP range is the same as the one described above. Care must be taken so that these public IP addresses are not in use by other machines in the network and routing for this range is correctly implemented.

If the ConPaaS frontend itself needs to be publicly accessible, the host-only network of VirtualBox can be replaced with a bridged network connected to a physical network interface that provides Internet access. Note that this bridge network must use a DHCP server that assigns a public IP address to the Nutshell or, alternatively, the Nutshell can be configured to use a static IP address (for example by editing the file `/etc/network/interfaces`). If the Nutshell is publicly accessible, you may want to make sure that tighter security is implemented: the default user for the ConPaaS frontend is removed and access to SSH and OpenStack dashboard is blocked.

CHAPTER 4

Internals

4.1 Introduction

A ConPaaS service may consist of three main entities: the manager, the agent and the frontend. The (primary) manager resides in the first VM that is started by the frontend when the service is created and its role is to manage the service by providing supporting agents, maintaining a stable configuration at any time and by permanently monitoring the service's performance. An agent resides on each of the other VMs that are started by the manager. The agent is the one that does all the work. Note that a service may contain one manager and multiple agents, or multiple managers that also act as agents.

To implement a new ConPaaS service, you must provide a new manager service, a new agent service and a new frontend service (we assume that each ConPaaS service can be mapped on the three entities architecture). To ease the process of adding a new ConPaaS service, we propose a framework which implements common functionality of the ConPaaS services. So far, the framework provides abstraction for the IaaS layer (adding support for a new cloud provider should not require modifications in any ConPaaS service implementation) and it also provides abstraction for the HTTP communication (we assume that HTTP is the preferred protocol for the communication between the three entities).

4.1.1 ConPaaS directory structure

You can see below the directory structure of the ConPaaS software. The *core* folder under *src* contains the ConPaaS framework. Any service should make use of this code. It contains the manager http server, which instantiates the python manager class that implements the required service; the agent http server that instantiates the python agent class (if the service requires agents); the IaaS abstractions and other useful code.

A new service should be added in a new python module under the *ConPaaS/src/conpaas/services* folder:

```
ConPaaS/  (conpaas/conpaas-services/)
|-- src
|   |-- conpaas
|   |   |-- core
|   |   |   |-- clouds
|   |   |   |   |-- base.py
|   |   |   |   |-- dummy.py
|   |   |   |   |-- ec2.py
|   |   |   |   |-- federation.py
|   |   |   |   |-- opennebula.py
|   |   |   |   |-- openstack.py
|   |   |   |-- agent.py
|   |   |   |-- controller.py
|   |   |   |-- expose.py
|   |   |   |-- file.py
|   |   |   |-- ganglia.py
|   |   |   |-- git.py
|   |   |   |-- https
|   |   |   |-- iaas.py
|   |   |   |-- ipop.py
|   |   |   |-- log.py
|   |   |   |-- manager.py
|   |   |   |-- manager.py.generic_add_nodes
|   |   |   |-- misc.py
|   |   |   |-- node.py
|   |   |   |-- services.py
|   |   |-- services
|   |       |-- cds/
|   |       |-- galera/
|   |       |-- helloworld/
|   |       |-- htc/
|   |       |-- htcondor/
|   |       |-- mapreduce/
|   |       |-- scalaris/
|   |       |-- selenium/
|   |       |-- taskfarm/
|   |       |-- webservers/
|   |       |-- xtreemfs/
|   |-- dist
|   |-- libcloud -> ../contrib/libcloud/
|   |-- setup.py
|   |-- tests
|       |-- core
|       |-- run_tests.py
|       |-- services
|       |-- unit-tests.sh
|-- config
|-- contrib
|-- misc
|-- sbin
|-- scripts
```

In the next paragraphs we describe how to add the new ConPaaS service.

4.2 Service Organization

4.2.1 Service's name

The first step in adding a new ConPaaS service is to choose a name for it. This name will be used to construct, in a standardized manner, the file names of the scripts required by this service (see below). Therefore, the names should not contain spaces, nor unaccepted characters.

4.2.2 Scripts

To function properly, ConPaaS uses a series of configuration files and scripts. Some of them must be modified by the administrator, i.e. the ones concerning the cloud infrastructure, and the others are used, ideally unchanged, by the manager and/or the agent. A newly added service would ideally function with the default scripts. If, however, the default scripts are not satisfactory (for example the new service would need to start something on the VM, like a memcache server) then the developers must supply a new script/config file, that would be used instead of the default one. This new script's name must be preceded by the service's chosen name (as described above) and will be selected by the frontend at run time to generate the contextualization file for the manager VM. (If the frontend doesn't find such a script/config file for a given service, then it will use the default script). **Note that some scripts provided for a service do not replace the default ones, instead they will be concatenated to them (see below the agent and manager configuration scripts).**

Below we give an explanation of the scripts and configuration files used by a ConPaaS service (there are other configuration files used by the frontend but these are not relevant to the ConPaaS service). Basically there are two scripts that a service uses to boot itself up - the manager contextualization script, which is executed after the manager VM booted, and the agent contextualization script, which is executed after the agent VM booted. These scripts are composed of several parts, some of which are customizable to the needs of the new service.

In the ConPaaS home folder (CONPAAS_HOME) there is the *config* folder that contains configuration files in the INI format and the *scripts* folder that contains executable bash scripts. Some of these files are specific to the cloud, other to the manager and the rest to the agent. These files will be concatenated in a single contextualization script, as described below.

- Files specific to the Cloud:

  (1) CONPAAS_HOME/config/cloud/*cloud_name*.cfg, where *cloud_name* refers to the clouds supported by the system (for now OpenNebula and EC2). So there is one such file for each cloud the system supports. These files are filled in by the administrator. They contain information such as the username and password to access the cloud, the OS image to be used with the VMs, etc. These files are used by the frontend and the manager, as both need to ask the cloud to start VMs.

  (2) CONPAAS_HOME/scripts/cloud/*cloud_name*, where *cloud_name* refers to the clouds supported by the system (for now OpenNebula and EC2). So, as above, there is one such file for each cloud the system supports. These scripts will be included in the contextualization files. For example, for OpenNebula, this file sets up the network.

- Files specific to the Manager:

  (3) CONPAAS_HOME/scripts/manager/manager-setup, which prepares the environment by copying the ConPaaS source code on the VM, unpacking it, and setting up the PYTHONPATH environment variable.

  (4) CONPAAS_HOME/config/manager/*service_name*-manager.cfg, which contains configuration variables specific to the service manager (in INI format). If the new service needs any other variables (like a path to a file in the source code), it should provide an annex to the default manager config file. This annex must be named *service_name*-manager.cfg and will be concatenated to default-manager.cfg

  (5) CONPAAS_HOME/scripts/manager/*service_name*-manager-start, which starts the server manager and any other programs the service manager might use.

  (6) CONPAAS_HOME/sbin/manager/*service_name*-cpsmanager (will be started by the *service_name*-manager-start script), which starts the manager server, which in turn will start the requested manager service.

  Scripts (1), (2), (3), (4) and (5) will be used by the frontend to generate the contextualization script for the manager VM. After this scripts executes, a configuration file containing the concatenation of (1) and (4) will be put in ROOT_DIR/config.cfg and then (6) is started with the config.cfg file as a parameter that will be forwarded to the new service.

  Examples:

Listing 1: Script (1) ConPaaS/config/cloud/opennebula.cfg

```
[iaas]
DRIVER = OPENNEBULA

# The URL of the OCCI interface at OpenNebula. Note: ConPaaS currently
# supports only the default OCCI implementation that comes together
# with OpenNebula. It does not yet support the full OCCI-0.2 and later
# versions.
URL =

# TODO: Currently, the TaskFarming service uses XMLRPC to talk to Opennebula.
# This is the url to the server (Ex. http://dns.name.or.ip:2633/RPC2)
XMLRPC =

# Your OpenNebula user name
USER =

# Your OpenNebula password
```

```
PASSWORD =

# The image ID (an integer). You can list the registered OpenNebula
# images with command "oneimage list" command.
IMAGE_ID =

# OCCI defines 4 standard instance types: small medium large and custom. This
# variable should choose one of these. (The small, medium and large instances have
# predefined memory size and cpu, but the custom one permits the customization of
# these parameters. The best option is to use the custom variable as some services,
# like map-reduce and mysql, must be able to start VMs with a given quantity of memory)
INST_TYPE = custom

# The network ID (an integer). You can list the registered OpenNebula
# networks with the "onevnet list" command.
NET_ID =

# The network gateway through which new VMs can route their traffic in
# OpenNebula (an IP address)
NET_GATEWAY =

# The DNS server that VMs should use to resolve DNS names (an IP address)
NET_NAMESERVER =

# The OS architecture of the virtual machines.
# (corresponds to the OpenNebula "ARCH" parameter from the VM template)
OS_ARCH =

# The device that will be mounted as root on the VM. Most often it
# is "sda" or "hda" for KVM, and "xvda2" for Xen.
# (corresponds to the OpenNebula "ROOT" parameter from the VM template)
OS_ROOT =

# The device on which the VM image disk is mapped.
DISK_TARGET =

# The device associated with the CD-ROM on the virtual machine. This
# will be used for contextualization in OpenNebula. Most often it is
# "sr0" for KVM and "xvdb" for Xen.
# (corresponds to the OpenNebula "TARGET" parameter from the "CONTEXT"
# section of the VM template)
CONTEXT_TARGET =

####################################################################
# The following values are only needed by the Task Farming service #
####################################################################

PORT =

# A unique name used in the service to specify different clouds
HOSTNAME =

# The accountable time unit. Different clouds charge at different
# frequencies (e.g. Amazon charges per hour = 60 minutes)
TIMEUNIT =

# The price per TIMEUNIT of this specific machine type on this cloud
COSTUNIT =

# The maximum number of VMs that the system is allowed to allocate from this
# cloud
MAXNODES =
SPEEDFACTOR =
```

Listing 2: Script (2) ConPaaS/scripts/cloud/opennebula

```
#!/bin/bash

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

/sbin/ifconfig eth0 $IP_PUBLIC netmask $NETMASK
/sbin/ip route add default via $IP_GATEWAY
echo "nameserver $NAMESERVER" > /etc/resolv.conf
echo "prepend domain-name-servers $NAMESERVER;" >> /etc/dhcp/dhclient.conf

HOSTNAME=`/usr/bin/host $IP_PUBLIC | cut -d' ' -f5 | cut -d'.' -f1`
/bin/hostname $HOSTNAME

#########################################################################################
# Create the one_auth file from contextualization variable ONE_AUTH_CONTENT
# and set it as an environment variable for the JVM
# This is needed for services that use XMLRPC instead of OCCI

if [ $ONE_AUTH_CONTENT ]; then
  export ONE_AUTH=/root/.one_auth
  export ONE_XMLRPC
  echo $ONE_AUTH_CONTENT > $ONE_AUTH
fi

# PCI Hotplug Support is needed in order to attach persistent storage volumes
# to this instance
/sbin/modprobe acpiphp
/sbin/modprobe pci_hotplug
```

Listing 3: Script (3) ConPaaS/scripts/manager/manager-setup

```
#!/bin/bash

# This script is part of the contextualization file. It
# copies the source code on the VM, unpacks it, and sets
# the PYTHONPATH environment variable.

# Is filled in by the director
DIRECTOR=%DIRECTOR_URL%
SOURCE=$DIRECTOR/download
ROOT_DIR=/root
CPS_HOME=$ROOT_DIR/ConPaaS

LOG_FILE=/var/log/cpsmanager.log
ETC=/etc/cpsmanager
CERT_DIR=$ETC/certs
VAR_TMP=/var/tmp/cpsmanager
VAR_CACHE=/var/cache/cpsmanager
VAR_RUN=/var/run/cpsmanager

mkdir $CERT_DIR
mv /tmp/*.pem $CERT_DIR

wget --ca-certificate=$CERT_DIR/ca_cert.pem -P $ROOT_DIR/ $SOURCE/ConPaaS.tar.gz
tar -zxf $ROOT_DIR/ConPaaS.tar.gz -C $ROOT_DIR/
export PYTHONPATH=$CPS_HOME/src/:$CPS_HOME/contrib/
```

Listing 4: Script (4) ConPaaS/config/manager/default-manager.cfg

```
[manager]
```

```
# Service TYPE will be filled in by the director
TYPE = %CONPAAS_SERVICE_TYPE%

BOOTSTRAP = $SOURCE
MY_IP = $IP_PUBLIC

# These are used by the manager to
# communicate with the director to:
#  - decrement the number of credits the user has.
#    (they are used when a VM ran more than 1 hour)
#  - request a new certificate from the CA
# Everything will be filled in by the director
DEPLOYMENT_NAME = %CONPAAS_DEPLOYMENT_NAME%
SERVICE_ID = %CONPAAS_SERVICE_ID%
USER_ID = %CONPAAS_USER_ID%
APP_ID = %CONPAAS_APP_ID%
CREDIT_URL = %DIRECTOR_URL%/callback/decrementUserCredit.php
TERMINATE_URL = %DIRECTOR_URL%/callback/terminateService.php
CA_URL = %DIRECTOR_URL%/ca/get_cert.php

IPOP_BASE_NAMESPACE = %DIRECTOR_URL%/ca/get_cert.php
# The following IPOP directives are added by the director if necessary
# IPOP_BASE_IP = %IPOP_BASE_IP%
# IPOP_NETMASK = %IPOP_NETMASK%
# IPOP_IP_ADDRESS = %IPOP_IP_ADDRESS%
# IPOP_SUBNET  = %IPOP_SUBNET%

# This directory structure already exists in the VM (with ROOT = '') - see
# the 'create new VM script' so do not change ROOT unless you also modify
# it in the VM. Use these files/directories to put variable data that
# your manager might generate during its life cycle
LOG_FILE = $LOG_FILE
ETC = $ETC
CERT_DIR = $CERT_DIR
VAR_TMP = $VAR_TMP
VAR_CACHE = $VAR_CACHE
VAR_RUN = $VAR_RUN
CODE_REPO = %(VAR_CACHE)s/code_repo

CONPAAS_HOME = $CPS_HOME

# The default block device where the disks are attached to.
DEV_TARGET = sdb

# Add below other config params your manager might need and save a file as
# %service_name%-manager.cfg
# Otherwise this file will be used by default
```

Listing 5: Script (5) ConPaaS/scripts/manager/default-manager-start

```
#!/bin/bash

# This script is part of the contextualization file. It
# starts a python script that parses the given arguments
# and starts the manager server, which in turn will start
# the manager service.

# This file is the default manager-start file. It can be
# customized as needed by the service.

$CPS_HOME/sbin/manager/default-cpsmanager -c $ROOT_DIR/config.cfg 1>$ROOT_DIR/manager.out 2>$ROOT_
manager_pid=$!
echo $manager_pid > $ROOT_DIR/manager.pid
```

Listing 6: Script (6) ConPaaS/sbin/manager/default-cpsmanager

```
#!/usr/bin/python
'''
Copyright (c) 2010-2012, Contrail consortium.
All rights reserved.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

 1. Redistributions of source code must retain the
    above copyright notice, this list of conditions
    and the following disclaimer.
 2. Redistributions in binary form must reproduce
    the above copyright notice, this list of
    conditions and the following disclaimer in the
    documentation and/or other materials provided
    with the distribution.
 3. Neither the name of the Contrail consortium nor the
    names of its contributors may be used to endorse
    or promote products derived from this software
    without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.


Created on Jul 4, 2011

@author: ielhelw
'''
from os.path import exists
from conpaas.core.https import client, server

if __name__ == '__main__':
  from optparse import OptionParser
  from ConfigParser import ConfigParser
  import sys

  parser = OptionParser()
  parser.add_option('-p', '--port', type='int', default=443, dest='port')
  parser.add_option('-b', '--bind', type='string', default='0.0.0.0', dest='address')
  parser.add_option('-c', '--config', type='string', default=None, dest='config')
  options, args = parser.parse_args()

  if not options.config or not exists(options.config):
    print >>sys.stderr, 'Failed to find configuration file'
    sys.exit(1)

  config_parser = ConfigParser()
```

```
try:
  config_parser.read(options.config)
except:
  print >>sys.stderr, 'Failed to read configuration file'
  sys.exit(1)

"""
Verify some sections and variables that must exist in the configuration file
"""
config_vars = {
  'manager': ['TYPE', 'BOOTSTRAP', 'LOG_FILE',
              'CREDIT_URL', 'TERMINATE_URL', 'SERVICE_ID'],
  'iaas': ['DRIVER'],
}
config_ok = True
for section in config_vars:
  if not config_parser.has_section(section):
    print >>sys.stderr, 'Missing configuration section "%s"' % (section)
    print >>sys.stderr, 'Section "%s" should contain variables %s' % (section, str(config_vars[
    config_ok = False
    continue
  for field in config_vars[section]:
    if not config_parser.has_option(section, field)\
    or config_parser.get(section, field) == '':
      print >>sys.stderr, 'Missing configuration variable "%s" in section "%s"' % (field, secti
      config_ok = False
if not config_ok:
  sys.exit(1)

# Initialize the context for the client
client.conpaas_init_ssl_ctx(config_parser.get('manager', 'CERT_DIR'),
                                  'manager', config_parser.get('manager', 'USER_ID'),
                                  config_parser.get('manager', 'SERVICE_ID'))

# Start the manager server
print options.address, options.port
d = server.ConpaasSecureServer((options.address, options.port),
                   config_parser,
                   'manager',
                   reset_config=True)
d.serve_forever()
```

- Files specific to the Agent

  They are similar to the files described above for the manager, but this time the contextualization file is generated by the manager.

4.2.3 Scripts and config files directory structure

Below you can find the directory structure of the scripts and configuration files described above.

```
ConPaaS/  (conpaas/conpaas-services/)
|-- config
|   |-- agent
|   |   |-- default-agent.cfg
|   |   |-- galera-agent.cfg
|   |   |-- helloworld-agent.cfg
|   |   |-- htc-agent.cfg
|   |   |-- htcondor.cfg
|   |   |-- mapreduce-agent.cfg
|   |   |-- scalaris-agent.cfg
|   |   |-- web-agent.cfg
|   |   |-- xtreemfs-agent.cfg
|   |-- cloud
|   |   |-- clouds-template.cfg
|   |   |-- ec2.cfg
|   |   |-- ec2.cfg.example
|   |   |-- opennebula.cfg
|   |   |-- opennebula.cfg.example
|   |-- ganglia
|   |   |-- ganglia_frontend.tmpl
|   |   |-- ganglia-gmetad.tmpl
|   |   |-- ganglia-gmond.tmpl
|   |-- ipop
|   |   |-- bootstrap.config.tmpl
|   |   |-- dhcp.config.tmpl
|   |   |-- ipop.config.tmpl
|   |   |-- ipop.vpn.config.tmpl
|   |   |-- node.config.tmpl
|   |-- manager
|       |-- default-manager.cfg
|       |-- htc-manager.cfg
|       |-- htcondor.cfg
|       |-- java-manager.cfg
|       |-- php-manager.cfg
|-- sbin
|   |-- agent
|   |   |-- default-cpsagent
|   |   |-- web-cpsagent
|   |-- manager
|       |-- default-cpsmanager
|       |-- php-cpsmanager
|       |-- taskfarm-cpsmanager
|-- scripts
    |-- agent
    |   |-- agent-setup
    |   |-- default-agent-start
    |   |-- htc-agent-start
    |   |-- htcondor-agent-start
    |   |-- mapreduce-agent-start
    |   |-- scalaris-agent-start
    |   |-- selenium-agent-start
    |   |-- taskfarm-agent-start
    |   |-- web-agent-start
    |   |-- xtreemfs-agent-start
    |-- cloud
    |   |-- dummy
    |   |-- ec2
    |   |-- federation
    |   |-- opennebula
    |   |-- openstack
    |-- create_vm
    |   |-- 40_custom
    |   |-- create-img-conpaas.sh
    |   |-- create-img-script.cfg
    |   |-- create-img-script.py
    |   |-- README
    |   |-- register-image-ec2-ebs.sh
    |   |-- register-image-ec2-s3.sh
    |   |-- register-image-opennebula.sh
    |   |-- scripts
    |       |-- 000-head
    |       |-- 003-create-image
    |       |-- 004-conpaas-core
    |       |-- 501-php
    |       |-- 502-galera
    |       |-- 503-condor
    |       |-- 504-selenium
    |       |-- 505-hadoop
    |       |-- 506-scalaris
    |       |-- 507-xtreemfs
    |       |-- 508-cds
    |       |-- 995-rm-unused-pkgs
    |       |-- 996-user
    |       |-- 997-tail
    |       |-- 998-ec2
    |       |-- 998-opennebula
    |       |-- 999-resize-image
    |-- manager
        |-- cds-manager-start
        |-- default-git-deploy-hook
        |-- default-manager-start
        |-- htc-manager-start
        |-- htcondor-manager-start
        |-- java-manager-start
        |-- manager-setup
        |-- notify_git_push.py
        |-- php-manager-start
        |-- taskfarm-manager-start
```

4.3 Implementing a new ConPaaS service using blueprints

Blueprints are service templates you can use to speed up the creation of a new service. You can use this blueprinting mechanism with `create-new-service-from-blueprints.sh`.

The `conpaas-blueprints` tree contains the following files:

```
conpaas-blueprints
|-- conpaas-client
|   |-- cps
|       |-- blueprint.py
|-- conpaas-frontend
|   |-- www
|       |-- images
|       |   |-- blueprint.png
|       |-- js
|       |   |-- blueprint.js
|       |-- lib
|           |-- service
|           |   |-- blueprint
|           |       |-- __init__.php
|           |-- ui
|               |-- instance
|               |   |-- blueprint
|               |       |-- __init__.php
|               |-- page
|                   |-- blueprint
|                       |-- __init__.php
|-- conpaas-services
    |-- scripts
    |   |-- create_vm
    |       |-- scripts
    |           |-- 5xx-blueprint
    |-- src
        |-- conpaas
            |-- services
```

```
                |-- blueprint
                    |-- agent
                    |   |-- agent.py
                    |   |-- client.py
                    |   |-- __init__.py
                    |-- __init__.py
                    |-- manager
                        |-- client.py
                        |-- __init__.py
                        |-- manager.py
```

Edit `create-new-service-from-blueprints.sh` and change the following lines to set up the script:

```
BP_lc_name=foobar                  # Lowercase service name in the tree
BP_mc_name=FooBar                  # Mixedcase service name in the tree
BP_uc_name=FOOBAR                  # Uppercase service name in the tree
BP_bp_name='Foo Bar'               # Selection name as shown on the frontend  create.php  page
BP_bp_desc='My new FooBar Service' # Description as shown on the frontend  create.php  page
BP_bp_num=511                      # Service sequence number for
                                   # conpaas-services/scripts/create_vm/create-img-script.cfg
                                   # Please look in conpaas-services/scripts/create_vm/scripts
                                   # for the first available number
```

Running the script in the ConPaaS root will copy the files from the tree above to the appropriate places in the `conpaas-client`, `conpaas-frontend` and `conpaas-services` trees. In the process of copying, the above keywords will be replaced by the values you entered, and files and directories named `*blueprint*` will be replaced by the new service name. Furthermore, the following files will be adjusted similarly:

```
conpaas-services/src/conpaas/core/services.py
conpaas-frontend/www/create.php
conpaas-frontend/www/lib/ui/page/PageFactory.php
conpaas-frontend/www/lib/service/factory/__init__.php
```

Now you are ready to set up the specifics for your service. In most newly created files you will find the following comment

```
*TODO: as this file was created from a BLUEPRINT file, you may want to
change ports, paths and/or methods (e.g. for hub) to meet your specific
service/server needs*.
```

So it's a good idea to do just that.

4.4 Implementing a new ConPaaS service by hand

In this section we describe how to implement a new ConPaaS service by providing an example which can be used as a starting point. The new service is called *helloworld* and will just generate helloworld strings. Thus, the manager will provide a method, called get_helloworld which will ask all the agents to return a 'helloworld' string (or another string chosen by the manager).

We will start by implementing the agent. We will create a class, called HelloWorldAgent, which implements the required method - get_helloworld, and put it in *conpaas/services/helloworld/agent/agent.py* (Note: make the directory structure as needed and providing empty __init__.py to make the directory be recognized as a module path). As you can see in Listing 7, this class uses some functionality provided in the conpaas.core package. The conpaas.core.expose module provides a python decorator (@expose) that can be used to expose the http methods that the agent server dispatches. By using this decorator, a dictionary containing methods for http requests GET, POST or UPLOAD is filled in behind the scenes. This dictionary is used by the built-in server in the conpaas.core package to dispatch the HTTP requests. The module conpaas.core.http contains some useful methods, like HttpJsonResponse and HttpErrorResponse that are used to respond to the HTTP request dispatched to the corresponding method. In this class we also implemented a method called startup, which only changes the state of the agent. This method could be used, for example, to make some initializations in the agent. We will describe later the use of the other method, check_agent_process.

Listing 7: conpaas/services/helloworld/agent/agent.py

```
from conpaas.core.expose import expose

from conpaas.core.https.server import HttpJsonResponse, HttpErrorResponse

from conpaas.core.agent import BaseAgent

class HelloWorldAgent(BaseAgent):
    def __init__(self,
                 config_parser, # config file
                 **kwargs):      # anything you can't send in config_parser
                                 # (hopefully the new service won't need anything extra)
      BaseAgent.__init__(self, config_parser)
      self.gen_string = config_parser.get('agent', 'STRING_TO_GENERATE')

    @expose('POST')
    def startup(self, kwargs):
      self.state = 'RUNNING'
      self.logger.info('Agent started up')
      return HttpJsonResponse()

    @expose('GET')
    def get_helloworld(self, kwargs):
      if self.state != 'RUNNING':
        return HttpErrorResponse('ERROR: Wrong state to get_helloworld')
      return HttpJsonResponse({'result':self.gen_string})
```

Let's assume that the manager wants each agent to generate a different string. The agent should be informed about the string that it has to generate. To do this, we could either implement a method inside the agent, that will receive the required string, or specify this string in the configuration file with which the agent is started. We opted for the second method just to illustrate how a service could make use of the config files and also, maybe some service agents/managers need some information before having been started.

Therefore, we will provide the *helloworld-agent.cfg* file (see Listing 8) that will be concatenated to the default-manager.cfg file. It contains a variable ($STRING) which will be replaced by the manager.

Listing 8: ConPaaS/config/agent/helloworld-agent.cfg

```
STRING_TO_GENERATE = $STRING
```

Now let's implement an http client for this new agent server. See Listing 9. This client will be used by the manager as a wrapper to easily send requests to the agent. We used some useful methods from conpaas.core.http, to send json objects to the agent server.

Listing 9: conpaas/services/helloworld/agent/client.py

```
import json
import httplib

from conpaas.core import https

def _check(response):
  code, body = response
  if code != httplib.OK: raise Exception('Received http response code %d' % (code))
  data = json.loads(body)
  if data['error']: raise Exception(data['error'])
  else: return data['result']

def check_agent_process(host, port):
  method = 'check_agent_process'
  return _check(https.client.jsonrpc_get(host, port, '/', method))

def startup(host, port):
  method = 'startup'
  return _check(https.client.jsonrpc_post(host, port, '/', method))

def get_helloworld(host, port):
  method = 'get_helloworld'
  return _check(https.client.jsonrpc_get(host, port, '/', method))
```

Next, we will implement the manager in the same manner: we will write the *HelloWorldManager* class and place it in the file *conpaas/services/helloworld/manager/manager.py*. (See Listing 10) To make use of the IaaS abstractions, we need to instantiate a Controller which controls all the requests to the clouds on which ConPaaS is running. Note the lines:

```
1: self.controller = Controller( config_parser)
2: self.controller.generate_context('helloworld')
```

The first line instantiates a Controller. The controller maintains a list of cloud objects generated from the *config_parser* file. There are several functions provided by the controller which are documented in the doxygen documentation of file *controller.py*. The most important ones, which are also used in the Hello World service implementation, are: *generate_context* (which generates a template of the contextualization file); *update_context* (which takes the contextualization template and replaces the variables with the supplied values); *create_nodes* (which asks for additional nodes from the specified cloud or the default one) and *delete_nodes* (which deletes the specified nodes).

Note that the *create_nodes* function accepts as a parameter a function (in our case *check_agent_process*) that tests if the agent process started correctly in the agent VM. If an exception is generated during the calls to this function for a given period of time, then the manager assumes that the agent process didn't start correctly and tries to start the agent process on a different agent VM.

Listing 10: conpaas/services/helloworld/manager/manager.py

```
from threading import Thread

from conpaas.core.expose import expose
from conpaas.core.manager import BaseManager

from conpaas.core.https.server import HttpJsonResponse, HttpErrorResponse

from conpaas.services.helloworld.agent import client

class HelloWorldManager(BaseManager):

    # Manager states - Used by the Director
    S_INIT = 'INIT'         # manager initialized but not yet started
    S_PROLOGUE = 'PROLOGUE' # manager is starting up
    S_RUNNING = 'RUNNING'   # manager is running
    S_ADAPTING = 'ADAPTING' # manager is in a transient state - frontend will keep
                            # polling until manager out of transient state
    S_EPILOGUE = 'EPILOGUE' # manager is shutting down
    S_STOPPED = 'STOPPED'   # manager stopped
    S_ERROR = 'ERROR'       # manager is in error state

    AGENT_PORT = 5555

    def __init__(self, config_parser, **kwargs):
        BaseManager.__init__(self, config_parser)
        self.nodes = []
        # Setup the clouds' controller
        self.controller.generate_context('helloworld')
        self.state = self.S_INIT

    def _do_startup(self, cloud):
        startCloud = self._init_cloud(cloud)
        self.controller.add_context_replacement(dict(STRING='helloworld'))

        try:
            nodes = self.controller.create_nodes(1,
                client.check_agent_process, self.AGENT_PORT, startCloud)

            node = nodes[0]

            client.startup(node.ip, self.AGENT_PORT)

            # Extend the nodes list with the newly created one
            self.nodes += nodes
            self.state = self.S_RUNNING
        except Exception, err:
            self.logger.exception('_do_startup: Failed to create node: %s' % err)
            self.state = self.S_ERROR

    @expose('POST')
    def shutdown(self, kwargs):
        self.state = self.S_EPILOGUE
        Thread(target=self._do_shutdown, args=[]).start()
        return HttpJsonResponse()

    def _do_shutdown(self):
        self.controller.delete_nodes(self.nodes)
        self.nodes = []
        self.state = self.S_STOPPED

    @expose('POST')
    def add_nodes(self, kwargs):
        if self.state != self.S_RUNNING:
            return HttpErrorResponse('ERROR: Wrong state to add_nodes')

        if 'node' in kwargs:
            kwargs['count'] = kwargs['node']

        if not 'count' in kwargs:
            return HttpErrorResponse("ERROR: Required argument doesn't exist")

        if not isinstance(kwargs['count'], int):
            return HttpErrorResponse('ERROR: Expected an integer value for "count"')

        count = int(kwargs['count'])

        cloud = kwargs.pop('cloud', 'iaas')
        try:
            cloud = self._init_cloud(cloud)
        except Exception as ex:
                return HttpErrorResponse(
                    "A cloud named '%s' could not be found" % cloud)

        self.state = self.S_ADAPTING
        Thread(target=self._do_add_nodes, args=[count, cloud]).start()
        return HttpJsonResponse()

    def _do_add_nodes(self, count, cloud):
        node_instances = self.controller.create_nodes(count,
                client.check_agent_process, self.AGENT_PORT, cloud)

        self.nodes += node_instances
        # Startup agents
        for node in node_instances:
            client.startup(node.ip, self.AGENT_PORT)
        self.state = self.S_RUNNING
        return HttpJsonResponse()

    @expose('GET')
    def list_nodes(self, kwargs):
        if len(kwargs) != 0:
            return HttpErrorResponse('ERROR: Arguments unexpected')

        if self.state != self.S_RUNNING:
            return HttpErrorResponse('ERROR: Wrong state to list_nodes')

        return HttpJsonResponse({
              'helloworld': [ node.id for node in self.nodes ],
              })

    @expose('GET')
    def get_service_info(self, kwargs):
        if len(kwargs) != 0:
            return HttpErrorResponse('ERROR: Arguments unexpected')

        return HttpJsonResponse({'state': self.state, 'type': 'helloworld'})

    @expose('GET')
    def get_node_info(self, kwargs):
        if 'serviceNodeId' not in kwargs:
            return HttpErrorResponse('ERROR: Missing arguments')

        serviceNodeId = kwargs.pop('serviceNodeId')

        if len(kwargs) != 0:
            return HttpErrorResponse('ERROR: Arguments unexpected')

        serviceNode = None
        for node in self.nodes:
            if serviceNodeId == node.id:
                serviceNode = node
                break

        if serviceNode is None:
            return HttpErrorResponse('ERROR: Invalid arguments')

        return HttpJsonResponse({
            'serviceNode': {
                            'id': serviceNode.id,
                            'ip': serviceNode.ip
                            }
            })

    @expose('POST')
    def remove_nodes(self, kwargs):
        if self.state != self.S_RUNNING:
            return HttpErrorResponse('ERROR: Wrong state to remove_nodes')

        if 'node' in kwargs:
            kwargs['count'] = kwargs['node']

        if not 'count' in kwargs:
            return HttpErrorResponse("ERROR: Required argument doesn't exist")

        if not isinstance(kwargs['count'], int):
            return HttpErrorResponse('ERROR: Expected an integer value for "count"')
        count = int(kwargs['count'])
        self.state = self.S_ADAPTING
        Thread(target=self._do_remove_nodes, args=[count]).start()
        return HttpJsonResponse()

    def _do_remove_nodes(self, count):
        for _ in range(0, count):
            self.controller.delete_nodes([ self.nodes.pop() ])

        self.state = self.S_RUNNING
        return HttpJsonResponse()

    @expose('GET')
    def get_helloworld(self, kwargs):
        if self.state != self.S_RUNNING:
            return HttpErrorResponse('ERROR: Wrong state to get_helloworld')

        messages = []

        # Just get_helloworld from all the agents
        for node in self.nodes:
            data = client.get_helloworld(node.ip, self.AGENT_PORT)
            message = 'Received %s from %s' % (data['result'], node.id)
            self.logger.info(message)
            messages.append(message)

        return HttpJsonResponse({ 'helloworld': "\n".join(messages) })
```

We can also implement a client for the manager server (see Listing 11). This will allow us to use the command line interface to send requests to the manager, if the frontend integration is not available.

Listing 11: conpaas/services/helloworld/manager/client.py

```
import httplib , json
from conpaas.core.http import HttpError, _jsonrpc_get, _jsonrpc_post,  _http_post, _http_get

def _check(response):
    code, body = response
    if code != httplib.OK: raise HttpError('Received http response code %d' % (code))
    data = json.loads(body)
    if data['error']: raise Exception(data['error'])
    else : return data['result']

def get_service_info(host, port):
    method = 'get_service_info'
    return _check(_jsonrpc_get(host, port , '/' , method))

def get_helloworld(host, port):
    method = 'get_helloworld'
    return _check(_jsonrpc_get(host, port , '/' , method))

def startup(host, port):
    method = 'startup'
    return _check(_jsonrpc_get(host, port , '/' , method))

def add_nodes(host, port , count=0):
    method = 'add_nodes'
    params = {}
    params['count'] = count
    return _check(_jsonrpc_post(host, port , '/', method, params=params))

def remove_nodes(host , port , count=0):
    method = 'remove_nodes'
    params = {}
    params['count'] = count
    return _check(_jsonrpc_post(host, port , '/', method, params=params))

def list_nodes(host, port):
    method = 'list_nodes'
    return _check(_jsonrpc_get(host, port , '/' , method))
```

The last step is to register the new service to the conpaas core. One entry must be added to file *conpaas/core/services.py*, as it is indicated in Listing 12. Because the Java and PHP services use the same code for the agent, there is only one entry in the agent services, called *web* which is used by both webservices.

Listing 12: conpaas/core/services.py

```
# -*- coding: utf-8 -*-

"""
    conpaas.core.services
    =====================

    ConPaaS core: map available services to their classes.

    :copyright: (C) 2010-2013 by Contrail Consortium.
"""

manager_services = {'php'    : {'class' : 'PHPManager',
                                'module': 'conpaas.services.webservers.manager.internal.php'},
                    'java'   : {'class' : 'JavaManager',
                                'module': 'conpaas.services.webservers.manager.internal.java'},
                    'scalaris' : {'class' : 'ScalarisManager',
                                  'module': 'conpaas.services.scalaris.manager.manager'},
                    'hadoop' : {'class' : 'MapReduceManager',
                                'module': 'conpaas.services.mapreduce.manager.manager'},
                    'helloworld' : {'class' : 'HelloWorldManager',
                                    'module': 'conpaas.services.helloworld.manager.manager'},
                    'xtreemfs' : {'class' : 'XtreemFSManager',
                                  'module': 'conpaas.services.xtreemfs.manager.manager'},
                    'selenium' : {'class' : 'SeleniumManager',
                                  'module': 'conpaas.services.selenium.manager.manager'},
                    'taskfarm' : {'class' : 'TaskFarmManager',
                                  'module': 'conpaas.services.taskfarm.manager.manager'},
                    'galera' : {'class' : 'GaleraManager',
                               'module': 'conpaas.services.galera.manager.manager'},

#                    'htcondor' : {'class' : 'HTCondorManager',
#                                  'module': 'conpaas.services.htcondor.manager.manager'},
                    'htc' : {'class' : 'HTCManager',
                                  'module': 'conpaas.services.htc.manager.manager'},
                    'generic' : {'class' : 'GenericManager',
                                  'module': 'conpaas.services.generic.manager.manager'},

#""" BLUE_PRINT_INSERT_MANAGER                 do not remove this line: it is a placeholder for i
                    }

agent_services = {'web' : {'class' : 'WebServersAgent',
                           'module': 'conpaas.services.webservers.agent.internals'},
                  'scalaris' : {'class' : 'ScalarisAgent',
                                'module': 'conpaas.services.scalaris.agent.agent'},
                  'mapreduce' : {'class' : 'MapReduceAgent',
                                 'module': 'conpaas.services.mapreduce.agent.agent'},
                  'helloworld' : {'class' : 'HelloWorldAgent',
                                  'module': 'conpaas.services.helloworld.agent.agent'},
                  'xtreemfs' : {'class' : 'XtreemFSAgent',
                                'module': 'conpaas.services.xtreemfs.agent.agent'},
                  'selenium' : {'class' : 'SeleniumAgent',
                                'module': 'conpaas.services.selenium.agent.agent'},
                    'galera' : {'class' : 'GaleraAgent',
                             'module': 'conpaas.services.galera.agent.internals'},

#                    'htcondor' : {'class' : 'HTCondorAgent',
#                                 'module': 'conpaas.services.htcondor.agent.agent'},
                  'htc' : {'class' : 'HTCAgent',
                                'module': 'conpaas.services.htc.agent.agent'},
                  'generic' : {'class' : 'GenericAgent',
                                'module': 'conpaas.services.generic.agent.agent'},
#""" BLUE_PRINT_INSERT_AGENT                 do not remove this line: it is a placeholder for ins
                  }
```

4.5 Integrating the new service with the frontend

So far there is no easy way to add a new frontend service. Each service may require distinct graphical elements. In this section we explain how the Hello World frontend service has been created.

4.5.1 Manager states

As you have noticed in the Hello World manager implementation, we used some standard states, e.g. INIT, ADAPTING, etc. By calling the *get_service_info* function, the frontend knows in which state the manager is. Why do we need these standardized stated? As an example, if the manager is in the ADAPTING state, the frontend would know to draw a loading icon on the interface and keep polling the manager.

4.5.2 Files to be modified

```
frontend
|-- www
    |-- create.php
    |-- lib
        |-- service
            |-- factory
                |-- __init__.php
```

Several lines of code must be added to the two files above for the new service to be recognized. If you look inside these files, you'll see that knowing where to add the lines and what lines to add is self-explanatory.

4.5.3 Files to be added

```
frontend
|-- www
    |-- lib
    |   |-- service
    |   |   |-- helloworld
    |   |       |-- __init__.php
    |   |-- ui
    |       |-- instance
    |           |-- helloworld
    |               |-- __init__.php
    |-- images
        |-- helloworld.png
```