ConPaaS Documentation

Release 1.3.1

The ConPaaS team <info@conpaas.eu>

November 14, 2013

Contents

CHAPTER 1

Installation

ConPaaS is a Platform-as-a-Service system. It aims at simplifying the deployment and management of applications in the Cloud.

The central component of ConPaaS, called *ConPaaS Director* (**cpsdirector**), 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**). 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 EC2 image (ami-1d620e74) is known to work well. Please note that the *root* account is disabled and that you should instead login as *admin*.

ConPaaS services are designed to run either in an *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 OpenNebula* describes how to create a ConPaaS 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 *Command line tool installation*.
4. Install **cpsfrontend** and configure it to use your ConPaaS director as explained in *Frontend installation*.

1.1 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 GNU/Linux system. Although the ConPaaS director might run on other distributions, only Debian versions 6.0 (Squeeze) and 7.0 (Wheezy) are officially supported. Also, only official Debian 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-1.3.1.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 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
   ```

3. Download http://www.conpaas.eu/dl/cpsdirector-1.3.1.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, 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 OpenNebula image suitable for ConPaaS is available in *ConPaaS on OpenNebula*.

The installation process will create an *Apache VirtualHost* for the ConPaaS director in `/etc/apache2/sites-available/conpaas-director`. 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 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
```

1.1.1 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
```

1.1.2 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
```

1.1.3 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.

1.1.4 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).

IPOP is useful when you need 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 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
```

1.1.5 Troubleshooting

There are a few things you can check if for some reason your Director installation is not behaving as expected.

If you cannot create services, this is what you should try to do on 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/cc
   ```

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:~# 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`.

1.2 Command line tool installation

The command line tool, called `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-1.3.1.tar.gz
```

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-1.3.1.tar.gz
```

1.3 Frontend installation

As for the Director, only Debian versions 6.0 (Squeeze) and 7.0 (Wheezy) are 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-1.3.1.tar.gz.

After having uncompressed it you should install the required Debian 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/conpaas/
```

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/conpaas/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!

1.4 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*.

1.4.1 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 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.

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.

1.4.2 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.

1.4.3 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 your 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.

1.5 ConPaaS on Amazon EC2

The Web Hosting Service 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 the Web Hosting Service. You can skip this section if you plan to install ConPaaS over 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.

1.5.1 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-f4c75fc4` United States West (Oregon)
- `ami-c3045aaa` United States East (Northern Virginia)
- `ami-b79271c0` Europe West (Ireland)

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

1.5.2 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 credetials can be found at About AWS Security Credentials.

1.5.3 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 port 3306 – used by the MySQL service
- 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.

1.6 ConPaaS on OpenNebula

The Web Hosting Service 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.

1.6.1 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!).

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.

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.

1.6.2 Make sure OpenNebula is properly configured

OpenNebula's OCCI daemon is used by ConPaaS to communicate with your OpenNebula cluster.

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

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

2. 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",port="-1"]
   ```

   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.

3. 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,
```

CHAPTER 2

User Guide

ConPaaS 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.

ConPaaS currently contains nine services:

- **Two Web hosting services** respectively specialized for hosting PHP and JSP applications;
- **MySQL 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.

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.

2.1 Usage overview

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, 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.

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.

2.1.1 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.

2.1.2 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 *Command line tool installation* 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
```

2.1.3 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.

2.2 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.

2.3 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.

2.3.1 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` or `zip` format. Attention: 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 “make 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
```

2.3.2 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.

2.3.3 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.

2.3.4 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
```

2.4 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.

2.4.1 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 "make 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*.

2.4.2 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.

2.5 The MySQL database service

The MySQL service provides the famous database in the form of a ConPaaS service. When scaling the service up and down, it creates (or deletes) database replicas using the master-slave mechanism. At the moment, the service does not implement load balancing of database queries between the master and its slaves. Replication therefore provides fault-tolerance properties but no performance improvement.

2.5.1 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.

2.5.2 Accessing the database

The frontend provides the command-line to access the database. 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 the `mysqldb` user has extended privileges. It can create new databases, new users etc.

2.5.3 Uploading a database dump

The ConPaaS frontend allows 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
```

2.6 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.

2.6.1 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.

2.6.2 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.

2.7 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.

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

2.8 The TaskFarm service

The TaskFarm 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.fter 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.

2.8.1 Preparing the ConPaaS services image

By default, the TaskFarm 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.

2.8.2 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.

2.8.3 The filesystem location

TaskFarm 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 TaskFarm agents.

2.8.4 The demo mode

With large bags of tasks and/or with long running tasks, the TaskFarm 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. TaskFarm service also provides a "demo" mode where the users can try the service with custom bags without spending time and money.

2.9 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.

2.9.1 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.

2.9.2 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).

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

2.9.4 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.

2.10 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.

2.10.1 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.

CHAPTER 3

Internals

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).

3.1 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/services* folder.

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

3.1.1 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.

3.1.2 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:

- 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.

3.2 Scripts and config files directory structure

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

3.2.1 Implementing a new ConPaaS service

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 *conpaasservices/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 [lst:helloworldagent], 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.

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 [lst:helloworldcfg]) that will be concatenated to the default-manager.cfg file. It contains a variable ($STRING) which will be replaced by the manager.

Now let's implement an http client for this new agent server. See Listing [lst:helloworldagentclient]. 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.

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*. 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.

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

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 [lst:helloworldservices]. 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.

3.2.2 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.

3.3 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.

3.4 Files to be modified

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.

3.5 Files to be added