ConPaaS Documentation

Release 2.1.0

The ConPaaS team <info@conpaas.eu>

November 17, 2016

Contents

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 **cps-tools**) 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, **cps-tools** 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, Ubuntu 14.04 and Ubuntu 16.04 images are known to work well.

ConPaaS services are designed to run either in an *OpenStack* 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 configure ConPaaS with Amazon EC2 can be found in *ConPaaS on Amazon EC2*. The section *ConPaaS on OpenStack* describes how to configure ConPaaS to work with an OpenStack cloud.
2. Install and configure **cpsdirector** as explained in *Director installation*. All system configuration takes place in the director.
3. Install and configure **cps-tools** as explained in *Installing and configuring cps-tools*.
4. 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), 14.04 (Trusty Tahr) and 16.04 (Xenial Xerus) 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-2.1.0.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 libssl-dev libffi-dev
$ 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-2.1.0.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 OpenStack image if you want to use ConPaaS on OpenStack. 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*.

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/2.4/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 components 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
```

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

2.1.3 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.

2.1.4 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

The new command line client for ConPaaS is called `cps-tools`.

2.2.1 Installing and configuring cps-tools

The command line `cps-tools` is a 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 arguments consist of 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-2.1.0.tar.gz
$ cd cps-tools-2.1.0
$ ./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-2.1.0.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 wi
$ cps-user get_certificate
>> enter you password
>> now you can use cps-tools commands
```

2.2.2 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   su
$ 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), 14.04 (Trusty Tahr) and 16.04 (Xenial Xerus) 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-2.1.0.tar.gz.

After having uncompressed it you should install the required packages:

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

If you use Ubuntu 16.04 (which ships with PHP 7), the following command may be used (the Frontend supports PHP 7 as well):

```
$ sudo apt-get install libapache2-mod-php php-curl php-zip
```

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` (or `nano /etc/php/7.0/apache2/php.ini` for PHP 7.0). 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 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 use specialized versions such as the Nutshell or ConPaaS on Raspberry PI.

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.

2.4.1 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-14f2d903` United States East (Northern Virginia)
- `ami-0db8ed6d` United States West (Northern California)
- `ami-9dae03fd` United States West (Oregon)
- `ami-db5e0da8` Europe West (Ireland)
- `ami-36f15b57` Asia Pacific (Tokyo)
- `ami-45389b26` Asia Pacific (Singapore)
- `ami-5f84bb3c` Asia Pacific (Sydney)
- `ami-5b1c8337` South America (Sao Paulo)

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

2.4.2 Registering your custom VM image to Amazon EC2

Using prebuilt Amazon Machine Images is the recommended way of running ConPaaS on Amazon EC2, as described in the previous section. If you plan to use one of these AMIs, you can skip this section and continue with the configuration of the Security Group.

You can also download a prebuilt ConPaaS services image that is suitable to be used with Amazon EC2, for example in case you wish to run ConPaaS in a different Availability Zone. This image is available from the following link:

ConPaaS VM image for Amazon EC2 (x86_64):

> http://www.conpaas.eu/dl/conpaas-2.1.0-amazon.img.tar.gz
> MD5: 2d2af415a29c8413b898eacfcc2f1343
> size: 541 MB

In case you prefer to use a custom services image, you can also create a new Amazon Machine Image yourself, by following the instructions from the Internals guide: *Creating A ConPaaS Services VM Image*. Come back to this section after you already generated the `conpaas.img` file.

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 an 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 a 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.

2.4.3 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 443 and 5555 used by the ConPaaS system (director, managers, and agents)
- TCP ports 80, 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 32636, 32638 and 32640 – used by the XtreemFS service
- TCP ports 22, 6121, 6122, 6123, 6130 and 8081 – used by the Flink service

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

2.5 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.

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.

If OpenStack was installed using the DevStack script, the easiest way to set the environment variables that authenticate the user is to source the `openrc` script from the `devstack` directory:

```
$ source devstack/openrc admin admin
```

2.5.1 Registering your ConPaaS image to OpenStack

The prebuilt ConPaaS images suitable to be used with OpenStack can be downloaded from the following links, depending on the virtualization technology and system architecture you are using:

ConPaaS VM image for OpenStack with KVM (x86_64):

http://www.conpaas.eu/dl/conpaas-2.1.0-openstack-kvm.img.tar.gz
MD5: 02ea7ef89ff81bf03668003a8d2feac6
size: 541 MB

ConPaaS VM image for OpenStack with LXC (x86_64):

http://www.conpaas.eu/dl/conpaas-2.1.0-openstack-lxc.img.tar.gz
MD5: f04e9dcc3059c5d10f599e72243055d3
size: 509 MB

ConPaaS VM image for OpenStack with LXC for the Raspberry Pi (arm):

http://www.conpaas.eu/dl/conpaas-2.1.0-rpi.img.tar.gz
MD5: 1eb2b8b369af3940460f85b12642f2e1
size: 528 MB

This section assumes that you already downloaded and decompressed one of the images above or created one 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> --disk-format raw --container-format bare --file <conpaa
```

2.5.2 Networking setup

ConPaaS requires instances to have public (floating) IP addresses assigned and will only communicate with an instance using its public IP address.

First, you need to make sure that floating addresses are configured. You can get a list containing all the configured floating IP addresses as follows:

```
$ nova floating-ip-bulk-list
```

If there are no addresses configured, you can add a new IP address range using the following command:

```
$ nova floating-ip-bulk-create --pool public --interface <interface> <new_range>
```

for example, using the **eth1** interface and the **172.16.0.224/27** address range:

```
$ nova floating-ip-bulk-create --pool public --interface eth1 172.16.0.224/27
```

Second, OpenStack should be configured to assign a floating IP address at every new instance creation. This can be done by adding the following line to the *[DEFAULT]* section of the nova configuration file (`/etc/nova/nova.conf`):

```
auto_assign_floating_ip = True
```

2.5.3 Security Group

As in the case of Amazon Web Services deployments, OpenStack deployments use security groups to limit 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*.

Your configured security groups can be found in Horizon by navigating to the *Project* > *Compute* > *Access & Security* menu in the left pane of the dashboard and then selecting the *Security Groups* tab.

Using the command line, the security groups can be listed using:

```
$ nova secgroup-list
```

You can use the `default` security group that is automatically created in every project. However note that, unless its default settings are changed, this security group denies all incoming traffic.

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.

2.5.4 SSH Key Pair

In order to use your OpenStack deployment with ConPaaS, you need to configure an SSH key pair that will allow you to login to an instance without using a password.

In Horizon, the key pairs can be found by navigating to the *Project* > *Compute* > *Access & Security* menu and then selecting the *Key Pairs* tab.

Using the command line, the key pairs can be listed using:

```
$ nova keypair-list
```

By default there is no key pair configured, so you should create a new one or import an already existing one.

2.5.5 Flavor

ConPaaS needs to know which instance type it can use, called *flavor* in OpenStack terminology. There are quite a few flavors configured by default, which can also be customized if needed.

The list of available flavors can be obtained in Horizon by navigating to the *Admin* > *System* > *Flavors* menu. Using the command line, the same result can be obtained using:

```
$ nova flavor-list
```

2.6 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 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.

The easiest way to try the Nutshell is to download the preassembled image for VirtualBox. This can be done from the following link:

VirtualBox VM containing ConPaaS in a Nutshell (2.5 GB):

http://www.conpaas.eu/dl/ConPaaS-Nutshell-2.1.0.ova
MD5: 9b01b07c1d0c45266345085cf65890ab

Warning: It is always a good idea to check the integrity of a downloaded image before continuing with the next step, as a corrupted image can lead to unexpected behavior. You can do this by comparing its MD5 hash with the one shown above. To obtain the MD5 hash, you can use the `md5sum` command.

Alternatively, you can also create such an image or a similar one that runs on standard clouds (OpenStack and Amazon EC2 are supported) by following the instructions in the Internals guide, section *Creating a Nutshell image*.

2.6.1 Running the Nutshell in VirtualBox

The easiest way to start the Nutshell is using VirtualBox.

As a lot of services run inside the Nutshell VM, it requires a significant amount of resources. The minimum requirements for a system to be able to run the Nutshell are as follows:

```
CPU: dual-core processor with hardware virtualization instructions
Memory: at least 6 GM of RAM (from which 3 GB should be allocated to the VM)
HDD: at least 30 GB of available space
```

The recommended system requirements for optimal performance:

```
CPU: Intel i7 processor or equivalent
Memory: at least 8 GB of RAM (from which 4 GB should be allocated to the VM)
HDD: Solid State Drive (SSD) with at least 30 GB of available space
```

Warning: It is highly advised to run the Nutshell on a system that meets the recommended system requirements, or else its performance may be severely impacted. For systems that do not meet the recommended requirements (but still meet the minimum requirements), a very careful split of the resources between the VM and the host system needs to be performed.

1. Make sure that hardware virtualization extensions are activated in your computer's BIOS. The procedure for activating them is highly dependent on your computer's manufacturer and model. Some general instructions can be found here:

   https://goo.gl/ZGxK9Z

2. If you haven't done this already, create a host-only network in VirtualBox. This is needed in order to allow access to the Nutshell VM and to the applications deployed in it from your host machine. To do so from the VirtualBox GUI, go to: *File* > *Preferences* > *Network* > *Host-only Networks*. Check if there is already a host-only network configured (usually called *vboxnet0*). If not, add one by clicking on the *Add host-only network* button.

3. Verify the settings of the host-only network. In the same window, select the host-only network (*vboxnet0*) and press the *Edit host-only network* button. In the *Adapter* tab, make sure that the following fields have these values:

```
IPv4 address: 192.168.56.1
IPv4 Network Mask: 255.255.255.0
```

   and in the *DHCP Server* tab:

```
Enable Server is checked
Server Address: 192.168.56.100
Server Mask: 255.255.255.0
Lower Address Bound: 192.168.56.101
Upper Address Bound: 192.168.56.254
```

   You can also use other values than the defaults presented above. In this case, note that you will also need to adjust the IP address range allocated by OpenStack to the containers to match your settings. You can do this by following the instructions from the following section of the User guide: *Changing the IP address space used by the Nutshell*.

4. Import the Nutshell appliance using the menu *File* > *Import Appliance*, or by simply double-clicking the *.ova* file in your file manager.

   Warning: Make sure you have enough free space on your hard drive before attempting this step as importing the appliance will extract the VM's hard disk image from the *.ova* archive, which occupies around 21 GB of hard disk space. Creating snapshots of the Nutshell VM will also require additional space, so for optimal operation, the recommended free space that should be available before importing the VM is 30 GB.

5. 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 VM, then following the menu: *Settings* > *System* > *Motherboard* / *Processor*. We recommend allocating at least 4 GB of RAM for the Nutshell to function properly. Make sure that enough memory remains for the host system to operate properly and never allocate more CPUs than what is available on your host computer.

6. It is also a very good idea to create a snapshot of the initial state of the Nutshell VM, immediately after it was imported. This allows the possibility to quickly revert to the initial state without importing the VM again when something goes wrong.

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

2.7 ConPaaS on Raspberry PI

ConPaaS on Raspberry PI is an extension to the ConPaaS project which uses one (or more) Raspberry PI(s) 2 or 3 Model B to create a cloud for deploying applications. Each Raspberry PI is configured as an OpenStack compute node (using LXC containers), running only the minimal number of OpenStack services required on such a node (`nova-compute` and `cinder-volume`). All the other OpenStack services, such as Glance, Keystone, Horizon etc., are moved outside of the PI, on a more powerful machine configured as an OpenStack controller node. The ConPaaS Director and both clients (command line and web frontend) also run on the controller node.

To ease the deployment of the system, we provide an image containing the raw contents of the Raspberry PI's SD card, along with a VirtualBox VM image (in the Open Virtualization Archive format) that contains the controller node and can be deployed on any machine connected to the same local network as the Raspberry PI(s). So, for a minimal working setup, you will need at least one Raspberry PI 2 or 3 Model B (equipped with a 32 GB SD card) and one laptop/desktop computer (with VirtualBox installed) that will host the backend VM. The two have to be connected to the same local network which, in the default configuration, uses IPs in the `172.16.0.0/24` range.

The two images can be downloaded from the following links:

RPI's SD card image (2.4 GB):

http://www.conpaas.eu/dl/ConPaaS-RPI-2.1.0-SDCard-32G.img.tar.gz
MD5: 6a8459104f8d10806b3a964bf21330c6

VirtualBox VM containing the backend server (2.7 GB):

http://www.conpaas.eu/dl/ConPaaS-RPI-2.1.0-Backend-VM.ova
MD5: b469741091f8e282b3e22635963f5783

> **Warning:** It is always a good idea to check the integrity of a downloaded image before continuing with the next steps, as a corrupted image can lead to unexpected behavior. You can do this by comparing its MD5 hash with the ones shown above. To obtain the MD5 hash, you can use the `md5sum` command.

2.7.1 Installing the image on the Raspberry PI

You need to write the image to the Raspberry PI's SD card on a different machine (equipped with an SD card reader) and then move the SD card back into the Raspberry PI.

Download and decompress the image, then write it to the SD card using the *dd* utility. You can follow the official instructions from the RaspberryPi.org website:

Linux: https://www.raspberrypi.org/documentation/installation/installing-images/linux.md

MacOS: https://www.raspberrypi.org/documentation/installation/installing-images/mac.md

> **Warning:** Decompressing the image will result in a 32 GB file (the raw SD card image), so please make sure that you have enough free space before attempting this step.

> **Warning:** Before writing the image, please make sure that the SD card has a capacity of at least 31998345216 bytes.

The image was designed to fit the majority of the 32 GB SD cards, as the actual size varies between manufacturers. As a result, its size may be a little lower than the actual size of your card, leaving some unused space near the end of the card. A lot more unused space remains if a bigger SD card (64 GB) is used. To recover this wasted space, you may adjust the partitions by moving the swap partition near the end of the card and expanding the main *ext4* partition.

> **Warning:** If you adjust the partitions, please make sure that the beginning of every partition remains aligned on a 4 MB boundary (the usual size of the SD card's erase block) or else performance may be negatively affected.

2.7.2 Deploying the Backend VM

Download the *.ova* file and import it into VirtualBox. In a graphical environment, you can usually do this by double-clicking the *.ova* file.

Adjust the resources allocated to the VM. Although the default settings use a pretty generous amount of resources (4 CPUs and 4 GB of RAM), reducing this to a less powerful configuration should work fine (for example 1 CPU and 2 GB of RAM).

Another very important configuration is setting the VM's network interfaces. Two interfaces should be present: the first one (called *eth0* inside the VM) should be configured as the *NAT* type to allow Internet access to the VM. The second interface (*eth1* inside the VM) should be bridged to an adapter connected to the same local network as the Raspberry PI, so in the VM's properties select *Bridged adapter* and choose the interface to which the Raspberry PIs are connected.

For more information regarding the usage of ConPaaS on Raspberry PI, please consult the *ConPaaS on Raspberry PI* section in the user 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;
- **XtreemFS service** offering a distributed and replicated file system;
- **Flink service** offering distributed stream and batch data processing;
- **Generic service** allowing the execution of arbitrary applications.

ConPaaS applications can be composed of any number of services. For example, an application can use a Web hosting service, a database service to store the internal application state and a file storage service to store access logs.

3.1 Usage overview

3.1.1 Web-based interface

Most operations in ConPaaS can be performed using the ConPaaS frontend, which provides a Web-based interface to the system. The frontend allows users to register, create and start applications, add and start services, upload code and configure each service.

- The Dashboard page displays the list of applications created by the current user.
- Each application has a separate page which allows the user to add or remove services.
- Each service has its own separate page which allows the user to configure it, upload code and data, and scale it up and down.

The ConPaaS front-end provides simple and intuitive interfaces for controlling applications and services. We describe here the most common features. Please refer to the next sections for service-specific functionality.

Create an application. In the main ConPaaS Dashboard, click on "create new application". Write the application's name and click on "OK". The application's name must be unique.

Start an application. From the main ConPaaS Dashboard, click on the desired application. To start the application, click on the "start application" button. This will create a new virtual machine which hosts the Application's Manager, a ConPaaS component in charge of managing the entire application.

Add a service. Click on "add new service", then select the service you want to add. This operation adds extra functionalities to the application manager which are specific to a certain service. These functionalities enable the application manager to be 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 the newly created service, then click on the "start" button. 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 service anymore, click on the "stop" button to stop the service. This stops all instances of the service.

Remove the service. Click "remove" to delete the service. At this point, all the state of the service manager is lost.

Stop the application. Even when all the services from an application are stopped, the application's manager will keep on running. To stop it, in the application's page, click on the "stop application" button. The application will not use resources anymore.

3.1.2 Command line interface

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 frontend.

The new command line client for ConPaaS is called `cps-tools`.

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

Command arguments:

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

Create an application:

```
$ cps-tools application create <appl_name>
$ cps-application create <appl_name>
```

List applications:

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

Start an application:

```
$ cps-tools application start <app_name_or_id>
$ cps-application start <app_name_or_id>
```

Available service types:

```
$ cps-tools service get_types
$ cps-service get_types
```

Add a service to an application:

```
$ cps-tools service add <service_type> <app_name_or_id>
$ cps-tools <service_type> add <app_name_or_id>
$ cps-<service_type> add <app_name_or_id>
```

List services:

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

Start a service:

```
$ cps-tools <service_type> start <app_name_or_id> <serv_name_or_id>
$ cps-service start <app_name_or_id> <serv_name_or_id>
$ cps-<service_type> start <app_name_or_id> <serv_name_or_id>
```

Service command specific arguments:

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

Scale the service up and down:

```
$ cps-service add_nodes <app_name_or_id> <serv_name_or_id>
$ cps-service remove_nodes <app_name_or_id> <serv_name_or_id>
```

List the available clouds:

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

3.1.3 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 you additional credit.

Note that every running application consumes credit, even if all its services are in the "stopped" state. The reason is that the application still has one "Application Manager" instance running. To stop using any credits you must also stop all your applications.

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 of application with id 1 using the command line tool:

```
$ cps-php upload_code 1 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:

```
$ cps-php upload_key <app_name_or_id> <serv_name_or_id> <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. There is only one git repository per application, so you only need to upload your SSH key once.

Below the area for entering the SSH key, the frontend will show the `git` command to be executed in order to obtain a copy of the repository. As there is only a single repository for all the services running inside an application, **the code that belongs to a specific service has to be placed in a directory with the name identical to the service id**, which has to be created by the user. The repository itself can then be used as usual. A new version of your application can be uploaded with `git push`.

```
user@host:~/code$ mkdir 1
user@host:~/code$ vi 1/index.php
user@host:~/code$ git add 1/index.php
user@host:~/code$ git commit -am "New index.php version for service 1"
user@host:~/code$ git push origin master
```

Warning: Do not forget to place the code belonging to a service in a directory with the name identical to the service id, or else the service will be unable to find the files.

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.

```
$ cps-php debug <app_name_or_id> <serv_name_or_id> <on | off>
```

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 the command line, the `add_nodes` subcommand can be used to add additional nodes to a service. It takes as arguments the number of backend nodes, web nodes and proxy nodes to add:

```
$ cps-php add_nodes <app_name_or_id> <serv_name_or_id> --backend COUNT --proxy COUNT --web COUNT
```

For example, adding two backend nodes to PHP service id 1 of application 1:

```
$ cps-php add_nodes 1 1 -- backend 2
```

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

```
$ cps-php add_nodes 1 1 --backend 1 --web 1 --cloud mycloud
```

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

```
$ cps-php remove_nodes 1 1 --backend 1
```

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.

3.4 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 command line tool:

```
$ cps-java upload_code <app_name_or_id> <serv_name_or_id> <path/to/archive.war>
```

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, the replication system being 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 better performance from a cluster, it is a best practice to use it in a balanced fashion, so that each node has approximately the same load of the others. To achieve this, the service allows users to allocate special load balancer nodes (`glb`) 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 behavior. 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 this information 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.

- "Galera Mean Misalignment" draws a real-time measure of the mean misalignment across the nodes. This information is derived from 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 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 metadata. 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.

3.6.1 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 allow performing administrative file-systems tasks when used 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:

```
$ cps-xtreemfs get_client_cert <app_name_or_id> <serv_name_or_id> <passphrase> <adminflag> <filena
$ cps-xtreemfs get_user_cert <app_name_or_id> <serv_name_or_id> <user> <group> <passphrase> <admi
```

3.6.2 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 the 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.

3.6.3 Policies

Different aspects of XtreemFS (e.g. replica- and OSD-selection) can be customized 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`:

```
$ cps-xtreemfs list_policies <app_name_or_id> <serv_name_or_id> <policy_type>
$ cps-xtreemfs set_policy <app_name_or_id> <serv_name_or_id> <policy_type> <policy> <volume>
```

3.6.4 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.

3.7 The Flink service

The Flink service facilitates the deployment of applications that use the Apache Flink platform for distributed stream and batch data processing. Flink provides data distribution, communication, and fault tolerance for distributed computations over data streams. Flink also supports batch processing applications, treated as special cases of stream processing.

A Flink node can assume two possible roles: **master** or **worker**. A master (also called *JobManager*) coordinates the distributed execution. It schedules tasks, coordinates checkpoints and recovery on failures, etc. The workers (also called *TaskManagers*) execute the tasks of a dataflow, and buffer and exchange the data streams. A Flink service will always have exactly one master and one or more workers. The first instance that is started by the service will always assume both roles (master and worker). All the other instances will be considered as worker nodes.

3.7.1 Running a Flink application

After a Flink service is started, the user can access the Flink Dashboard using the link that appears in the upper-right corner of the service page. From the dashboard, the Flink deployment can be used in the same way as a regular installation.

As an example, we will illustrate how to upload and execute the WordCount sample application. The **jar** containing the application can be found in the *examples* directory inside the Flink binary package, which can be downloaded from the official website:

http://www-eu.apache.org/dist/flink/flink-1.1.1/flink-1.1.1-bin-hadoop1-scala_2.10.tgz

Start a Flink service, wait for it to become ready and access the Flink Dashboard using the link that appears in the upper-right corner of the service page. Go to the "Submit new Job" page and click on the "Add new" button. Select the *WordCount.jar* file and click on "Upload". Tick the box in front of the job's name. To be able to see the output, redirect it to the TaskManager's output file by entering the following text in the "Program Arguments" field:

```
--output file:///var/cache/cpsagent/flink-taskmanager.log
```

To run the job, press the "Submit" button. You will be shown a page where you can monitor the progress of your job. When the job finished execution, you can check the output by accessing the "TaskManager out" link from the Flink service's page.

For more information on using Flink, please consult the official documentation: https://ci.apache.org/projects/flink/flink-docs-release-1.1/

3.8 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.

3.8.1 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.

3.8.2 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.

3.8.3 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 command line tool:

```
$ cps-generic upload_code 1 1 test-code.tar.gz
Code version code-pw1LKs uploaded
$ cps-generic enable_code 1 1 code-pw1LKs
code-pw1LKs enabled
$ cps-generic list_codes 1 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:

```
$ cps-generic download_code <app_name_or_id> <serv_name_or_id> --version <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:

```
$ cps-generic delete_code <app_name_or_id> <serv_name_or_id> <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.

3.8.4 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:

```
$ cps-generic upload_key <app_name_or_id> <serv_name_or_id> <filename>
```

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

```
$ cps-generic list_keys <app_name_or_id> <serv_name_or_id>
```

There is only one git repository per application, so you only need to upload your SSH key once.

After 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
```

As there is only a single repository for all the services running inside an application, **the code that belongs to a specific service has to be placed in a directory with the name identical to the service id**, which has to be created by the user. The repository itself can then be used as usual. A new version of your application can be uploaded with `git push`:

```
$ cd code
$ mkdir 1
$ <create the scripts in this directory>
$ git add 1/{init,notify,run,interrupt,cleanup}.sh
$ git commit -m "New code version"
$ git push origin master
```

Warning: Do not forget to place the code belonging to a service in a directory with the name identical to the service id, or else the service will be unable to find the files.

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:

```
$ cps-generic list_codes 1 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.

3.8.5 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:

```
$ cps-generic create_volume <app_name_or_id> <vol_name> <vol_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:

```
$ cps-generic list_nodes <app_name_or_id> <serv_name_or_id>
```

The list of all storage volumes can be retrieved with:

```
$ cps-generic list_volumes <app_name_or_id> <serv_name_or_id>
```

This command detaches and deletes a storage volume:

```
$ cps-generic delete_volume <app_name_or_id> <agent_id>
```

3.8.6 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:

```
$ cps-generic add_nodes <app_name_or_id> <serv_name_or_id> --count <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:

```
$ cps-generic remove_nodes <app_name_or_id> <serv_name_or_id> --count <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 allows 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:

```
$ cps-generic run <app_name_or_id> <serv_name_or_id> -p <parameters>
$ cps-generic interrupt <app_name_or_id> <serv_name_or_id> -p <parameters>
$ cps-generic cleanup <app_name_or_id> <serv_name_or_id> -p <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 renaming 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.

3.8.7 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:

```
$ cps-generic get_script_status <app_name_or_id> <serv_name_or_id>
```

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 commands:

```
$ cps-generic get_agent_log <app_name_or_id> <serv_name_or_id> <agent_id>
$ cps-generic get_agent_log <app_name_or_id> <serv_name_or_id> <agent_id> -f agent.out
$ cps-generic get_agent_log <app_name_or_id> <serv_name_or_id> <agent_id> -f agent.err
```

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

```
$ cps-generic list_nodes <app_name_or_id> <serv_name_or_id>
```

3.9 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.

3.9.1 Starting the Nutshell

In this section, we assume that the Nutshell is already installed into VirtualBox according to the instructions in the Installation guide. If this is not the case, you may want to check these instructions first: *ConPaaS in a Nutshell*.

1. Open VirtualBox and start the Nutshell VM by selecting it from the list on the left side and then clicking the *Start* button.
2. Wait for the Nutshell VM to finish booting. Depending on your computer's hardware configuration, this process may take a few minutes. Any messages that may appear in the VM window at this stage are usually harmless debug messages which can be ignored.
3. When the login prompt appears, the Nutshell VM is ready to be used.

3.9.2 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://192.168.56.2
```

> **Warning:** The first time you access the web frontend, a security warning will appear, stating that the SSL certificate of the website is invalid. This is normal, as the certificate is self-signed. To proceed further, you need to confirm that you want to continue anyway. The procedure is different depending on your web browser.

Note that the frontend is accessible only from your local machine. Other machines will not be able to access it. A default ConPaaS 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 want to have a look under the hood, Horizon can be reached (using HTTP, not HTTPS) at the same IP address:

```
http://192.168.56.2
```

The credentials for Horizon are:

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

3.9.3 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 (the preferred method):

```
$ ssh stack@192.168.56.2
```

The login credentials are:

```
Username: stack
Password: conpaas
```

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

```
$ cps-tools application create "First app"
$ cps-tools application start 1
$ cps-tools service add helloworld 1
$ cps-tools service start 1 1
```

OpenStack commands are also available. For example:

```
$ nova list
```

lists all the active instances and:

```
$ cinder list
```

lists all the existing storage 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.

3.9.4 Changing the IP address space used by the Nutshell

In the standard configuration, the Nutshell VM is assigned the static IP address `192.168.56.2`, part of the `192.168.56.0/24` subnet that is used by the host-only network of VirtualBox. ConPaaS services running inside the Nutshell VM also need to have IP addresses assigned, one for each container that is started inside the Nutshell VM. This is done using OpenStack's floating IP mechanism, which is configured to use an IP range from `192.168.56.10` to `192.168.56.99`, part of the same `192.168.56.0/24` subnet.

This configuration was carefully chosen to not overlap with the pool used by the DHCP server of the host-only network of VirtualBox which, in the default settings, uses a range from `192.168.56.101` to `192.168.56.254`. To check the range that is used in your system, you can navigate in the VirtualBox window to the following menu: *File* > *Preferences* > *Network* > *Host-only Networks*. Select the *vboxnet0* network and click on the *Edit host-only network* button and then *DHCP server*.

To modify the IP address range used by the Nutshell VM, you need to change the static address assigned to the Nutshell VM itself and also the IP range used by OpenStack to assign floating IP addresses to the containers. You need to make sure that all these addresses are part of the subnet used by the host-only network of VirtualBox and also that they do not overlap with this network's DHCP server pool (in the case other VMs with interfaces in the host-only network are started and receive addresses from this pool). You may need to adjust the host-only network's configuration in VirtualBox for this these conditions to be met.

The static IP address of the Nutshell VM can be changed by editing the `/etc/network/interfaces` file. The interface that is part of the host-only network is the second one (`eth1`), this is the one that should have the IP assigned. The first one (`eth0`) is only used to provide Internet access to the Nutshell VM.

To modify the IP range used to assign floating IP addresses to containers, 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 eth1 <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
```

3.9.5 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. As in the previous scenario, the Nutshell's IP address can be configured by editing the `/etc/network/interfaces` file. If the Nutshell is publicly accessible, you may want to make sure that tighter security is implemented: the default user for the ConPaaS frontend should be removed and access to SSH and OpenStack dashboard should be blocked.

3.10 ConPaaS on Raspberry PI

The following ConPaaS services are supported on the Raspberry PI version of ConPaaS:

- **php**: PHP version 5.6 with Nginx
- **java**: Apache Tomcat 7.0 servlet container
- **xtreemfs**: XtreemFS-1.5 distributed file system
- **flink**: Apache Flink 1.1 stream and batch data processing
- **generic**: deployment of arbitrary server-side applications

For instructions on how to install the Raspberry PI version of ConPaaS, please refer to the relevant section in the Installation guide: *ConPaaS on Raspberry PI*.

3.10.1 Access credentials

Backend VM:

```
IP address: 172.16.0.1
user: stack
password: raspberry
```

For OpenStack's dashboard (Horizon):

```
URL: http://172.16.0.1/
user: admin
password: password
```

For the ConPaaS web frontend:

```
URL: https://172.16.0.1/
user: test
password: password
```

Raspberry PI:

```
IP address: 172.16.0.11
user: pi
password: raspberry
```

Containers deployed on the Raspberry PI:

```
IP addresses (public): between 172.16.0.225 and 172.16.0.254
IP addresses (private): between 172.16.0.32 and 172.16.0.61
user: root
password: conpaas
```

3.10.2 Networking setup

IP addresses on the Raspberry PI and backend VM are already configured, all in the `172.16.0.0/24` range. The Raspberry PI is also configured to accept a secondary IP address using DHCP. If this is available, it will use it for Internet access. If not, it will route the Internet traffic through the backend VM. Everything is already configured, no other configurations are needed. In principle there is no need to have Internet access on the PI (if the hosted application does not require it), however note that in this case you will need to manually set the correct time on the Raspberry PI after every reboot, or else the SSL certificates-based authentication in ConPaaS will fail.

If another device has to take part in this local network (for example to allow it to easily ssh into the different components of the system, or for the clients of the application hosted on the Raspberry PIs), you can use any IP in that range that does not collide with the ones used by the components listed above. For example, additional servers can have IP addresses between `172.16.0.2` and `172.16.0.10`, additional Raspberry PIs can use IPs between `172.16.0.12` and `172.16.0.31`, clients can use IPs between `172.16.0.200` and `172.16.0.223`. The ranges `172.16.0.64/26`, `172.16.0.128/26` are also completely free.

The system was designed to allow connecting the components using an already-existing local network that you may have, without interfering too much with it. That's why it does not run by default a DHCP server to automatically allocate IPs to other machines that get connected to this network. On the other hand, this means that you will need to manually add an IP address to any other machine that needs to take part in this network. This address can be added as a secondary IP address, besides the usual address that your device has, if using an already-existing network. For example, in order to access the system from the laptop that hosts the backend VM, another IP address from the `172.16.0.0/24` range needs to be assigned as the secondary address to the *eth0* interface of this laptop.

3.10.3 Usage example

Here follows an usage example in which we create and start a new Generic Service using the command line tools. The same outcome can also be achieved using the graphical frontend, which can be accessed using the backend VM's IP address (note that the protocol should be **HTTPS**, not HTTP): https://172.16.0.1/

1. Start the Backend VM. Start the Raspberry PI. Allow them some time to finish booting.
2. Make sure the time is synchronized between the Raspberry PI and the Backend VM. This step is crucial in order to allow the SSL certificates-based authentication in ConPaaS to succeed. As the Raspberry PI does not have an internal battery to keep the time when powered off, it relies on the NTP protocol to set its time. If there is no Internet connectivity or updating the time through NTP fails, the correct time will have to be set manually using the `date` command after every reboot.
3. Check that the OpenStack services are up and running. On the backend server, run the following command:

```
stack@nutshell:~$ nova-manage service list
[... debugging output omitted ...]
Binary           Host                                 Zone             Status     State Updat
nova-conductor   nutshell                             internal         enabled    :-)   2015-
nova-cert        nutshell                             internal         enabled    :-)   2015-
nova-scheduler   nutshell                             internal         enabled    :-)   2015-
nova-consoleauth nutshell                             internal         enabled    :-)   2015-
nova-compute     raspberrypi                          nova             enabled    :-)   2015-
nova-network     nutshell                             internal         enabled    :-)   2015-
```

   As in the example above, you should see 6 `nova` services running, all of them should be up (smiley faces). Pay extra attention to the `nova-compute` service, which is running on the Raspberry PI, and may become ready a little later than the others.

   Do not proceed further if any service is down.

4. Create a new application using ConPaaS:

```
stack@nutshell:~$ cps-tools application create "Test application"
Application 'Test application' created with id 1.
```

5. Start the application. This will start a new container for the Application Manager:

```
stack@nutshell:~$ time cps-tools application start 1
Application 'Test application' with id 1 is starting...  done.

real       2m04.515s
user       0m0.704s
sys        0m0.152s
```

This step should take around 2-3 minutes. During this time, the first container is created and the ConPaaS Application Manager is started and initialized.

Check that the container is up and running with `nova list`:

```
stack@nutshell:~$ nova list
+--------------------------------------+-----------------------+--------+------------+-------
| ID                                   | Name                  | Status | Task State | Power
+--------------------------------------+-----------------------+--------+------------+-------
| 3c5c3375-1e73-4e0a-b6cc-223460c726e0 | conpaas-rpi-u1-a1-mgr | ACTIVE | -          | Runnir
+--------------------------------------+-----------------------+--------+------------+-------
```

6. Add a Generic service to the application:

stack@nutshell:~$ cps-tools service add generic 1 Service generic successfully added to application 1 with id 1.

7. Start the newly added service. This will start the second container on the Raspberry PI in which the first ConPaaS agent can host an application:

```
stack@nutshell:~$ time cps-tools service start 1 1
Service 1 is starting...

real       1m02.043s
user       0m4.948s
sys        0m1.384s
```

This step should take around 1-2 minutes. During this time, the second container is created and the ConPaaS Agent is started and initialized.

8. Find out the IP address of the newly started container:

```
stack@nutshell:~$ cps-tools generic list_nodes 1 1
aid sid role   ip           agent_id       cloud
-------------------------------------------------
  1   1 master 172.16.0.226 iaasi-00000012 default
```

You can also determine the IP addresses of the containers with `nova list`:

```
stack@nutshell:~$ nova list
+--------------------------------------+-------------------------------+--------+------------+
| ID                                   | Name                          | Status | Task State |
+--------------------------------------+-------------------------------+--------+------------+
| 3c5c3375-1e73-4e0a-b6cc-223460c726e0 | conpaas-rpi-u1-a1-mgr         | ACTIVE | -          |
| 2a1d758d-5300-4d7f-8ba2-4f1499838a7d | conpaas-rpi-u1-a1-s1-generic | ACTIVE | -          |
+--------------------------------------+-------------------------------+--------+------------+
```

9. Log on to the container and check that the ConPaaS Agent is running correctly (the default script just prints some information):

```
stack@nutshell:~$ ssh root@172.16.0.226
root@172.16.0.226's password: [conpaas]
Linux conpaas 4.1.12-v7+ #824 SMP PREEMPT Wed Oct 28 16:46:35 GMT 2015 armv7l
[... welcome message omitted ...]
root@server-2a1d758d-5300-4d7f-8ba2-4f1499838a7d:~# cat generic.out
Sun Nov  8 16:21:21 UTC 2015
Executing script init.sh
```

```
Parameters (0):
My IP is 172.16.0.226
My role is master
My master IP is 172.16.0.226
Information about other agents is stored at /var/cache/cpsagent/agents.json
[{"ip": "172.16.0.226", "role": "master", "id": "iaasi-00000012"}]
```

If the output looks like in the example above, everything is running smoothly!

For more information on the Generic service, please refer to section *The Generic service*.

10. Do not forget to delete the service after you're done with it:

```
stack@nutshell:~$ cps-tools service remove 1 1
Service 1 of application 1 has been successfully removed.
```

CHAPTER 4

Internals

4.1 Introduction

A ConPaaS application represents a collection of ConPaaS services working together. The application manager is a process that resides in the first VM that is created when the application is started and is in charge of managing the entire application. The application manager represents the single control point for the entire application.

A ConPaaS service consists of three main entities: the service manager, the service agent, and the web frontend. The service manager is a component that supplements the application manager with service-specific functionality. 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. A service agent resides on each of the VMs that are started by the service. The agents are the ones doing all the work.

To implement a new ConPaaS service, you must provide a new service manager, a new service agent and a new service frontend (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 the common functionality of the ConPaaS services. So far, the framework provides abstractions for the IaaS layer (adding support for a new cloud provider should not require modifications in any ConPaaS service implementation) and it also provides abstractions 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 base classes for service managers and agents, 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
|   |   |   |   |-- openstack.py
|   |   |   |-- agent.py
|   |   |   |-- callbacker.py
|   |   |   |-- expose.py
|   |   |   |-- file.py
|   |   |   |-- ganglia.py
|   |   |   |-- git.py
|   |   |   |-- https
```

```
|   |   |   |-- log.py
|   |   |   |-- manager.py
|   |   |   |-- misc.py
|   |   |   |-- node.py
|   |   |   |-- services.py
|   |   |-- services
|   |   |       |-- flink/
|   |   |       |-- generic/
|   |   |       |-- helloworld/
|   |   |       |-- mysql/
|   |   |       |-- webservers/
|   |   |       |-- xtreemfs/
|   |-- setup.py
|-- config
|-- contrib
|-- misc
|-- sbin
|-- scripts
```

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

4.2 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 *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 prints a line of text in the agent's log file. This method could be used, for example, to make some initializations in the agent.

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
from conpaas.core.misc import check_arguments

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):
        try:
```

```
            exp_params = []
            check_arguments(exp_params, kwargs)
        except Exception as ex:
            return HttpErrorResponse("%s" % ex)

        self.logger.info('Agent started up')
        return HttpJsonResponse()

    @expose('GET')
    def get_helloworld(self, kwargs):
        try:
            exp_params = []
            check_arguments(exp_params, kwargs)
        except Exception as ex:
            return HttpErrorResponse("%s" % ex)

        self.logger.info('Agent returned hello')
        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 service 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) A service manager

supplements the application manager with service-specific functionality. It does so by overriding the methods inherited from the base manager class. These methods will be called by the application manager when the corresponding event occurs. For example, *on_start* is called immediately after the service is started, *on_add_nodes* after additional nodes have been added to the service, *on_remove_nodes* after nodes have been removed, *on_stop* after the service was stopped.

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

from conpaas.core.misc import check_arguments, is_in_list, is_not_in_list,\
    is_list, is_non_empty_list, is_list_dict, is_list_dict2, is_string,\
    is_int, is_pos_nul_int, is_pos_int, is_dict, is_dict2, is_bool,\
    is_uploaded_file

class HelloWorldManager(BaseManager):

    def __init__(self, config_parser, **kwargs):
        BaseManager.__init__(self, config_parser)
        self.state = self.S_INIT

    def get_service_type(self):
        return 'helloworld'

    def get_context_replacement(self):
        return dict(STRING='helloworld')

    def on_start(self, nodes):
        return self.on_new_nodes(nodes)

    def on_stop(self):
        self.logger.info("Removing nodes: %s" %[ node.id for node in self.nodes ])
        return self.nodes[:]

    def on_add_nodes(self, nodes):
        return self.on_new_nodes(nodes)

    def on_remove_nodes(self, node_roles):
        count = sum(node_roles.values())
        del_nodes = []
        cp_nodes = self.nodes[:]
        for _ in range(0, count):
            node = cp_nodes.pop()
            del_nodes += [ node ]
        if not cp_nodes:
            self.state = self.S_STOPPED
        else:
            self.state = self.S_RUNNING

        self.logger.info("Removing nodes: %s" %[ node.id for node in del_nodes ])
        return del_nodes

    def on_new_nodes(self, nodes):
        try:
            for node in nodes:
                client.startup(node.ip, self.AGENT_PORT)
```

```
            return True
        except Exception, err:
            self.logger.exception('_do_startup: Failed to create node: %s' % err)
            return False

    @expose('GET')
    def list_nodes(self, kwargs):
        try:
            exp_params = []
            check_arguments(exp_params, kwargs)
        except Exception as ex:
            return HttpErrorResponse("%s" % ex)

        try:
            self.check_state([self.S_RUNNING, self.S_ADAPTING])
        except:
            return HttpJsonResponse({})

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

    @expose('GET')
    def get_service_info(self, kwargs):
        try:
            exp_params = []
            check_arguments(exp_params, kwargs)
        except Exception as ex:
            return HttpErrorResponse("%s" % ex)

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

    @expose('GET')
    def get_node_info(self, kwargs):
        node_ids = [ str(node.id) for node in self.nodes ]
        exp_params = [('serviceNodeId', is_in_list(node_ids))]
        try:
            serviceNodeId = check_arguments(exp_params, kwargs)
        except Exception as ex:
            return HttpErrorResponse("%s" % ex)

        for node in self.nodes:
            if serviceNodeId == node.id:
                serviceNode = node
                break

        return HttpJsonResponse({
            'serviceNode': {
                            'id': serviceNode.id,
                            'ip': serviceNode.ip,
                            'vmid': serviceNode.vmid,
                            'cloud': serviceNode.cloud_name,
                            'role': serviceNode.role,
                            'logs': self.get_role_logs(serviceNode.role)
                            }
            })

    @expose('GET')
    def get_helloworld(self, kwargs):
        try:
            exp_params = []
            check_arguments(exp_params, kwargs)
            self.check_state([self.S_RUNNING])
```

```
        except Exception as ex:
            return HttpErrorResponse("%s" % ex)

        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) })
```

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) 2010-2016 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'
                    'helloworld' : {'class' : 'HelloWorldManager',
                                    'module': 'conpaas.services.helloworld.manager.manager'},
                    'xtreemfs'   : {'class' : 'XtreemFSManager',
                                    'module': 'conpaas.services.xtreemfs.manager.manager'},
                    'mysql'      : {'class' : 'MySQLManager',
                                    'module': 'conpaas.services.mysql.manager.manager'},
                    'flink'      : {'class' : 'FlinkManager',
                                    'module': 'conpaas.services.flink.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 installing n
}

agent_services = {'web'        : {'class' : 'WebServersAgent',
                                  'module': 'conpaas.services.webservers.agent.internals'},
                  'helloworld' : {'class' : 'HelloWorldAgent',
                                  'module': 'conpaas.services.helloworld.agent.agent'},
                  'xtreemfs'   : {'class' : 'XtreemFSAgent',
                                  'module': 'conpaas.services.xtreemfs.agent.agent'},
                  'mysql'      : {'class' : 'MySQLAgent',
                                  'module': 'conpaas.services.mysql.agent.internals'},
                  'flink'      : {'class' : 'FlinkAgent',
                                  'module': 'conpaas.services.flink.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 installing new
}
```

4.3 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 to create the web frontend page for a service.

4.3.1 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.

4.3.2 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.

4.3.3 Files to be added

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

4.4 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 images that contain the dependencies for *all* available services. If you intend to use these images and do not need a specialized VM image, then you can skip this section.

4.4.1 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 OpenStack. 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.

4.4.2 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.

4.4.3 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 the `df -h` command. 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/pts /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.

4.5 Creating a Nutshell image

Starting with the release 1.4.1, ConPaaS is shipped together with a VirtualBox appliance containing the Nutshell VM image. This section explains how to create a similar image that can be deployed on a different virtualization technology (such as the other clouds supported by ConPaaS). The next section describes the procedure for recreating the VirtualBox image. If you are interested only in installing the standard VirtualBox image that is shipped with ConPaaS, you may skip this chapter entirely and only read the installation guide available here: *ConPaaS in a Nutshell*.

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 LXC 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.

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

4.5.1 Creating a Nutshell image for VirtualBox

As mentioned earlier the Nutshell VM can also run on VirtualBox. In order to generate a Nutshell image compatible with VirtualBox, you have to set the *cloud* value to *vbox* in 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 4 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*.

4.6 Preinstalling an application into a ConPaaS Services Image

A ConPaaS Services Image contains all the necessary components needed in order to run the ConPaaS services. For deploying arbitrary applications using ConPaaS, *The Generic service* provides a mechanism to install and run the application, along with its dependencies. The installation, however, has to happen during the initialization of every new node that is started, for example in the `init.sh` script of the Generic Service. If installing the application with its dependencies takes a long time or, in general, is not desired to happen during every deployment of a new node, another option is available: preinstalling the application inside the ConPaaS Services Image. The current section describes this process.

1. Download a ConPaaS Services Image appropriate for your computer architecture and virtualization technology. Here are the download links for the latest images:

   ConPaaS VM image for Amazon EC2 (x86_64):

   http://www.conpaas.eu/dl/conpaas-2.1.0-amazon.img.tar.gz
   MD5: 2d2af415a29c8413b898eacfcc2f1343
   size: 541 MB

   ConPaaS VM image for OpenStack with KVM (x86_64):

   http://www.conpaas.eu/dl/conpaas-2.1.0-openstack-kvm.img.tar.gz
   MD5: 02ea7ef89ff81bf03668003a8d2feac6
   size: 541 MB

   ConPaaS VM image for OpenStack with LXC (x86_64):

   http://www.conpaas.eu/dl/conpaas-2.1.0-openstack-lxc.img.tar.gz
   MD5: f04e9dcc3059c5d10f599e72243055d3
   size: 509 MB

   ConPaaS VM image for OpenStack with LXC for the Raspberry Pi (arm):

   http://www.conpaas.eu/dl/conpaas-2.1.0-rpi.img.tar.gz
   MD5: 1eb2b8b369af3940460f85b12642f2e1
   size: 528 MB

   Warning: If you choose to use one of the images above, it is always a good idea to check its integrity before continuing to the next step. A corrupt image may result in unexpected behavior which may be hard to trace. You can check the integrity by verifying the MD5 hash with the `md5sum` command.

   Alternatively, you can also create one such image using the instructions provided in the section *Creating A ConPaaS Services VM Image*.

   The following steps will use as an example the image for the Raspberry PI. For other architecture or virtualization technologies, the commands are the same.

> **Warning:** The following steps need to be performed on a machine with the same architecture and a similar operating system. For the regular images, this means the 64-bit version of a Debian or Ubuntu system. For the Raspberry PI image, the steps need to be performed on the Raspberry PI itself (with a Raspbian installation, arm architecture). Trying to customize the Raspberry PI image on an x86 system will not work!

2. Log in as **root** and change to the directory where you downloaded the image.
3. Decompress the downloaded image:

```
root@raspberrypi:/home/pi# tar xaf conpaas-rpi.img.tar.gz
```

4. (Optional) If you need to expand the size of the image, you can do it right now. As the image is in the raw format, expanding the size can be done by increasing the size of the image file. For example, to increase the size with 1 GB:

```
root@raspberrypi:/home/pi# dd if=/dev/zero bs=4M count=256 >> conpaas-rpi.img
256+0 records in
256+0 records out
1073741824 bytes (1.1 GB) copied, 56.05551 s, 19 MB/s
```

   If you have the package `qemu-utils` installed, you can also use `qemu-img` instead:

```
root@raspberrypi:/home/pi# qemu-img resize conpaas-rpi.img +1G
Image resized.
```

5. Map a loop device to the ConPaaS image:

```
root@raspberrypi:/home/pi# losetup -fv conpaas-rpi.img
Loop device is /dev/loop0
```

   > **Warning:** If you already have other loop devices in use, the output of this command may contain a different loop device. Take a note of it and replace *loop0* with the correct device in the following commands.

6. If you increased the size of the image in step 3, you now need to also expand the file system. First, check the integrity of the filesystem with the following command:

```
root@raspberrypi:/home/pi# e2fsck -f /dev/loop0
e2fsck 1.42.9 (4-Feb-2014)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
root: 44283/117840 files (9.1% non-contiguous), 409442/470528 blocks
```

   You can now expand the file system:

```
root@raspberrypi:/home/pi# resize2fs /dev/loop0
resize2fs 1.42.9 (4-Feb-2014)
Resizing the filesystem on /dev/loop0 to 732672 (4k) blocks.
The filesystem on /dev/loop0 is now 732672 blocks long.
```

7. Create a new directory and mount the image to it:

```
root@raspberrypi:/home/pi# mkdir conpaas-img
root@raspberrypi:/home/pi# mount /dev/loop0 conpaas-img/
```

   Now you can access the contents of the image inside the `conpaas-img` directory.

8. Copy your application's binaries and any other static content that you want to include in the image somewhere under the `conpaas-img` directory.

9. To install any prerequisites, you may want to change the root directory to `conpaas-img`. But first, you will need to mount `/dev`, `/dev/pts` and `/proc` in the `conpaas-img` directory (which will become the new root directory), or else the installation of some packages may fail:

```
root@raspberrypi:/home/pi# mount -obind /dev conpaas-img/dev
root@raspberrypi:/home/pi# mount -obind /dev/pts conpaas-img/dev/pts
root@raspberrypi:/home/pi# mount -t proc proc conpaas-img/proc
```

10. You can now execute the chroot:

```
root@raspberrypi:/home/pi# chroot conpaas-img
```

    Your root directory is now the root of the image.

11. To use *apt-get*, you need to set a working DNS server:

```
root@raspberrypi:/# echo "nameserver 8.8.8.8" > /etc/resolv.conf
```

    This example uses the Google Public DNS; you may, however, use any DNS server you prefer.

    Check that the Internet works in this new environment:

```
root@raspberrypi:/# ping www.conpaas.eu
PING carambolier.irisa.fr (131.254.150.34) 56(84) bytes of data.
64 bytes from carambolier.irisa.fr (131.254.150.34): icmp_seq=1 ttl=50 time=35.8 ms
[... output omitted ...]
```

12. Use *apt-get* to install any packages that your application requires:

```
root@raspberrypi:/# apt-get update
Hit http://archive.raspbian.org wheezy Release.gpg
Hit http://archive.raspbian.org wheezy Release
[... output omitted ...]

root@raspberrypi:/# apt-get install <...>
```

13. Make the final configurations (if needed) and make sure that everything works.

14. Clean-up:

    Exit the chroot:

```
root@raspberrypi:/# exit
exit
root@raspberrypi:/home/pi#
```

    Unmount `/dev`, `/dev/pts` and `/proc`:

```
root@raspberrypi:/home/pi# umount conpaas-img/proc
root@raspberrypi:/home/pi# umount conpaas-img/dev/pts
root@raspberrypi:/home/pi# umount conpaas-img/dev
```

    Unmount the image:

```
root@raspberrypi:/home/pi# umount conpaas-img
```

    Remove the directory:

```
root@raspberrypi:/home/pi# rm -r conpaas-img
```

    Delete the loop device mapping:

```
root@raspberrypi:/home/pi# losetup -d /dev/loop0
```

    That's it! Now the file `conpaas-rpi.img` contains the new ConPaaS image with your application pre-installed.

You can now register the new image to the cloud of your choice and update the ConPaaS Director's settings to use the new image. Instructions are available in the Installation guide:

For Amazon EC2: *Registering your custom VM image to Amazon EC2*

For OpenStack: *Registering your ConPaaS image to OpenStack*

C

D

E

O