UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Integration of KVM in the Openhuaca Cloud Platform

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Daniel Campos Gómez

In partial fulfilment

of the requirements for the degree in

NETWORK ENGINEERING

Advisor: Jose Luis Muñoz Tapia

Barcelona, June 2018

Abstract

The purpose of this project is the integration of KVM technology into a cloud platform called Openhuaca that lets the user of the software the possibility of creating and managing virtual containers and virtual machines in an easy and fast way. Besides, it can be administrated with domains. It is thought to be implemented in small or medium stages; for example, in a teaching area to help all the students' necessities in a centralized way. Therefore, every student can connect to all virtualized facilities with their specific digital certificate.

The complexity of the project has made it necessary to create a working team formed by a supervisor, ex-students and various TFG students from different universities in order to review and introduce new functionalities to the project efficiently and quicker.

Resum

L'objectiu d'aquest projecte és integrar la tecnología KVM a una plataforma cloud anomenada Openhuaca. Aquesta plataforma permeteix a l'usuari crear i gestionar diversos contenedors i màquines virtuals d'una manera ràpida i senzilla. A més, pot ser administrada per diversos dominis. Aquest projecte està pensat per a petits o mitjans entorns, per exemple, a l'àrea docent per poder controlar els ordinadors dels laboratoris de la Universitat. Openhuaca està pensat per cobrir les necessitats dels alumnes de forma centralitzada. Per tant, cada estudiant podrà conectar-se als seus contenidors i màquines virtuals amb el seu propi certificat digital.

La complexitat del projecte ha fet necessaria la creació d'un equip de treball format per un supervisor, antics estudiants i diversos estudiants de TFG de diferents universitats amb el fi d'implementar noves funcionalitats a l'eina i perfeccionar les existents d'una manera eficient i ràpida.

Resumen

El objetivo de este proyecto es integrar la tecnología KVM en una plataforma cloud denominada Openhuaca. Esta plataforma permite al usuario crear y gestionar diversos contenedores y máquinas virtuales de una forma rápida y sencilla. Además, puede ser administrada por diversos dominios. Está pensado para pequeños o medianos entornos, por ejemplo, en el área docente para poder controlar los ordenadores de los laboratorios de la Universidad. Openhuaca está preparado para cubrir las necesidades de los alumnos de forma centralizada. Por lo tanto, cada estudiante podrá conectarse a sus contenedores y máquinas virtuales con su propio certificado digital.

La complejidad del proyecto ha hecho necesaria la creación de un equipo de trabajo formado por un supervisor, antiguos estudiantes y varios estudiantes de TFG de diferentes universidades con el fin de implementar nuevas funcionalidades en la herramienta y perfeccionar las ya existentes de una forma eficiente y rápida.

Acknowledgements

Openhuaca has been divided in various threads of work, so the supervisor and the teammates have supported the development of the thesis:

Jose Luis Muñoz, who is a teacher from ETSETB doctorate in Networking, as the supervisor, has given the team advice during development. At the same time, he was one of the beta testers and has done a functional review of the documentation presented.

Rafa Genés, who is a co-worker, has been an initial code developer of LxC for Openhuaca. [14]

Jorge Eduardo Buzzio, who is a student from Lima, Perú. As a co-worker, has been code developer. In addition, he has done a functional review of the code implemented and a functional review of the thesis with the supervisor.

Daniel Capdevila, who is a student from ETSETB, as a co-worker, has been a code reviewer and beta tester.

Finally, the j3o networking team collaborated with the initial developments of the LxC extension and, afterwards, with the integration of KVM.

Revision history and approval record

Revision	Date	Purpose
0	13/04/2018	Document creation
1	14/04/2018	Document revision – Added *chapters 1.1 to 1.3.3*
2	17/04/2018	Document revision – Added *chapter 2*
3	20/04/2018	Document revision – Added *chapters 3.1 to 3.2.3.3.*
4	27/04/2018	Document revision – Added *chapters 5 and 6*
5	02/05/2018	Document revision – Added *QCOW2 extra information, DNSMASQ configuration guide and Libvirt installation guide.*
6	03/05/2018	Document revision – Added *KVM QEMU installation*
7	07/05/2018	Document revision – Added *Storage formats for VM disk images*
8	08/05/2018	Document revision – Added *Lbivirt Troubleshooting & Annexes creation*
9	23/05/2018	Document revision – Added *PAN Bibliography & Annexes*
10	01/06/2018	Document revision – Added *Work Packages*
11	06/06/2018	Document revision – Added *SSH comparative in Annexes & Next Steps (etcd)*
12	08/06/18	Document revision – Revised *State of the art & Added Next Steps (snapshots with libvirt)*
13	12/06/18	Document revision – Added *Managing of VM in Openhuaca*
14	27/06/18	Document revision – Added *Management of KVM guest's naming in Openhuaca & Set up serial TTY in KVM guests*
15	28/06/18	General Document Revision

DOCUMENT DISTRIBUTION LIST

Name	**e-mail**
Daniel Campos Gómez	campos.gomez.daniel@gmail.com
Jose Luis Muñoz Tapia	jose.munoz@entel.upc.edu
Jorge Eduardo Buzzio García	jbuzzio410@gmail.com
Rafa Genés Durán	rafa.gd.10@gmail.com
Daniel Capdevila	danicapdevila3@gmail.com

Written by: Campos Gómez, Daniel		Reviewed and approved by: Muñoz Tapia, Jose Luis	
Date	02/07/2018	Date	02/07/2018
Name	Daniel Campos Gómez	Name	Jose Luis Muñoz Tapia
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

This thesis has the purpose of explain detailed how to integrate a new virtualization technology into open source project Openhuaca; specifically, integrating KVM in Openhuaca. It is a cloud platform based on LxC containers till this moment. This containers are a kind of virtual machines isolated from the kernel. It manages the container users, resources, permissions and generates their own domains and certifications to make an access control of each container.

The document includes the objectives, the requirements, the implementation of KVM and integration procedures and the incidences that the project suffered during the realization.

For further information, please visit www.openhuaca.com

1.1. Statement of purpose

The main purpose of this project is learning about the existing methods of organization in order to integrate into another project a new feature properly and understand the necessities and risks that entail aggregate new features to a complex application. How to face the possible issues, either when making a revising of code, communicating with the rest of the team or working in parallel with the others developers.

On the other hand, the goal of Openhuaca is to cover the need to have a simply cloud platform with a virtual machines where a user does not need large configurations or complex environments. Openhuaca has been designed for those small or medium stages where you need some virtualized environments but have few resources, so it has been created as an installable package so that in any Debian-type distribution can be installed and allow the user to create a very customizable cloud platform in an easy way.

At first, the main idea of Openhuaca project was to create an extended version of LxC. But to this day, as a result of the good work made before in this project, rises up the need of extend this project. Integrating KVM into Openhuaca Cloud Platform is the next step to carry out to make Openhuaca grows-up and become an application with more virtualization solutions.

1.2. Requirements and specifications

This project, from its beginnings to the moment, has been designed to be compatible with the main part of equipment so the system requirements are minimal.

- User needs to have installed a debian-type distribution to be able to install the package, for instance Linux Ubuntu distributions or Linux Mint distributions as well.
- User equipment has to include hardware virtualization features that help accelerate virtual machines. Nowadays PCs use to have it, but it could be possible that they don't have it enabled by default. This issue is easy solved just accessing to the BIOS and enabling the VT-x / AMD-V hardware acceleration.
- LxC, libvirt, QEMU and KVM are software required in order to run the platform. This software is available in the official repositories of Ubuntu.

Things to take into account if running an LxC virtualized machine is required:

- Each container is located in the kernel next to the native operating system, so the user has to design the machine take into account the resources that want to assign to the host and each container. It is possible to modify the resources dynamically using the command line.

1.3. Methods and procedures

Different methods have been followed for the software development in order to have minimal runtime and source code as efficient as possible. The documentation of this project has been carefully written and reviewed for the supervisor and other co-workers. The communication with the supervisor and other co-workers has been performed using technologies as jitsi (a video/audio streaming platform online) and arranging meetings.

1.3.1. About the software application

Initially the platform was developed in bash. Nowadays, thanks to work of the team mates, the application code is python, an interpreted high-level programming language very common in development. In fact, we can found it in video games, web frameworks or web services (for instance BitTorrent or Spotify). Besides, python provides many facilities due to it has many libraries to develop with. In fact, in this project, libvirt library is very useful to integrate KVM into Openhuaca.

On the other hand, GIT was used to share the code developed in each co-worker host with another teammates. The repository was created in a virtual machine in the j3o.upc.edu server and secured with credentials. In this way, the project could be modified in any computer remotely. In addition, in this machine was installed another container that let the possibility of installing Openhuaca and test all its functionalities during development.

Finally, the j3o networking team department provided two softwares. “Debrepros” is a script that allows the developer to generate a Debian package from a few directories and the second was an example of a script about how to create certification authorities and ssh certificates.

1.3.2. Documentation

UPC’s moodle has been used to download the documentation required for this project. All documentation has been edited with open source and once delivered and reviewed by the supervisor have been printed in PDF format to be uploaded.

In order to share all the documentation with the teammates and supervisor, it was used a SVN repository in j3o machine. There were uploaded all the documentation, researches and information related to the project to maintain all of the team learned about the progress and technical information about the implementation of the new technology.

All the information to deliver has been reviewed and approved by the supervisor of the project and officially submitted for the evaluation of the work in the moodle.

1.3.3. Communication

In terms of communication, it is necessary to divide the information into two well differentiated techniques: face-to-face and remote communication.

In face-to-face communication, in order to improve the quality of the communications when designing the functional scheme of the application, or when a critical step is required to be done in the integration. So, we have been able to agree a meeting in this situations, but also we have gathered monthly in order to talk about the project and specify the next steps to carry out. Some of these meetings were also scheduled together with the supervisor and other teammates.

On the other hand, remote communications have been focused on emails for solving small issues (or questions) and agreeing meetings. Moreover, we have used VoIP technologies like jitsi or Google Hangouts as a calling tool which allows us to share screen and to keep in touch continual with the rest of co-workers.

Finally, in the j3o machine, a remote desktop was installed, VNC. So that, several people could enter the desktop, allowing the team to have a remote meeting and everyone can view and edit in the same terminal.

1.4. Work Plan

1.4.1. Work Packages

<table>
<tr><td colspan="2">Project: Integration of KVM on the Openhuaca Cloud Platform</td><td>WP ref: 1</td></tr>
<tr><td colspan="2">Major constituent: Hardware and Environment</td><td>Sheet 1 of 4</td></tr>
<tr><td colspan="2" rowspan="2">Short description:
Set up workspace. Install software required, repositories, etc</td><td>Planned start date: 13/02/2018
Planned end date: 15/02/2018</td></tr>
<tr><td>Start event: 13/02/2018
End event: 15/02/2018</td></tr>
<tr><td>Internal task T1:
Install Ubuntu, SVN, VNC, etc
Internal task T2:
Define project objectives</td><td>Deliverables:

N/A</td><td>Dates:

15/02/2018</td></tr>
</table>

Table 1 .- WP1

<table>
<tr><td>Project: Integration of KVM on the Openhuaca Cloud Platform</td><td colspan="2">WP ref: 2</td></tr>
<tr><td>Major constituent: Software</td><td colspan="2">Sheet 2 of 4</td></tr>
<tr><td rowspan="2">Short description:
Development of KVM & QEMU virtual machines tester code to understand how VM Management works and integrate it to Openhuaca.</td><td colspan="2">Planned start date: 26/02/2018
Planned end date: 18/06/2018</td></tr>
<tr><td colspan="2">Start event: 15/02/2018
End event: 30/06/2018</td></tr>
<tr><td>Internal task T1:
Develop a simple-checker script with libvirt.
Internal task T2:
Integrate features tested and checked previously
Internal task T3:
Add manuals and manage domains</td><td>Deliverables:

Openhuaca 2.0</td><td>Dates:

02/07/2018</td></tr>
</table>

Table 2 .- WP 2

<table>
<tr><td colspan="2">Project: Integration of KVM on the Openhuaca Cloud Platform</td><td>WP ref: 3</td></tr>
<tr><td colspan="2">Major constituent: Software</td><td>Sheet 3 of 4</td></tr>
<tr><td colspan="2" rowspan="2">Short description:
Integration of KVM & QEMU virtual machines in Openhuaca</td><td>Planned start date: 15/03/2018
Planned end date: 30/06/2018</td></tr>
<tr><td>Start event: 15/03/2018
End event: 30/06/2018</td></tr>
<tr><td>Internal task T1:
Update Openhuaca
Internal task T2:
Integrate features, test and check if they work as expected
Internal task T3:
Add certificates</td><td>Deliverables:

Openhuaca 2.0</td><td>Dates:

02/07/2018</td></tr>
</table>

Table 3 .- WP 3

<table>
<tr><td colspan="2">Project: Integration of KVM on the Openhuaca Cloud Platform</td><td>WP ref: 4</td></tr>
<tr><td colspan="2">Major constituent: Documentation</td><td>Sheet 4 of 4</td></tr>
<tr><td colspan="2" rowspan="2">Short description:
Write documentation files.</td><td>Planned start date: 16/02/2018
Planned end date: 30/06/2018</td></tr>
<tr><td>Start event: 16/02/2018
End event: 30/06/2018</td></tr>
<tr><td>Internal task T1:
Write manual files for KVM on Openhuaca
Internal task T2:
Proposal, Critial Review
Internal task T3:
Thesis and Presentation</td><td>Deliverables:

Openhuaca 2.0</td><td>Dates:

02/07/2018</td></tr>
</table>

Table 4 .- WP 4

1.4.2. Milestones

WP#	Short Title	Milestone / deriverable	Date (week)
1	Introduction	Prepare Workspace	2
2	Feature: Initial Files	Release 1.0	4
2	Feature: KVM-cmd (Python)	KVM 1.0	7
2	Feature: KVM manuals	KVM 1.1	9
2	Feature: KVM domains	KVM 2.0	13
3	Feature: KVM integration into Openhuaca	Openhuaca 2.0	16
4	Doc. Proejct Proposal	Deliver project proposal	3
4	Doc. Critical Review	Deliver project critical review	12
4	Doc. Thessis	Deliver Thesis	20
4	Presentation	Deliver file.pptx	22

Table 5 .- Milestones

1.4.3. Gantt Diagram

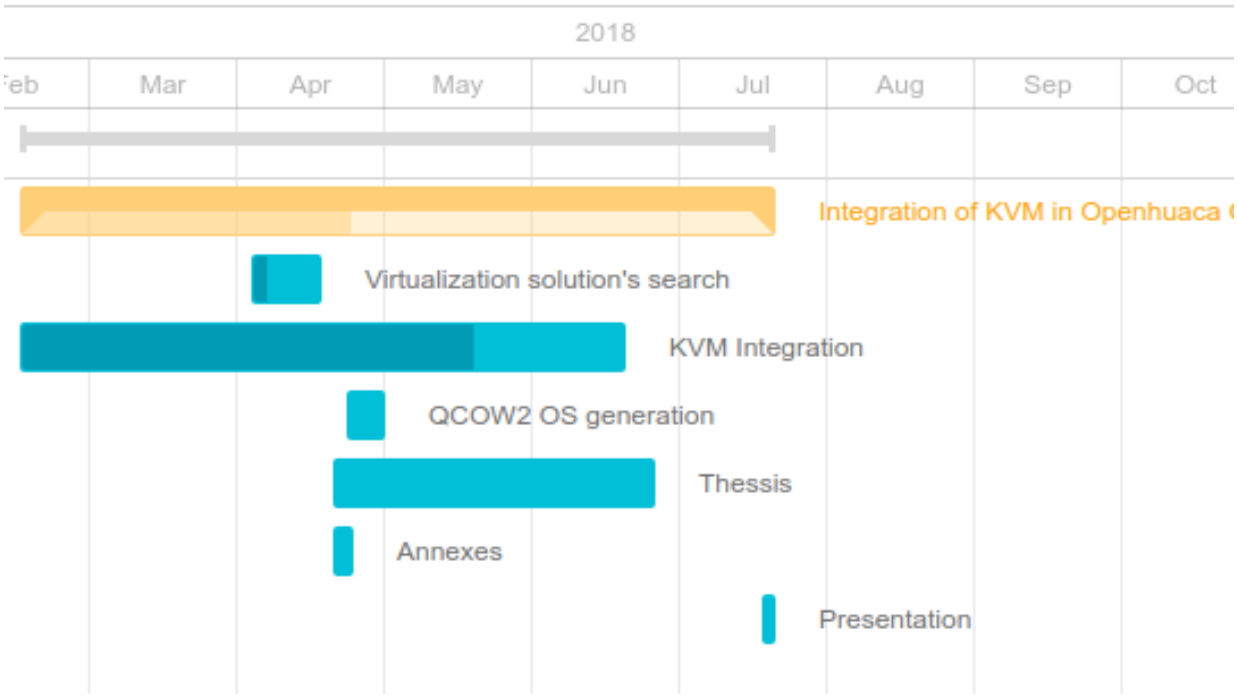

Figure 1.- Gantt Diagram

Task name	Start date	End date	Duration hour	Progress	Estimation (hours)
Total estimate	*2018/02/15*	*2018/07/20*	*892*		*1254*
Integration of KVM in Openhuaca Cloud ...	**2018/02/15**	2018/07/20	892	43%	1254
Virtualization solution's search	2018/04/03	2018/04/17	84	20%	84
KVM Integration	2018/02/15	2018/06/19	708	75%	708
QCOW2 OS generation	2018/04/23	2018/04/30	48	0%	48
Thessis	2018/04/20	2018/06/25	372	0%	372
Annexes	2018/04/20	2018/04/24	18	0%	18
Presentation	2018/07/18	2018/07/20	24	0%	24

Figure 2.- Detailed Gantt Diagram

1.5. Plan changes and incidences

1.5.1. Plan changes

There are not significant modifications in the Work Plan designed at first. Although I can highlight

- Searching information, user guides, manuals of KVM / QEMU / VIRSH / LIBVIRT initially was setted to 10 days but this task has taken to me about 20 days. In fact, is common to find some new features of these technologies in forums and I use to review this weekly.

- Critical Review document initially was setted to 63 days, but it has taken 5 days.

1.5.2. Incidences

Due to this project is based on code development, is very usual to encounter some difficulties. Next, I list the most significant issues that I had to deal with:

- Compatibility between features that the physical machine could offer and the requirements of software. Is common to able advanced virtualization properties from the BIOS of the physical host.

- Comprehension of the whole project in order to start integrating the main utilities to manage KVM

VM together with the previous project which includes LxC ones.

- Understanding what tools I can use and how to use it since this is an innovative project.

This means take a good basis, study very well the tools available and the main objective of the project, as well as study the programming language to use and to be absolutely safe of what are you doing and what is the next step to take.

2. State of the art of the technology used or applied in this thesis:

In order to know the potential of integration of a new technology in the use of this product in the market, it was necessary to make some research about the cloud platforms with the services that offer kernel-level virtualization used nowadays. Therefore, services with VMWare or VirtualBox are discarded because they create virtual machines on the host OS and not directly on the kernel of the physical computer.

So, the following sections are about reviews of the state of art of virtualization software.

2.1. LxC

LxC is a non-graphical and lightweight method to run multiple virtual units simultaneously on a single control host. Containers are isolated with Kernel Control Groups (as known as cgroups) and Kernel Namespaces. This is, creating diverse types of Linux distributions to assigning limited resources to each one. But unlike Openhuaca, it does not have user management, nor allow a user to manage the domains or any kind of modification of the network. It only generates a bridge on the host and binds all containers with VETH level 2 links. So, all the containers are generated in the same network and are connected each other.

Also it should be noted that Openhuaca prepares the containers to accept SSH connections and orchestrates the certification authorities and their certificates.

LxC provides an operating system-level virtualization where the Kernel controls the isolated containers with the limitation of there is not a method that has direct hardware access, so, you cannot run near-real-time-requirements for applications. Another restraint is that with this technique you cannot virtualize any OS since it is based on OS-level virtualization technology that allows creation and running of multiple isolated Linux environments (also called VE) [10]

2.2. Docker

Docker is a software that performs operating-system-level virtualization. Docker is primarily developed for Linux, where it uses the resource isolation features of the Linux kernel such as kernel namespaces to allow independent containers to run within a single Linux instance.

Docker implements a high-level API to provide lightweight containers that run processes in isolation. A Docker container, unlike a virtual machine, does not require a separated OS. Instead, it relies on the kernel's functionality and uses resource isolation for CPU and memory, and separate namespaces to isolate the application's view of the OS.

Docker is designed to run from a host and create different services, so it performs container virtualization, but is not intended to manage an entire computer environment. However, it is a really useful tool; it is not focused on the same community as Openhuaca, letting the project to have a market that does not match with it.

As summary, Docker is very complete and improved tool from LxC, but has focused on virtualizing processes and not the entire operating system creating differentiation from Openhuaca. [11]

2.3. PROXMOX

Another remarkable to talk about is Proxmox, it is an open source container virtualization program, both the LxC type and KVM. It also has a great control of domains and has a very comfortable web interface where it is possible to monitor and manage the containers. This program is the most similar to Openhuaca but has two major differences that can be considered large enough to think that Openhuaca has a different audience from Proxmox.

Proxmox is very similar to Openhuaca, but there are two key points that differentiate it. Proxmox is distributed in ISO files, so user have to install on their own operating system while Openhuaca distributes debian packages, that allows to be installed in any Linux distribution. The other difference is that Proxmox is designed for big stages so user needs an initial investment in hardware much greater than using Openhuaca.[21]

2.4. KVM

KVM is a virtualization solution based on the Kernel. It allows to implement a complete virtualization with Linux. Since Linux version 2.6.20, this module is included in the Linux kernel. KVM uses raw OS disk images. In this way, every machine has its self-virtualized hardware, its NIC, its hard disks, its graphical card, etc. Another remarkable feature of KVM is that, by means of raw image we can run any virtual Windows or Linux OS.

Implementation of KVM in Openhuaca will offer to users a very simple way to manage, create and delete VM. Below will be detailed the features of KVM. [12]

2.5. QEMU

QEMU is a processor emulator based on dynamic binary translation. This means, it translates binary code of the source architecture into code adapted for the guest architecture. It can virtualize both Windows and Linux OS and it can run in ordinary x86 architecture. The main purpose of this technology is to emulate an OS into another without distribute its hard disk.

QEMU implements the Copy-On-Write disk format. It can be declared a virtual unit of certain space of disk (for instance 30GBytes) but the disk image will use only the required space. Another interesting feature is that it is a support to run Linux binaries in other architectures. Besides, with QEMU, it is possible to maintain the state of the guest system image and write changes in other separated image. It is a simple way to recover the states of the VM in case of failure.

In Openhuaca, we use the QEMU driver (through the corresponding API detailed next) as provided of a single system wide privileged driver to manage the instances. [13]

3. Methodology / project development:

In this chapter is described the methodology followed in all relevant methods that has been utilized such as communication methods or the procedure of creating the work documentation. Also, it includes a detailed description of Openhuaca as well as an exhaustive information of the software integrated.

3.1. Communication

As it was mentioned in the introduction, the communications can be divided between in-person and remote meetings.

In face-to-face communication, in order to improve the quality of the communications when making critical decisions such as how to manage the new VM created with the technologies integrated into the platform. It is a good technique in the beginning because simplifies a lot the adaptation of a new worker in the project. In particular, it went well to my adaptation in the project.

Once the comprehension of the project is reached, the communication became remote. In this way, it is easy to the whole group of project to agree meetings and the meetings could be more frequently and without time restrictions. This is a very useful method when developing and testing code. This communication, also, is supported graphically thanks to a virtualized machine enabled in the server j3o. With it, the group could make audio meeting by VoIP applications and share the desktop of the virtual machine, so spaced out from speaking this tool let the team to work on the same screen.

The supervisor decided to use GIT as a method of sharing, downloading and updating (in a fetched way) the software. So, each developer member of the team must use this application to make changes in Openhuaca code, downloading the resources to each physical host, developing code, and then; committing the new code to the machine in the j3o server. Once the software is upgraded, it is necessary to use the “debrepos-j3o” software to create an installable Debian package. Then, the new code can be tested.

3.2. Openhuaca

3.2.1. What is Openhuaca?

Openhuaca is a Cloud Platform that nowadays merges two virtualization techniques such as LxC and KVM in order to offer solutions that can be configured by users in an easy and fast way. For instance, Openhuaca can be used to deploy a network in an educational location in a centralized way. This application is designed for little or medium environments. So, Openhuaca can allow the connection to the network to several users in a centralized way but maintaining the privacy. This is, every user has its own container and its own certificates.

Further information about Openhuaca’s infrastructure is detailed in the next chapters.

3.2.2. Openhuaca's networking

Openhuaca's networking is divided in domains. Every domain is composed for a virtual bridge to which LxC or KVM instances are linked.

In Openhuaca domains must be created. This can be done by typing:

```
openhuaca domain --create domain
```

The output of the command shows the network assigned, for example, in this case 172.20.1.0/24. It is possible to consult the status of the domain, if required by typing

```
openhuaca domain --status
```

Once it is created, its initial state is STOPPED, so we need to run it. In order to assign automatically an IP address to the virtual instances of a certain domain, every virtual bridge domain has set up a dnsmasq. So, to run a domain, type:

```
openhuaca domain --start
```

dnsmasq provides network infrastructure for small networks; DNS, DHCP, router advertisement and network boot. It is a lightweight software and have a small footprint for routers and firewalls (in the case that there are). Then, dnsmasq is in charge of assigning IP addresses to hosts of a certain domain but it also is capable of act as a DNS server by reading its local DNS /etc/hosts if required. So, machines which are configured by DHCP have their names automatically included in the DNS and the names can specified by each machine or centrally by associating a name with a MAC address or UUID in the dnsmasq configuration file.

In Openhuaca, attach a domain means replace the terminal using actually by the tty of a certain guest. Then, once device is created through a template in a certain domain and attached to a bridge, it is possible to swap the console by the tty of the guest. In order to attach a new instance in a specific domain type:

```
openhuaca attach –d alpha –n instance
```

It is possible to list the instances of a certain domain just typing:

```
openhuaca ls –f –d domain
```

Once the domain is functional, user can generate instances LxC or KVM.

In the below image, an example of Openhuaca's network distribution is represented.

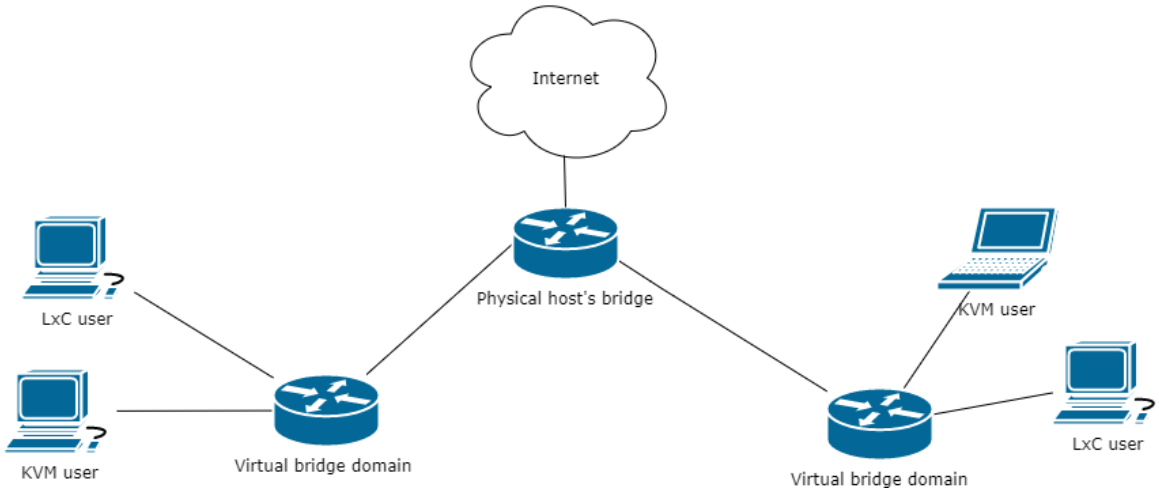

Figure 3.- Openhuaca's networking

3.2.2.1. DNSMASQ configuration guide example

As mentioned before, dnsmasq provides a local DNS server, a DHCP server with support for DHCPv6 and PXE, and a TFTP server. It is designed to be lightweight and have a small footprint, suitable for resource constrained routers and firewalls. dnsmasq can also be configured to cache DNS queries for improved DNS lookup speeds to previously visited sites.

To configure dnsmasq, you need to edit /etc/dnsmasq.conf. The file contains extensive comments explaining its options. For all available options see dnsmasq(8).

dnsmasq by default enables its DNS server. If you do not require it, you need to explicitly disable it by setting DNS port to 0:

```
/etc/dnsmasq.conf
port=0
```

To check configuration file(s) syntax, execute:

```
$ dnsmasq --test
```

Start the daemon

Start/enable `dnsmasq.service`.

To see if dnsmasq started properly, check the system's journal:

```
$ journalctl -u dnsmasq.service
```

The network will also need to be restarted so the DHCP client can create a new `/etc/resolv.conf`.

DNS cache setup

To set up dnsmasq as a DNS caching daemon on a single computer edit /etc/dnsmasq.conf and uncomment the listen-address directive, adding in the localhost IP address:

```
listen-address=127.0.0.1
```

To use this computer to listen on its LAN IP address for other computers on the network:

```
listen-address=192.168.1.1    # Example IP of a certain domain
```

It is recommended that you use a static LAN IP in this case.

Multiple IP address settings:

```
listen-address=127.0.0.1,192.168.1.1
```

DNS addresses file

Note that most of this can also be done natively in `/etc/resolvconf.conf` using the name_servers and name_servers_append options.

After configuring dnsmasq, the DHCP client will need to prepend the localhost address to the known DNS addresses in `/etc/resolv.conf`. This causes all queries to be sent to dnsmasq before trying to resolve them with an external DNS. After the DHCP client is configured, the network will need to be restarted for changes to take effect.

One option is a pure resolv.conf configuration. To do this, just make the first nameserver in /etc/resolv.conf point to localhost:

```
/etc/resolv.conf
nameserver 127.0.0.1
# External nameservers
...
```

Now DNS queries will be resolved first with dnsmasq, only checking external servers if dnsmasq cannot resolve the query. dhcpcd, unfortunately, tends to overwrite `/etc/resolv.conf` by default, so if you use DHCP it is a good idea to protect `/etc/resolv.conf`. To do this, append `nohook resolv.conf` to the `dhcpcd config` file:

```
/etc/dhcpcd.conf
...
nohook resolv.conf
```

It is also possible to write protect your resolv.conf:

```
# chattr +i /etc/resolv.conf
```

More than three nameservers

A limitation in the way Linux handles DNS queries is that there can only be a maximum of three nameservers used in `resolv.conf`. As a workaround, you can make localhost the only nameserver in `resolv.conf`, and then create a separate resolv-file for your external nameservers. First, create a new resolv file for dnsmasq:

```
/etc/resolv.dnsmasq.conf
# Google's nameservers, for example
nameserver 8.8.8.8
nameserver 8.8.4.4
```

And then edit /etc/dnsmasq.conf to use your new resolv file:

```
/etc/dnsmasq.conf
...
resolv-file=/etc/resolv.dnsmasq.conf
...
```

dhcpcd

dhcpcd has the ability to prepend or append nameservers to /etc/resolv.conf by creating (or editing) the /etc/resolv.conf.head and /etc/resolv.conf.tail files respectively:

```
echo "nameserver 127.0.0.1" > /etc/resolv.conf.head
dhclient
```

For dhclient, uncomment in /etc/dhclient.conf:

```
prepend domain-name-servers 127.0.0.1;
```

NetworkManager

NetworkManager has a plugin to enable DNS using dnsmasq. The advantages of this setup is that DNS lookups will be cached, shortening resolve times, and DNS lookups of VPN hosts will be routed to the relevant VPN's DNS servers (especially useful if you are connected to more than one VPN).

Make sure dnsmasq has been installed, but has been disabled. Then, edit `/etc/NetworkManager/NetworkManager.conf` and change the dns in the `[main]` section:

```
/etc/NetworkManager/NetworkManager.conf
[main]
...
```

```
dns=dnsmasq
```

Now restart `NetworkManager` or reboot. `NetworkManager` will automatically start dnsmasq and add `127.0.0.1 to /etc/resolv.conf`. The actual DNS servers can be found in `/run/NetworkManager/resolv.conf`. You can verify dnsmasq is being used by doing the same DNS lookup twice with $ drill example.com and verifying the server and query times.

Custom configuration

Custom configurations can be created for dnsmasq by creating configuration files in `/etc/NetworkManager/dnsmasq.d/`. For example, to change the size of the DNS cache (which is stored in RAM):

```
/etc/NetworkManager/dnsmasq.d/cache.conf
cache-size=1000
```

Adding a custom domain

It is possible to add a custom domain to hosts in your (local) network:

```
local=/home.lan/
domain=home.lan
```

In this example it is possible to ping a host/device (e.g. defined in your `/etc/hosts` file) as `hostname.home.lan`.

Uncomment expand-hosts to add the custom domain to hosts entries:

```
expand-hosts
```

Without this setting, you will have to add the domain to entries of /etc/hosts.

DHCP server

By default dnsmasq has the DHCP functionality turned off, if you want to use it you must turn it on in `(/etc/dnsmasq.conf)`. Here are the important settings:

```
# Only listen to routers' LAN NIC.  Doing so opens up tcp/udp
port 53 to
# localhost and udp port 67 to world:
interface=<LAN-NIC>


# dnsmasq will open tcp/udp port 53 and udp port 67 to world
to help with
# dynamic interfaces (assigning dynamic ips). Dnsmasq will
discard world
# requests to them, but the paranoid might like to close them
and let the
# kernel handle them:
bind-interfaces
```

```
# Optionally set a domain name
domain=example.com

# Set default gateway
dhcp-option=3,192.168.1.1

# Set DNS servers to announce
dhcp-option=6,8.8.8.8,8.8.4.4

# Dynamic range of IPs to make available to LAN PC and the
lease time.
# Ideally set the lease time to 5m only at first to test
everything works okay before you set long-lasting records.
dhcp-range=192.168.111.50,192.168.111.100,12h

# If you'd like to have dnsmasq assign static IPs to some
clients, bind the LAN computers
# NIC MAC addresses:
dhcp-host=aa:bb:cc:dd:ee:ff,192.168.111.50
dhcp-host=aa:bb:cc:ff:dd:ee,192.168.111.51
```

PXE server

PXE requires DHCP and TFTP servers, both functions can be provided by dnsmasq.

dnsmasq can add PXE booting options to a network with an already running DHCP server:

```
/etc/dnsmasq.conf
interface=enp0s0
bind-dynamic
dhcp-range=192.168.0.1,proxy
set up #TFTP server and #DHCP server
```

In case pxe-service does not work (especially for UEFI-based clients), combination of dhcp-match and dhcp-boot can be used. See RFC4578 for more client-arch numbers for use with dhcp boot protocol. [1]

3.2.3. Integration of KVM in Openhuaca

3.2.3.1. KVM vs QEMU

QEMU resides in the user space and provides system emulation including the processor and various peripherals such as disk, VGA, PCIs, serial/parallel ports, etc. Mainly it works by special recompiler that transforms binary code written for a given processor into another one. This is to say, to run MIPS code on a PPC MAC, or ARM in an x86 PC). Typically, QEMU is deployed along with KVM as an in-kernel accelerator where KVM executes most of the guest code natively, while QEMU emulates the rest of the peripherals needed by the guest. If VM needs to talk to external devices, QEMU uses pass-through.

In the specific case where both source and target are the same architecture (for instance the common case of x86 on x86), it is used KQEMU. It still has to parse the code to remove instruction that requires privileged permissions and replace them with context switches. To make it as efficient as possible on x86 Linux distributions, there is this kernel module to handle it.

Being a kernel module, KQEMU is able to execute most code unchanged, replacing only the lowest-level ring0-only instructions. In this case, userspace (ring3) QUEMU still allocates all the RAM for the emulated machine, and loads the code. The difference is that instead of recompiling the code, it calls KQEMU to execute it. All the peripheral hardware emulation is done in QEMU. This is faster that QEMU since most code is unchanged, but still has to transform ring0 code (most of the code in the VM's kernel).[13]

KVM is a Linux Kernel Module included in the mainline since 2.6.20 version. It is an open source software that switches the processor into a new guest state. The guest state has its own set of ring states, but privileged ring0 instructions fall back to the hypervisor code. Since it is a new processor mode of execution, the code doesn't have to be modified in any way. If KVM runs together with QEMU, it acts as fork of QEMU executable. Both work actively to keep differences at a minimum, and there are advantages doing it. The goal is that QEMU should work anywhere and, if a KVM kernel module is available, it could be automatically used. The QEMU software focuses on hardware emulation and portability, while KVM folks focus on the kernel module, sometimes moving small pieces of the emulation there to improve the performance, and interfacing with the rest of the ring3 code.

So, the KVM-QEMU executable works as normal QEMU, allocating RAM, loading the code and, instead of recompiling it, it spawns a thread. This thread calls the KVM kernel module to switch to guest mode and proceeds to execute the VM code. On a privileged instruction, it switches back to the KVM kernel module which, if necessary, signals the QEMU thread to handle most of the hardware emulation. For instance, if you require to work with a VM with 2 or 3 cores, kvm-qemu creates 2 or 3 threads, each of them calls the KVM kernel module to start executing. The concurrency is managed by the normal Linux scheduler, keeping code small. [12]

To sum up, QEMU and KVM both are able to act as a hypervisor. The reason because they work together is that QEMU is slower since it comes to system which don't have hardware virtualization. KVM helps QEMU to access to hardware virtualization features on different architectures. It also adds the acceleration feature to the QEMU process. So, when they are together, QEMU is the emulator (hypervisor) and KVM is the

accelerating agent. In figure 4 is represented the software model distribution of this two features.

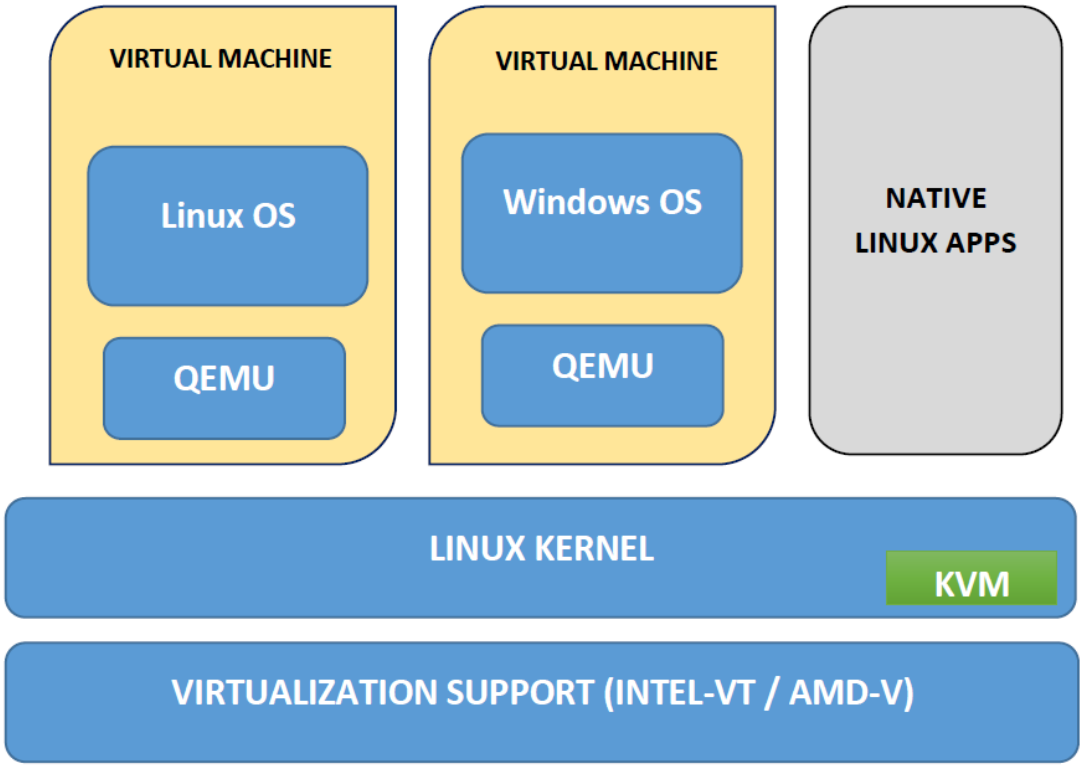

Figure 4.- Software model distribution

In this project, as will be explained below, it is used libvirt. Libvirt is a virtualization management library. It manages both KVM and QEMU. It consists of three utilities namely: an API library, a daemon (livbirtd) and a command line tool (virsh). But this feature will be more detailed in the next section.

3.2.3.2. Storage formats for virtual machines disk images

RAW

The RAW storage format has a performance advantage over QCOW2 in that no formatting is applied to virtual machine disk images stored in the RAW format. Virtual machine data operations on disk images stored in RAW format require no additional work from hosts. When a virtual machine writes data to a given offset in its virtual disk, the I/O is written to the same offset on the backing file or logical volume.

Raw format requires that the entire space of the defined image be preallocated unless using externally managed thin provisioned LUNs from a storage array.

QCOW2

QCOW2 is a storage format for virtual machine disk images. QCOW stands for QEMU copy on write. The QCOW2 format decouples the physical storage layer from the virtual layer by adding a mapping between logical and physical blocks. Each logical block is mapped to its physical offset, which enables storage over-commitment and virtual

machine snapshots, where each QCOW volume only represents changes made to an underlying disk image.

The initial mapping points all logical blocks to the offsets in the backing file or volume. When a virtual machine writes data to a QCOW2 volume after a snapshot, the relevant block is read from the backing volume, modified with the new information and written into a new snapshot QCOW2 volume. Then the map is updated to point to the new place.

3.2.3.3. More about QCOW2

As said before, qcow is a file format for disk image files used by QEMU, a hosted virtual machine monitor. It stands for "QEMU Copy-On Write" and uses a disk storage optimization strategy that delays allocation of storage until it is actually needed. Files in qcow format can contain a variety of disk images which are generally associated with specific guest operating systems. Two versions of the format exist: *qcow*, and *qcow2*, which use the *.qcow* and *.qcow2* file extensions, respectively.

One of the main characteristics of QCOW disk images is that files with this format can grow as data is added. This allows for smaller file sizes than raw disk images, which allocate the whole image space to a file, even if parts of it are empty. This is particularly useful for file systems that do not support sparse files, such as FAT32.

The QCOW format also allows storing changes made to a read-only base image on a separate QCOW file by using copy on write. This new QCOW file contains the path to the base image to be able to refer back to it when required. When a particular piece of data has to be read from this new image, the content is retrieved from it if it is new and was stored there; if it is not, the data is fetched from the base image.

Optional features include zlib-based transparent decompression.

One disadvantage of qcow images is that they cannot be mounted directly as raw disk images would. A utility that is able to read qcow files is required before being able to mount one.

QCOW2 is an updated version of the qcow format, intended to supersede it. The difference to the original version is that qcow2 supports multiple snapshots through a newer, flexible model for storing snapshots. So, due to this new features, QCOW version 2 is the required version to create our VMs. [3]

3.2.3.4. Libvirt

3.2.3.4.1. What is libvirt?

Libvirt is an open source hypervisor-independent virtualization API and toolkit that is able to interact with the virtualization capabilities in several OS. Libvirt provides a generic layer to manage domains on a node. As nodes may be remotely located, libvirt provides methods required to manage the domains within the limits of hypervisor support for the operations. So, it provides methods to enumerate, monitor and use the resources available on the managed mode, such as CPUs, memory storage, networking, etc. Due to management node can be located on a separate physical machine to the management program, this should only be done using secure protocols. But first of explain all the features that makes libvirt a very useful tool for this project, it kis necessary to define some critical vocabulary terms:

- Domain: An instance of an operating system running on a virtualized machined provided by the hypervisor
- Hypervisor: A layer of software allowing the virtualization of a node in a set of virtual machines which may have different configurations to the node itself.
- Node: The physical server. Nodes may be one of many different types such as database nodes or cluster nodes as well.

The libvirt python module is intended to extend to all functions necessary for management of virtual machines. This includes both the core hypervisor and host resources required by VMs.

3.2.3.4.2. Libvirtd

Libvirtd is the server side daemon component of the libvirt virtualization system management, including activities such as starting, stopping and migrating guests between host servers, configuring networking, etc. This daemon is running on the host server. So, the libvirt client libraries connect to this daemon to issue tasks and collect information about the configuration and resources of the host system and guests.

By default, the libvirtd daemon listens for requests on a local UNIX domain socket. It is possible to set up the configuration file of libvirtd to additionally listening on TCP/IP socket.

Restarting libvirtd does not impact running guests and they will be picked up automatically on the condition that their XML configuration files have been defined. If not, these guest will be lost from the configuration.

To close this chapter, there are some remarkable files configuration of libvirt interesting to mention: [22]

- `/etc/libvirtd.conf`: The default configuration file used by libvirtd, unless overridden on the command line using the –f | --config option.
- `/var/run/libvirt/libvirt-sock(-ro)`: The sockets libvirt will use.
- `/etc/pki/CA/cacert.pem`: The TLS Certificate Authority certificate libvirtd will use.
- `/etc/pki/libvirt/servercert.pem`: The TLS Server certificate libvirtd will use.
- `/etc/pki/libvirt/serverkey.pem`: The TLS Server private key libvirtd will use.

3.2.3.4.3. Installation Guide

As mentioned before, libvirt library is used to interface with different virtualization technologies.

Before getting started with libvirt it is best to make sure your hardware supports the necessary virtualization extensions for KVM. Enter the following from a terminal prompt:

`kvm-ok` or `egrep -c '(vmx|svm)' /proc/cpuinfo`

A message will be printed informing you if your CPU does or does not support hardware virtualization. On many computers with processors supporting hardware assisted virtualization, it is necessary to activate an option in the BIOS to enable it.

Virtual Networking

There are a few different ways to allow a virtual machine access to the external network. The default virtual network configuration includes bridging and iptables rules implementing usermode networking, which uses the SLIRP protocol. Traffic is NATed through the host interface to the outside network.

To enable external hosts to directly access services on virtual machines a different type of bridge than the default needs to be configured. This allows the virtual interfaces to connect to the outside network through the physical interface, making them appear as normal hosts to the rest of the network.

Installation

To install the necessary packages, from a terminal prompt enter:

```
sudo apt install qemu-kvm libvirt-bin
```

After installing libvirt-bin, the user used to manage virtual machines will need to be added to the libvirtd group. Doing so will grant the user access to the advanced networking options. In a terminal enter:

```
sudo adduser $USER libvirtd
```

If the user chosen is the current user, you will need to log out and back in for the new group membership to take effect.

In more recent releases the group was renamed to libvirt. Upgraded systems get a new libvirt group with the same gid as the libvirtd group to match that.

You are now ready to install a Guest operating system. Installing a virtual machine follows the same process as installing the operating system directly on the hardware. You either need a way to automate the installation, or a keyboard and monitor will need to be attached to the physical machine.

In the case of virtual machines a Graphical User Interface (GUI) is analogous to using a physical keyboard and mouse. Instead of installing a GUI the virt-viewer application can be used to connect to a virtual machine's console using VNC.

The virt-viewer application allows you to connect to a virtual machine's console. virt-viewer does require a Graphical User Interface (GUI) to interface with the virtual machine.

To install virt-viewer from a terminal enter:

```
sudo apt install virt-viewer
```

Once a virtual machine is installed and running you can connect to the virtual machine's console by using:

```
virt-viewer guest
```

Similar to virt-manager, virt-viewer can connect to a remote host using SSH with key authentication, as well:

```
virt-viewer -c qemu+ssh://virtnode1.mydomain.com/system guest
```

Be sure to replace web_devel with the appropriate virtual machine name.

If configured to use a bridged network interface you can also setup SSH access to the virtual machine.

There are several ways to automate the Ubuntu installation process, for example using preseeds, kickstart, etc. Refer to the Ubuntu Installation Guide for details.

Yet another way to install an Ubuntu virtual machine is to use uvtool. This application, available as of 14.04, allows you to set up specific VM options, execute custom post-install scripts, etc. [9]

There are many ways to install a KVM domain. You can use the virt-installer tool or qemu installer tool as well. In Openhuaca project is used de qemu installer.

Openhuaca automatic base generator plugin

Openhuaca has a method which automates completely the process of installation an OS's iso in a QCOW2 file. It is decided to use QCOW2 format due to can grow larger than the actual data stored within, this happens because the guest OS normally only marks a deleted File as zero, it doesn't gets actually deleted, so underlying QCOW2 file cannot differenciate between allocated and used and allocated but not used storage.

So, the method implemented in Openhuaca it is just required the path to the iso to install and the maximum data storage that the QCOW2 file can reach. So, in order to automate the process of create a new base of a certain OS the method follows the steps explained below.

The purpose of this manual is to explain the steps that Openhuaca carries out and are totally invisible to the user to do this hard process easier. So, it is explained how to create an image that will be the base-hardisk of the VM that we will managed with libvirt methods in Openhuaca later.

So, the steps to carry out are:

1. Create a .qcow2 file

```
qemu-img create -f qcow2 path/where/save/qcow2/base.qcow2
```

2. Start a QEMU guest with this image and an installable OS (iso)

```
Qemu-system-x86_64 -name basename -enable-kvm -M pc-0.12
-m 768 -smp 2 -boot d -drive
file=/path/where/save/qcow2/base.qcow2,if=virtio,index=0,
```

```
media=disk,format=qcow2                                    -drive
file=/path/to/iso.iso,index=0,media=cdrom                   -net
nic,model=virtio,macaddr=AUTOMATIC:MAC:GENERATOR -vga std
-display vnc 0 -balloon virtio -k es -usbdevice mouse
```

3. Restart QEMU guest but without the iso file and setting up the type of image as QCOW2.

```
qemu-img create -f qcow2 -b basename.qcow2 replica1.qcow2
```

4. Replicate the .qcow2 file where the OS has been installed every time that is required to create a new guest with this OS.

```
qemu-system-x86_64 -name basename -enable-kvm -M pc-0.12 -
m 768                          -smp 2 -boot d -drive
file=/path/to/replica1.qcow2,if=virtio,index=0,media=disk
,format=qcow2        -net        nic,model=virtio,macaddr=
AUTOMATIC:MAC:GENERATOR -vga std -display vnc 0 -balloon
virtio -k es -usbdevice mouse
```

Things to take into account:

- The base format has to be QCOW2.
- Options of qemu-system-x86_64 [4]

-smp
[cpus=]*n*[**,cores=***cores*][**,threads=** *threads*][**,sockets=***sockets*][**,maxcpus=***maxcpus*]
Simulate an SMP system with *n* CPUs. On the PC target, up to 255 CPUs are supported. On Sparc32 target, Linux limits the number of usable CPUs to 4. For the PC target, the number of *cores* per socket, the number of *threads*per cores and the total number of *sockets* can be specified. Missing values will be computed. If any on the three values is given, the total number of CPUs *n* can be omitted. *maxcpus* specifies the maximum number of hotpluggable CPUs. SMP is supported with up to 255 CPUs.

-net nic[,vlan=*n*][**,macaddr=***mac*][**,model=***type*]
[**,name=** *name*][**,addr=***addr*][**,vectors=***v*]
Create a new Network Interface Card and connect it to VLAN *n* (*n* = 0 is the default). The NIC is an e1000 by default on the PC target. Optionally, the MAC address can be changed to *mac*, the device address set to *addr* (PCI cards only), and a *name* can

```
       be assigned for use in monitor commands. Optionally,
       for PCI cards, you can specify the number v of MSI-X
       vectors that the card should have; this option
       currently only affects virtio cards; set v = 0 to
       disable MSI-X. If no -net option is specified, a
       single NIC is created. QEMU can emulate several
       different models of network card. Valid values
       for type are "virtio", "i82551", "i82557b",
       "i82559er", "ne2k_pci", "ne2k_isa", "pcnet",
       "rtl8139", "e1000", "smc91c111", "lance" and
       "mcf_fec". Not all devices are supported on all
       targets. Use "-net nic,model=help" for a list of
       available devices for your target.


-vga type
       Select type of VGA card to emulate. Valid values
       for type are
cirrus
            Cirrus Logic GD5446 Video card. All Windows
            versions starting from Windows 95 should
            recognize and use this graphic card. For optimal
            performances, use 16 bit color depth in the guest
            and the host OS. (This one is the default)
std
            Standard VGA card with Bochs VBE extensions. If
            your guest OS supports the VESA 2.0 VBE
            extensions (e.g. Windows XP) and if you want to
            use high resolution modes (>= 1280x1024x16) then
            you should use this option.
```

- VNC support should be available by default. If not, there may be a problem in firewall or qemu/libvirt configuration.

Regarding qemu by default it only accepts local vnc connections. To change that edit `/etc/libvirt/qemu.conf`:

```
vnc_listen = ”0.0.0.0”
```

This will enable connections from all hosts, which depending what you are trying to do can be very dangerous. In that file you can find more info on how to secure things up.

Also, to enable remote access is required, edit `/etc/libvirt/libvirtd.conf`:

```
listen_tcp = 1
```

The same security concerns regarding QEMU apply here.

3.2.3.4.4. About guest domains

In terms of libvirt, a domain is an instance of an operating system running on a virtualized machine. The connection object with the hypervisor provides methods to manage new guests taking into account that every guest must have some unique identifiers. These identifiers are:

- ID: is an unsigned integer, unique amongst running guest domains on a single host so, an inactive domain does not have ID.
- Name: is a short string, unique amongst all guest domains (running or just defined) on a single host. The hypervisor will store the domain configuration as XML files on disk, according to the domain name.
- UUID (Universally Unique IDentifier): is a 128 bit code to identify an instance uniquely in a system with different domains (understood like several IP regions). Further information about UUID can be founded in RFC 4122.

There are two guest types: Transcient and persistent.

A Transcient guest domain can be managed only when it is active on the host and, if it is powered off, all traces of it disappear. However, a persistent domain has its configuration maintained in a data store in an implementation defined format on the host by the hypervisor so, if a persistent domain is powered off, it is still possible to manage its inactive configuration. [5]

In libvirt, domains are defined with XML files which works as a config init files for the guests. So, the XML file is a critical feature when defining domains. It is possible to ensure that the definition is correct. Libvirt provides a method to be aware of it, `virt-xml-validate`. Into these files there are definitions of the features of the guests as identifiers or RAM memory, number of vCPUs, etc. In XML guest definition must have the boot image of the OS. In fact, this definition is a critical point in the XML due to the Openhuaca boot systems image policy, the libvirt domains will be booted from QCOW2 files. Openhuaca has a repository of images from different operating systems such as different distributions of Linux and Windows. These are images that have installed a simple version of a certain operating system, in such a way that the user who owns the virtual machine can install what he needs and, in addition, have base images as light as possible. Then, each time a new instance is defined with libvirt, the image of the corresponding operating system will be replicated in a file of QCOW2 format. It is a Copy-On-Write image format, highly supported by the QEMU emulator. It is a representation of a device with the size of a fixed block in a file. Therefore, it is a smaller file. Because it is Copy-On-Write, it is an image that represents the changes made to a disk image. It is also compatible with the concept of snapshots.

Once XML file is generated, it is necessary to define it as instance. Libvirt API provides the method `defineXML` to do so. Notice that it is necessary to establish a connection with the hypervisor (QEMU in this case) to carry out operations that requires hypervisor actions such as list domains. At this point, it is created a VM and its state is “shut off”.

Talking about the states more detailed, the lifecycle of a KVM guest is made up 5 states, so a domain guest can be in

1. Undefined: This is the baseline state. Libvirt doesn't know anything about domains in this state because the domain hasn't been defined or created yet.
2. Defined or Stopped: The domain has been defined, but it's not running. Only persistent domains can be in this state. When the transient domain is stopped or shutdown, it creates to exist.
3. Running: The domains has been created and started either transient or persistent domain. Either domain in this state is being actively executed on hypervisor
4. Paused: The domain execution on hypervisor has been suspended. Its state has been temporally stored until is resumed. The domain does not have any knowledge whether it was paused or not.
5. Saved: The domain state is stored to persistent storage. The domain can be restored and it does not notice that any time has passed.

Graphically it can be shown like the following image.

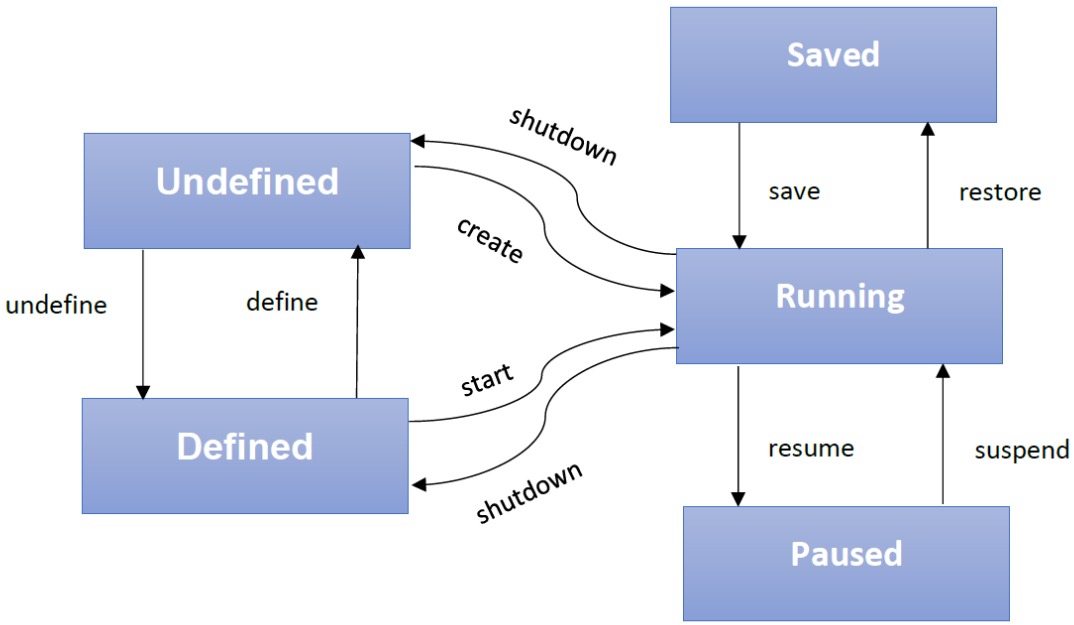

Figure 5.- VM guest state flow

To the end user, the process looks very simple. To create an instance, the user just have to type

```
openhuaca create -d domain -n name -b base.
```

In this way, if the base typed in LxC-type, openhuaca will use the proper LxC methods. If not, the software will apply the KVM ones. Then, by typing

```
openhuaca start -d domain -n name
```

3.2.3.5. Managing of VM in Openhuaca

The goal of this chapter is to explain the guest management focusing this explanation in KVM ones but taking into account that KVM lives together LxC.

First of all, as mentioned before, KVM (an LxC, too) works from a minimum installed image file written into a qcow2 file. This minimum installed image is known as base in Openhuaca. In rough outlines, taking into account that Openhuaca is finally distribuited as a debian package, it is distribuited into some basic directories.So the distribution of Openhuaca files is distribuited as represented below.

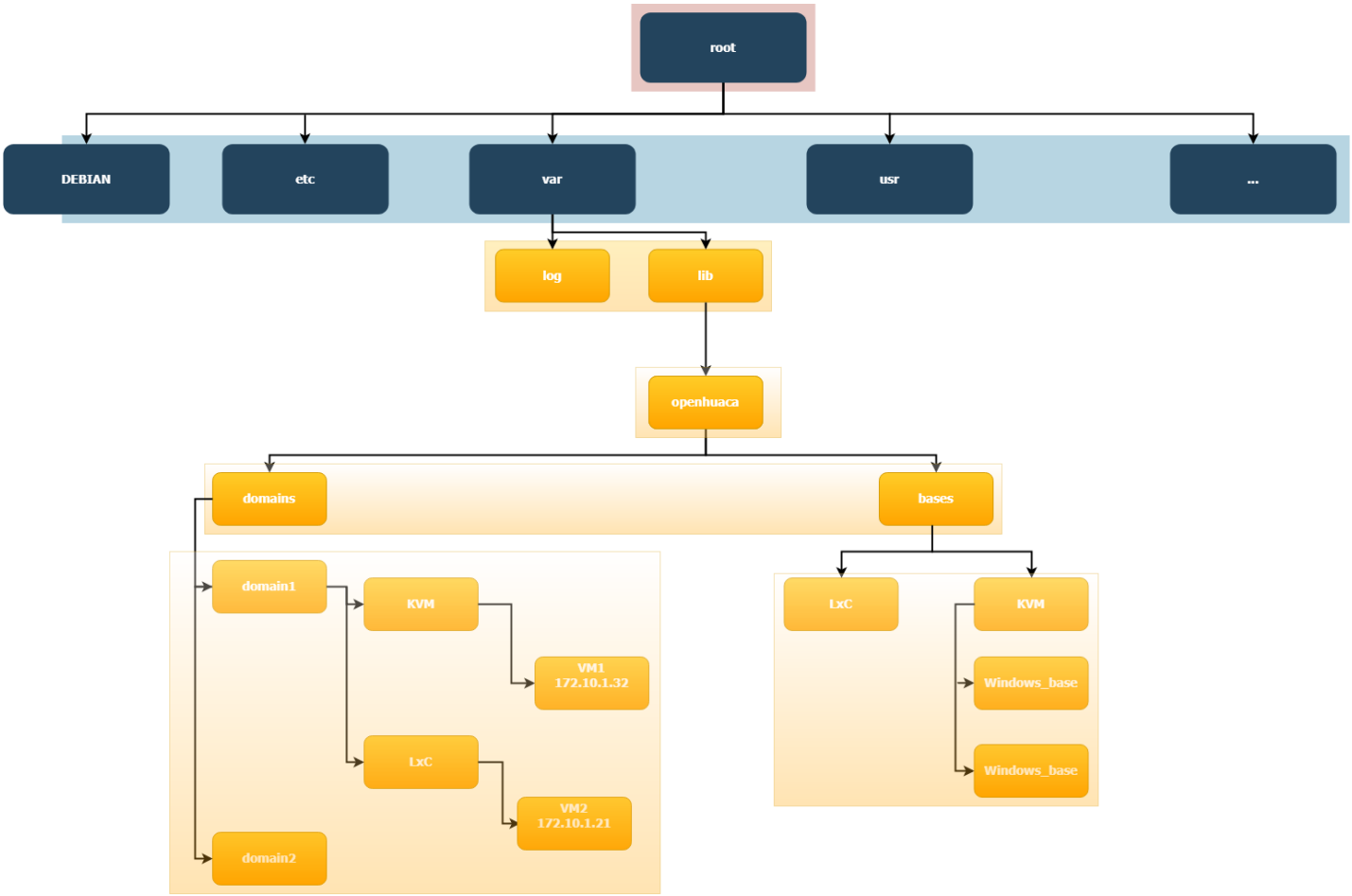

Figure 6.- Openhuaca's directories distribution

As it's shown, blue boxes are the default directories of a Linux OS excepting the DEBIAN one. Meanwhile, yellow boxes are the directories created by Openhuaca. Then, if a KVM base is created (in figure 6, p.eg. Windows_base) it is located in `/var/lib/openhuaca/bases/KVM/` and, if create a VM on a certain domain is created it will be available on location `/var/lib/openhuaca/domain/KVM/VM`

Due to Openhuaca can run different types of virtualization solutions, it is a must to distinguish them when running a command. Openhuaca manages it automatically just finding the guest out in the related path. This is, if VM is in KVM list machines of `domain_path + / kvm`, then the agent called in the command is KVM and, therefore, the commands developed to manage KVM guest will be applied. Quite the opposite, the

agent called in the command will be LxC which will be managed with the LxC Openhuaca methods.

In 4.1Openhuaca commands further information about the commands available to manage KVM guests with this platform can be found. The procedure to follow when running KVM guests in Openhuaca is:

1. Create a domain

In Openhuaca, creating a domain means creating a virtual switch with a certain IP range totally independent of the rest of other possible domains. This domain can define as guests as @IP available has. To create a domain:

```
openhuaca domain  --create domain
```

This command will create a new stopped domain. It is possible to run it with `--start domain` option.

Domains can be listed in two ways:

```
openhuaca ls -d domain
```

To list all VM of a certain domain

```
openhuaca ls -f -d domain
```

To list all the domains and its VM.

By typing the second command the user will receive an output like the showed in the following image.

```
[STOPPED] dom1
+------+------+-----------+--------+------+
| NAME | TYPE | AUTOSTART | STATUS | IPv4 |
+------+------+-----------+--------+------+
+------+------+-----------+--------+------+

[RUNNING] dom2
+--------+------+-----------+---------+-------------+
|  NAME  | TYPE | AUTOSTART |  STATUS |     IPv4    |
+--------+------+-----------+---------+-------------+
| huaca2 | lxc  |     0     | RUNNING | 172.20.1.28 |
+--------+------+-----------+---------+-------------+

[RUNNING] dom3
+----------+------+-----------+---------+-------------+
|   NAME   | TYPE | AUTOSTART |  STATUS |     IPv4    |
+----------+------+-----------+---------+-------------+
| oh_rocks | kvm  |     0     | RUNNING | 172.30.0.98 |
|  new_oh  | lxc  |     0     | STOPPED |      -      |
| new_oh3  | lxc  |     0     | STOPPED |      -      |
+----------+------+-----------+---------+-------------+
```

Figure 7.- Openhuaca list domains

It is also possible to list the bases available for create a guest with the command

```
openhuaca base_list
```

Thus, guests can be distinguished by its base. They can be created with the command

```
openhuaca create –b base –d domain –n name
```

It is necessary to specify where the domain is created and which the domain's name is. Domain's name must be unique. To verify the name, openhuaca checks in the proper path if exists an instance called with this name. If not, a new instance with this name is created taking into account what its base is to decide which guest type is.

The next step is work with domains, openhuaca allows management of VMs. For example, to start a domain:

```
openhuaca start –d domain –n name
```

To stop a domain:

```
openhuaca stop –d domain –n name
```

3.2.3.6. Managing of KVM guest's naming in Openhuaca

Openhuaca is designed to relate the hostname with the VM UUID. It is created a .config file to manage the list of instances in a certain domain. Due to KVM guests are defined from a generic and basic OS image stored into a qcow2 file; initially, these machines have the hostname by default setted up in their base. [24]

In order to change the default hostname by the proper one, in KVM directory where the qcow2 of the VM is storaged, it is necessary to mount the QEMU/KVM disk image. To do so, first of all it is necessary to add a LKM of NBD protocol

```
modprobe nbd max_part = 8
```

Then, share the disk on the network and create the device entries

```
qemu-nbd –connect=/dev/nbd0 /path/to/kvm/images/file.qcow2
```

and mount it:

```
mount /dev/nbd0p1 /path/to/kvm/images
```

At this point, we are able to change the `/etc/hostname` and `/etc/hosts` to the proper configuration. When done, umount and unshare it

```
umount /path/to/kvm/images
nbd-client –d /dev/nbd0
```

3.2.3.7. Set up Serial TTY in KVM guests

It is possible to access to KVM guest directly the Serial Console interface [26], in which case setting up bridged networking or SSH is not necessary. Access via the Serial Console provides an alternate way of accessing the servers to compliment the default VNC access. This can be done using the virsh of libvirt. This console is located in a `/dev/pts/xx` device. To be able to acces to this Serial Console, it is a must to define the proper parameters into the libvirt XML definition. An example of it could be

```
<serial type='pty'>
   <target port='0'/>
 </serial>
 <console type='pty'>
   <target type='serial' port='0'/>
</console>
```

Besides, it is necessary to configure the Serial Console in the Guest. This can be done by enabling the appropriate service. To do so, just access to the guest (e.g. using SSH) and then set up the serial-getty service:

```
systemctl enable serial-getty@ttyS0.service
systemctl start serial-getty@ttyS0.service
```

This procedure must be done every time that a guest is created. Then, there are two ways to implement an automatism which does this configuration on-start by first time a machine. The two ways are based on the previous chapter due to again, is necessary to mount the file system and modify some files.

3.2.3.7.1. Configuration of console access on the target machine with GRUB2 and systemd

If you configure the serial console in GRUB2 systemd will create a getty listener on the same serial device as GRUB2 by default. So, this is the only configuration needed for Arch running with systemd. To make grub enable the serial console, open `/etc/default/grub` in an editor like vim. Change the `GRUB_CMDLINE_DEFAULT` line to start the console on `/dev/ttyS0`. Note in the example below, we set two consoles up; one on tty0 and one on the serial port.

```
GRUB_CMDLINE_LINUX_DEFAULT="console=tty0 console=ttyS0,38400n8"
```

Now we need to tell grub where the console is and what command to start in order to enable the serial console (Note as above for Linux kernel, one can append multiple input/output terminals in grub e.g. `GRUB_TERMINAL="console serial"` would enable both display and serial):

```
## Serial console
GRUB_TERMINAL=serial
GRUB_SERIAL_COMMAND="serial --speed=38400 --unit=0 --word=8 --
parity=no --stop=1"
```

Rebuild the grub.cfg file with following command:

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

After a reboot, getty will be listening on `/dev/ttyS0`, expecting 38400 baud, 8 data bits, no parity and one stop bit. When Arch boots, systemd will automatically start a getty session to listen on the same device with the same settings.

3.2.3.7.2. Configuration of console access on the target machine with system

Ignore this entire section if you have configured GRUB2 to listen on the serial interface. If you do not want GRUB2 to listen on the serial device, but only want getty listening after boot then follow these steps. [27]

To start getty listening on `/dev/ttyS0` start/enable `getty@ttyS0.service`.

You can check to see the speed(s) getty is using with systemctl, but should be 38400 8N1:

```
# systemctl status serial-getty@ttyS0.service
```

Getty will be listening on device `/dev/ttyS0` expecting 38400 baud, 8 data bits, no parity and one stop bit-times.

By the side of client calling console, the python module (libvirt) provides solutions for handling both events and timers. They are invoked through a function interface in order to make easier the integration of the interface into either a graphical or console program.

3.2.3.7.3. Event Handling

The Python libvirt module supplies a framework for event handling. While this is most useful for graphical programs, it can also be used for console programs to provide a consistent user interface and control the processing of console events.

Event handling is done through the functions

`virEventAddHandle, virEventRegisterDefaultImpl, virEventRegisterImpl, virEventRemoveHandle, virEventRunDefaultImpl,`
and `virEventUpdateHandle`.

Creating an event requires that an event loop has previously been registered with

`virEventRegisterImpl` or `virEventRegisterDefaultImpl`.

3.2.3.7.4. Timer Handling

The Python libvirt module supplies a framework for timer handling. Creating a timer requires that an event loop has previously been registered with

`virEventRegisterImpl` or `virEventRegisterDefaultImpl`.

Timer handling is done through the functions

`virEventAddTimeout, virEventUdateTimeout`, and `virEventRemoveTimeout`.

The implementation will support many timers.

To create a new timer call the

`VirEventAddTimout`

After the `virEventRegisterImpl` or the `virEventRegisterDefaultImpl` function has been invoked.

The timer can be removed using the `VirEventRemoveTimout` or updated with the `virEventUpdateTimeout` function after it has been added.[28]

Livbirt provides a code example to use this featores in the proper way as is shown below:

```
# consolecallback - provide a persistent console that survives guest
reboots
from __future__ import print_function

import sys, os, logging, libvirt, tty, termios, atexit

def reset_term():
    termios.tcsetattr(0, termios.TCSADRAIN, attrs)

def error_handler(unused, error):
    # The console stream errors on VM shutdown; we don't care
    if (error[0] == libvirt.VIR_ERR_RPC and
        error[1] == libvirt.VIR_FROM_STREAMS):
        return
    logging.warn(error)

class Console(object):
    def __init__(self, uri, uuid):
        self.uri = uri
        self.uuid = uuid
        self.connection = libvirt.open(uri)
        self.domain = self.connection.lookupByUUIDString(uuid)
        self.state = self.domain.state(0)
        self.connection.domainEventRegister(lifecycle_callback,
self)
        self.stream = None
        self.run_console = True
        logging.info("%s initial state %d, reason %d",
                     self.uuid, self.state[0], self.state[1])

def check_console(console):
    if (console.state[0] == libvirt.VIR_DOMAIN_RUNNING or
        console.state[0] == libvirt.VIR_DOMAIN_PAUSED):
        if console.stream is None:
```

```
            console.stream                                          =
console.connection.newStream(libvirt.VIR_STREAM_NONBLOCK)
            console.domain.openConsole(None, console.stream, 0)

console.stream.eventAddCallback(libvirt.VIR_STREAM_EVENT_READABLE,
stream_callback, console)
    else:
        if console.stream:
            console.stream.eventRemoveCallback()
            console.stream = None


    return console.run_console

def stdin_callback(watch, fd, events, console):
    readbuf = os.read(fd, 1024)
    if readbuf.startswith(""):
        console.run_console = False
        return
    if console.stream:
        console.stream.send(readbuf)

def stream_callback(stream, events, console):
    try:
        received_data = console.stream.recv(1024)
    except:
        return
    os.write(0, received_data)

def   lifecycle_callback   (connection,   domain,   event,   detail,
console):
    console.state = console.domain.state(0)
    logging.info("%s transitioned to state %d, reason %d",
                 console.uuid, console.state[0], console.state[1])

# main
if len(sys.argv) != 3:
```

```
    print("Usage:", sys.argv[0], "URI UUID")
    print("for example:", sys.argv[0], "'qemu:///system' '32ad945f-7e78-c33a-e96d-39f25e025d81'")
    sys.exit(1)

uri = sys.argv[1]
uuid = sys.argv[2]

print("Escape character is ^]")
logging.basicConfig(filename='msg.log', level=logging.DEBUG)
logging.info("URI: %s", uri)
logging.info("UUID: %s", uuid)

libvirt.virEventRegisterDefaultImpl()
libvirt.registerErrorHandler(error_handler, None)

atexit.register(reset_term)
attrs = termios.tcgetattr(0)
tty.setraw(0)

console = Console(uri, uuid)
console.stdin_watch = libvirt.virEventAddHandle(0, libvirt.VIR_EVENT_HANDLE_READABLE, stdin_callback, console)

while check_console(console):
    libvirt.virEventRunDefaultImpl()
```

3.3. Documentation

The documentation side has been carry out in a simple way, although maintaining the philosophy of open source, so all the documentation has been written with LibreOffice.

Once all the documentation was done, it was sent to the project's supervisor for validation. Finally, when the documents were approved by the tutor a PDF version was print and delivered in the UPC Moodle service.

In order to share all the documentation with the supervisor, a SVN repository is available in the j3o machine, so all files were saved there in both editable and PDF version.

3.4. Troubleshooting

This section is composed of the most representative cases of failure when running KVM (on libvirt) guests.

3.4.1. The daemon cannot be started

If the libvirt daemon cannot be started we will receive an output like:

```
# /etc/init.d/libvirtd start
* start-stop-daemon: failed to start `/usr/sbin/libvirtd'
[ !! ]
* ERROR: libvirtd failed to start
```

However, if there's nothing in the /var/log/messages, it is strongly recommendable to change the libvirt logging:

```
/etc/libvirt/libvirtd.conf:
     #Uncommend this line
     Log_outputs="3:syslog:libvirtd"
```

Once, try again to start the daemon and it is possible to see the error in `/var/log/messages`:

As an example, we can obtain the following message:

```
May 8 17:22:09 bart libvirtd: 17576: info : libvirt version: 0.9.9
May 8 17:22:09 bart libvirtd: 17576: error :
virNetTLSContextCheckCertFile:92: Cannot read CA certificate
'/etc/pki/CA/cacert.pem': No such file or directory
May 8 17:22:09 bart /etc/init.d/libvirtd[17573]: start-stop-
daemon: failed to start `/usr/sbin/libvirtd'
May 8 17:22:09 bart /etc/init.d/libvirtd[17565]: ERROR: libvirtd
failed to start
```

Which means that file used as TLS authority is missing when libvirt is run in 'Listen for TCP/IP connections' mode.

To solve this issue, just install the correct CA certificate. Be aware of not using the TLS but bare TCP instead, set `listen_tls = 0` and `listen_tcp = 1` in

`/etc/libvirt/libvirtd.conf` and change the `LIBVIRTD_ARGS` variable in `/etc/sysconfig/libvirtd` to don't pass the `–listen`. [23]

3.4.2. Failed to connect to the hypervisor

There are lots of errors that can occur while connecting to the server. Then, in this subchapter just are described the two more common cases.

- **No connection driver available**

  This can happen when libvirt is compiled from sources. The error means there is no driver to use with the specified URI. (e.g. "qemu" for "qemu://server/")

  Check if the last part of configure shows like this:

```
configure: Drivers
configure:
configure: <driver>: yes
```

If <driver>: no, that means configure failed to find all the tools or libraries to implement this support or there was the `–without <driver>` flag active. So, to solve just do not specify this flag on the command line configuration script and configure the sources again.

- **Permission denied**

  The following message uses to appear:

```
error: Failed to connect socket to '/var/run/libvirt/libvirt-sock': Permission denied
error: failed to connect to the hypervisor
```

The connection to QEMU without any hostname specified is by default using unix sockets. If there is no error running this command as root it is probably just misconfigured.

To connect as non-root user using UNIX sockets, configure following options in `/etc/libvirt/libvirtd.conf` accordingly:

```
unix_sock_group = <group>
unix_sock_ro_perms = <perms>
unix_sock_rw_perms = <perms>
```

3.4.3. Common XML errors

XML files are used by libvirt to store structured data. So, it is a must to be aware of mis-formatted XML documents, inappropriate values or missing elements because it may produce errors. Fortunately, libvirt has `xml-validate file.xml` method to validate if the file is properly defined. If there are no errors then the description is well-formed from an XML point of view and matches the libvirt schema.

XML documents stored by libvirt contain definitions of domains (guests), their states and configurations. All of those documetns are automatically generated by Openhuaca when a KVM guest is created. The file name is valid only on the host machine defined by the URI. So, it may be the machine command is run on.

Libvirt developers maintain a set of XML schemas bundled with libvirt and defining as much as possible the constructs allowed in XML documents used in libvirt. If XML validation is okey, then chances are that libvirt will understand all constructs from your XML, thought this doesn't means that the XML file is 100% correct, for example, schemas cannot detect options which are valid only for a given hypervisor.

3.4.4. No guest machines are present

No virtual machines are present when appeal to the Openhuaca listing method although the daemon was successfully started.

First, it is convenient to verify if KVM kernel modules are really inserted in the kernel

```
$ lsmod | grep kvm
kvm_intel       131247 0
kvm             324632 1 kvm_intel
```

If you have an AMD machine, `kvm_amd` instead of `kvm_intel`.

`modprobe <modulename>` can be used to insert the modules.

If it is correct, try verify that virtualization extensions are supported and enabled on the host. Enabling virtualization extension on the host hardware's firmware configuration is a must to run KVM virtualization:

```
$ egrep "(vmx|svm)" /proc/cpuinfo
flags      : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv
svm_lock nrip_save
flags      : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv
svm_lock nrip_save
```

At last, it should be verified that the URI of the client is correct (e.g. `qemu: ///system`). There may be other hypervisors present and libvirt will talk to them by default.

3.4.5. Domain starting fails with Error "monitor socket did not show up"

When starting a domain, we obtain an output similar to

```
error: Failed to create domain from vm.xml error: monitor socket
did not show up.: Connection refused
```

This can means although libvirt works well, qemu process fails to start up then, libvirt quits when trying to connect to qemu or qemu agent monitor socket. But, to known the exact error, it `/var/log/libvirt/qemu/vm.log`

In most cases, the error in guest log could tell what happened and it's the best way to fix the problem according to the error. One thing to take into account (because is very common issue) is if a host was shut down while the guest was running the libvirt-guests init script attempted to perform a managed save of the guest. The save image is corrupted due to an incomplete save image process and it will not be loaded by qemu. In this failure case, the guest log will show an attempt to use "-incoming" as one of its arguments which means that libvirt is trying to start qemu by migrating the saved state file. Newer libvirt versions takes steps to avoid the corruption, as well as forcing-boot of the domain as a way to bypass any managed save image.

4. Results

The final result of this TFG project is the Openhuaca-1.6 version. In this version it is possible to create an LxC wrapper, so it has exacly the same functionalities of LxC but realized with the openhuaca commands. But this feature has been able since 1.1 version. So, the main improvement for 1.6 version is the integration of KVM into Openhuaca Cloud Platform. Now Openhuaca is able to manage both LxC and KVM guests' machines by just typing openhuaca commands.

In the Annexes part, more detailed information about protocols, libvirt API for Pyhon and so on can be found. It is also attached the installable package of Openhuaca, the source files and the Developer's Guide, so there is the entire source code of the latest version.

This document has the purpose of be a significant guide to understand how KVM virtualization is used by Openhuaca and how to troubleshoot in case of failure. Due to a certain issue can be caused by so many random causes, the troubleshooting is a highlight of the more representative cases.

In this section, there is a brief summary about the commands and the basic configuration of Openhuaca environment. All the detailed information of each command and a guide to create any type of environment can be found in the attached documentation to this project. In addition, there is also technical information explanation of how each functionality has been performed.

4.1. Openhuaca commands

In this section there's a brief description about every command and it is specified the parameters needed by all of them. Finally, there's included at least one typical example of functionality. Nowadays, Openhuaca works on CLI, so all the interactions between the user and the product are implemented with python commands.

Goal of the command	**Command**
Create a base	`openhuaca base-create kvm –n name –iso os.iso`
List bases	`openhuaca base-list`
Create a domain	`openhuaca domain --create domain`
Start a domain	`openhuaca domain --start domain`
Stop a domain	`openhuaca domain --stop domain`
Create a guest	`openhuaca create –b base –d domain –n instance`
Start a guest	`openhuaca start –d domain –n instance`
Stop a guest	`openhuaca stop –d domain –n instance`
Freeze a guest	`openhuaca freeze –d domain –n instance`
Unfreeze a guest	`openhuaca unfreeze –d domain –n instance`
Start the guest 'console	`openhuaca console –d domain –n instance`
List domain guests	`openhuaca ls –d domain`
List guests (all domains)	`openhuaca ls –f`

Table 6.- Result command summary

The whole list of Openhuaca commands and detailed description of each one can be found in section 10.1.

5. Budget

This project has not required any kind of prototype or hardware in order to be developed. All the software used and implemented has been done with open source programs. Only has needed a software development team and a container of j3o networking department.

This is why in the following economic study is just considered that a computer has been purchased to carry out the project development and the fact of being a research project the university has lent all the resources to create the container on the j3o server.

Item	*Cost (€)*
i5 computer with 8GB RAM 1TB SDD	799
Open source software (OS, Python, Openstack, Libvirt…)	0
Jose Luis Muñoz (Project manager)	45€/h (part-time during 4 months)
Jorge Buzzio (QA tester)	30€/h (10h dedicated to each review)
Carla Brugulat (web developer)	30 €/h (full-time for 4 months)
Rafa Genés (QA tester)	30€/h (10h dedicated to each review)
Dani Capdevila (Ansible prov. developer)	30 €/h (full-time for 4 months)
Daniel Campos (KVM integrator)	30 €/h (full-time for 4 months)
TOTAL INVESTMENT	**131.599**

Table 7.- Project budget

6. Environment Impact

Integration of KVM in Openhuaca Cloud Platform (and Openhuaca project) does not have too much affect to the environment because it is based on coding and software developing.

Creating virtual environments requires less hardware than having real environments. So, it requires less equipment but more powerful, which reduces the consumption of materials. By contrast, having more powerful equipment, the levels of energy consumed grow. Then, Openhuaca is engaged to find the correct balance between these two concepts explained.

7. Conclusions and future development:

This should include your summary, conclusions and recommendations.

7.1. Conclusions

This project was initially thought to create an extended version of LxC, but, once seen the potential of Openhuaca, supervisor decided to add new virtualization technologies such as KVM.

Thanks to the support of my team-workers and supervisor I have took the most of me to learn quick and effective how to program in python just like some new technologies of virtualization and its features. Besides, during the project I think that I have improved my team-work skills and self-learning to face a realistic engineering problem. In addition, I told with the supervisor to apply for continue as an Openhuaca developer to implement the new features for the platform.

Nowadays, I can conclude that thanks to KVM implementation, Openhuaca is now a multi-tenant Cloud Platform with multiple virtualization technologies which allows to the user work with their favourite OS in a simple way.

7.2. Future Development

7.2.1. Snapshots with libvirt

Using libvirt, there are 3 ways to take a guest snapshot [18] . Due to the domain information is enclosed into XML files, taking a snapshot means taking XML snapshot. So you can take:

- Disk snapshot: there are two ways to take a snapshot, internal, using for example qcow2 track both the snapshot and changes since the snapshot in a single file. Another way is the external where the snapshot is one file, and changes since the snapshot are in another file.
- VM state: tracks only the state of RAM and all other resources in use by the guest.
- System checkpoint: It is very similar to hibernation, it's a combination of disk snapshots for all disks as well as VM memory state and it are used to resume the guest.

Snapshots are maintained in a hierarchy. A domain can have a current snapshot, which is the most recent snapshot compared to the current state of the domain (although a domain might have snapshots without a current snapshot, if snapshots have been deleted in the meantime). Creating or reverting to a snapshot sets that snapshot as current, and the prior current snapshot is the parent of the new snapshot. Branches in the hierarchy can be formed by reverting to a snapshot with a child, then creating another snapshot.

The domain snapshot may contain information like the name for this snapshot (used to be based on the time when it was created), a description, memory, disks, etc.

As an example, using this XML configuration to create a disk snapshot of just vda on a qemu domain with two disks:

```
<domainsnapshot>
  <description>Snapshot of OS install and updates</description>
  <disks>
    <disk name='/path/to/old'>
      <source file='/path/to/new'/>
    </disk>
    <disk name='vdb' snapshot='no'/>
  </disks>
</domainsnapshot>
```

Note `/path/to/old` is the read-only backing file to the new active file `/path/to/new`. The domain element within the snapshot XML records the state of the domain just before the snapshot. The domainsnapshot element contains child elements of the libvirt snapshot.

7.2.2. etcd

The name ECTD [19] is referred to a “d”istribuited “/etc”. ETCD can be more technically described as a daemon that runs across all computers in a cluster, providing a dynamic con_guration registry. This kind of setting allows various con_guration data to be easily and reliably shared between the cluster members. All changes in stored data are re_ected across the entire cluster because, as said before, the key_value data stored within ETCD is automatically distributed and replicated with automated master election and consensus establishment using the Raft algorithm. On the other hand, the achieved redundancy prevents failures of single cluster members from causing data loss. Moreover, ETCD is written in Go, which has excellent cross-platform support and small binaries.

However, latency in ETCD nodes is the most important metric to track, due to the fact that a severe latency would introduce instability within the cluster because Raft is only as fast as the slowest node in the majority. The best practice to avoid this problem, and make the con_guration as reliable as possible, is properly tuning the cluster, establishing a proper cluster size, controlling members status, preparing a backup and so on. ETCD has been pre-tuned on cloud providers with highly variable networks, and handles leader elections during network partitions, tolerating machine failure, including the leader.

7.2.3. Additional future development

In addition of the improvements mentioned before, there are some tasks that are in progress or in a very initial stage.

1. Finish certificate commands in Python (Work in progress by Jorge Buzzio)
2. Code testing (make some stress challenges to openhuaca environment to see its behavior)
3. Web Services like be able to access to command prompt via web or more services configured in Openhuaca VMs just by accessing with web browser (https://host/domain/instance1/serviceX)
4. Improve the help command, almost the way in which the help is shown to the user.
5. MVP: stablish a date to deliver openhuaca as an open-source product opened to the public.
6. Add NAT

For example, if there is configured a web site in some guest, be able to access to it from Internet.

Bibliography

[1] Archlinux. "DNSMASQ". *Open source DNS software*, 2018. [Online] Available: https://wiki.archlinux.org/index.php/dnsmasq#Configuration. [Accessed: 02 May 2018].

[2] Lallinaho, Pasi. "Libvirt". *Libvirt*, 2014. [Online] Available: https://help.ubuntu.com/lts/serverguide/libvirt.html. [Accessed: 02 May 2018].

[3] Debiman "QEMU-SYSTEM-x86_64". *qemu-system-x86_64,* 2017. [Online] Available: https://manpages.debian.org/stretch/qemu-system-x86/qemu-system-x86_64.1.en.html. [Accessed: 03 May 2018].

[4] RedHat. "Enterprise Virtualization". *RedHat Enterprise Virtualization*, 2018. [Online] Available: https://access.redhat.com/documentation/en-US [Accessed: 07 May 2018].

[5] RedHat Inc and others. "Libvirt Application Development Guide Using Python".*A guide to libvirt application development with Python*, 2016. [Online] Available: https://libvirt.org/docs/ [Accessed: 12 March 2018].

[6] Palo Alto Networks. "Virtualization with KVM". *VM-Series System Requirements,* 2018. [Online] Available: https://www.paloaltonetworks.com/documentation/81/virtualization/virtualization/about-the-vm-series-firewall/vm-series-models/ [Accessed: 23 May 2018]

[7] Ubuntu. "Virtualization with KVM". *Information about what CPU supports Hardware virtualization,* 2016. [Online] Available: http://www.linux-kvm.org/page/Processor_support [Accessed: 12 May 2018]

[8] Linux-kvm.org "Troubleshooting". *How can I check that I'm not falling back to QEMU with no hardware acceleration?,* 2015. [Online] Available: http://www.linux-kvm.org/page/FAQ#How_can_I_check_that_I.27m_not_falling_back_to_QEMU_with_no_hardware_acceleration.3F [Accessed: 06 May 2018]

[9] Ubuntu. "Ubuntu Server Guide". *Virtualization libvirt*, 2018. [Online] Available: https://help.ubuntu.com/lts/serverguide/libvirt.html [Accessed: 18 May 2018]

[10] Debian Org. "LxC". *LxC,* 2018. [Online] Available: https://wiki.debian.org/LXC [Accessed: 23 February 2018]

[11] Docker. "Docker". *What is docker?"* 2018. [Online] Available: https://www.docker.com/what-docker [Accessed: 23 February 2018]

[12] Wikipedia Org. "Kernel-based Virtual Machine". *Kernel-based Virtual Machine,* 2018. [Online] Available: https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine [Accessed: 23 February 2018]

[13] VPS NET Org. "QEMU". *What is QEMU? ,* 2018. [Online] Available: https://www.vps.net/blog/what-is-qemu/ [Accessed: 23 February 2018]

[14] Rafael Genés. "Development of a multi-tenant cloud platform based on OS containers". Thesis, Faculty of ETSETB, Spain, 2017.

[15] J. Romkey. "A NONSTANDARD FOR TRANSMISSION OF IP DATAGRAMS OVER SERIAL LINES: SLIP". *Serial Line IP*, 2018. [Online] Available: https://tools.ietf.org/html/rfc1055 [Accessed: 04 June 2018]

[16] StackExchange Org. "Why is using an SSH key is more secure than using passwords?". *Why is using an SSH key is more secure than using passwords?*, 2014. [Online] Available: https://security.stackexchange.com/questions/69407/why-is-using-an-ssh-key-more-secure-than-using-passwords [Accessed: 06 June 2018]

[17] SuperUSer Org. "Why is using an SSH key is more secure than using passwords?". *Why is using an SSH key is more secure than using passwords?*, 2011. [Online] Available: https://superuser.com/questions/303358/why-is-ssh-key-authentication-better-than-password-authentication [Accessed: 06 June 2018]

[18] Red Hat Inc. "Snapshot XML Format". *Snapshot XML*, 2017. [Online] Available: https://libvirt.org/formatsnapshot.html [Accessed: 07 June 2018]

[19] Sonia Rivas. "Development of a cluster of LxC containers". Master Thesis, Faculty of ETSETB, Spain, 2017

[20] Diego Ongaro and John Ousterhout. Standford University. *In search of an Understable Consensus Algorithm (Extended Version)*, 2014.

[21] Proxmox. "Virtualization". *Open-Source Virtualization Platform,* 2016. [Online] Available: https://www.proxmox.com/en/proxmox-ve [Accessed: 08 June 2018]

[22] W. David Ashley, D. Berrange, C. Lalancette, L. Stump, D. Veillard, D. Coulson, D. Jorm, S. Radvan. *Libvirt Application Development Guide Using Python,* 2016.

[23] Red Hat Inc and other, “Troubleshooting”. *Troubleshooting*, 2016. [Online] Available: https://wiki.libvirt.org/page/Troubleshooting [Accessed: 25 June 2018]

[24] Kumari.net, “Mounting a QEMU Image”. *Mounting a QEMU Image,* 2015. [Online] Available: https://www.kumari.net/index.php/system-administration/49-mounting-a-qemu-image [Accessed: 27 June 2018]

[25] Wouter Verhest, “Network Block Device”. *Network Block Device,* 2006. [Online] Available: https://nbd.sourceforge.io/ [Accessed: 27 June 2018]

[26] Ubuntu. “KVM”. *KVM Access,* 2016. [Online] Available: https://help.ubuntu.com/community/KVM/Access [Accessed: 23 May 2018]

[27] Arch Linux, “Serial Console”. *Working with serial console*, 2018 [Online] Available: https://wiki.archlinux.org/index.php/working_with_the_serial_console [Accessed: 27 June 2018]

[28] Red Hat Inc. “Chapter 9. Event and Timer Handling”. *Event and Timer Handling,* 2017 [Online] Available: https://libvirt.org/docs/libvirt-appdev-guide-python/en-US/html/libvirt_application_development_guide_using_python-Timer_Handling.html [Accessed: 27 June 2018]

Appendices

8. Appendix I

8.1. SLIRP Protocol

Danny Gasparovsky released SLIRP protocol in 1995. It is a TCP / IP emulator, which turns an ordinary shell account into a SLIP/PPP account. This allows shell users (or guest instances in this case) to use all the Internet applications like Netscape, Mosaic, CUSeeMe, etc. So, it can be understood as Serial Line Internet Protocol (SLIP) that is an encapsulation of the IP (lighter than IP due to SLIP has less overhead) designed to work over serial ports and modem connections. Another protocol more used in PCs, is the Point-to-Point Protocol.

So, SLIP protocol is commonly used on dedicated serial links. It is useful for allowing mixes of hosts and routers to communicate with one another. However, in virtualization, it is used to establish a reliable communication between a guest and its serial console. It is available for most UNIX systems and some terminal concentrators and IBM PC implementations.

Then, this protocol defines two special characters: END and ESC. In fact, libvirt recognize the ESC character as ^ [scape sequence. Deep down, the SLIP protocol recognize these two characters as:

HEX VALUE	OCT VALUE	ABBREVIATION	DESCRIPTION
0xC0	300	END	Frame End
0xDB	333	ESC	Frame Escape
0xDC	334	ESC_END	Transposed Frame End
0xDD	335	ESC_ESC	Transposed Frame Escape

Table 8 .- Special characters to SLIP Protocol

Should be noticed that the ESC character of SLIP is not the ASCII ESCape character, ESC will just indicate the SLIP ESC character. To send a packet, a SLIP instance just starts sending the data in the packet. If a data byte is the same code as END character, a two-byte sequence of ESC and hex value 0xDC is sent instead. When the last byte character has been sent, an END character is then transmitted. The sized of the SLIP packets is not specified, but is usual to work with size packets of 1006 bytes including the IP and transport protocol headers but no the framing characters.

As anecdote, Phil Karn suggests begin as well as end packets with an END character. This will flush any erroneous bytes that have been caused by line noise. In the normal case, the receiver will simply see two back-to-back END characters, which will generate a bad IP packet. If there was line noise, the data received due to it will be discarded without affecting the following packet.

SLIP has no type field. Thus, only one protocol can be run over a SLIP connection. If a serial line connects two multi-protocol computers, those computers should be able to use more than one protocol over the line.

Usually, streams of packets in a single TCP connection have few changed fields in the IP and TCP headers, so, since SLIP uses a simple compression algorithm, it might just send the changed parts of the headers instead of the complete headers.

In RFC1055 [15] can be found a C example code that shows how the protocol is used. To see the packets flowing just tap with Wireshark between two instances both must have IP defined.

```
/* SLIP special character codes
    */
   #define END             0300    /* indicates end of packet */
   #define ESC             0333    /* indicates byte stuffing */
   #define ESC_END         0334    /* ESC ESC_END means END data byte */
   #define ESC_ESC         0335    /* ESC ESC_ESC means ESC data byte */

   /* SEND_PACKET: sends a packet of length "len", starting at
    * location "p".
    */
   void send_packet(p, len)
           char *p;
           int len; {

     /* send an initial END character to flush out any data that may
      * have accumulated in the receiver due to line noise
      */
        send_char(END);

     /* for each byte in the packet, send the appropriate character
      * sequence
      */
          while(len--) {
                  switch(*p) {
                /* if it's the same code as an END character, we send a
                   * special two character code so as not to make the
                   * receiver think we sent an END
                   */
                  case END:
                          send_char(ESC);
                          send_char(ESC_END);
                          break;

                  /* if it's the same code as an ESC character,
                   * we send a special two character code so as not
                   * to make the receiver think we sent an ESC
                   */
                  case ESC:
                          send_char(ESC);
                          send_char(ESC_ESC);
                          break;

                  /* otherwise, we just send the character
                   */
                  default:
                          send_char(*p);
                              }
```

```
                p++;
                }



        /* tell the receiver that we're done sending the packet
         */
        send_char(END);
        }

/* RECV_PACKET: receives a packet into the buffer located at "p".
 *      If more than len bytes are received, the packet will
 *      be truncated.
 *      Returns the number of bytes stored in the buffer.
 */
int recv_packet(p, len)
        char *p;
        int len; {
        char c;
        int received = 0;

        /* sit in a loop reading bytes until we put together
         * a whole packet.
         * Make sure not to copy them into the packet if we
         * run out of room.
         */
        while(1) {
                /* get a character to process
                 */
                c = recv_char();

                /* handle bytestuffing if necessary
                 */
                switch(c) {

                /* if it's an END character then we're done with
                 * the packet
                 */
                case END:
                        /* a minor optimization: if there is no
                         * data in the packet, ignore it. This is
                         * meant to avoid bothering IP with all
                         * the empty packets generated by the
                         * duplicate END characters which are in
                         * turn sent to try to detect line noise.
                         */
                        if(received)
                                return received;
                        else
                                break;

                /* if it's the same code as an ESC character, wait
                 * and get another character and then figure out
                 * what to store in the packet based on that.
                 */
                case ESC:
```

```
        c = recv_char();

                /* if "c" is not one of these two, then we
                 * have a protocol violation.  The best bet
                 * seems to be to leave the byte alone and
                 * just stuff it into the packet
                 */
                switch(c) {
                case ESC_END:
                        c = END;
                        break;
                case ESC_ESC:
                        c = ESC;
                        break;
                        }

            /* here we fall into the default handler and let
             * it store the character for us
             */
            default:
                    if(received < len)
                            p[received++] = c;
                    }
            }
    }
```

9. Appendix II

9.1. Requirements for KVM installation

9.1.1. Hardware Resources [6]

VM-Series Model	Supported Hypervisors	Supported vCPUs	Minimum Memory	Minimum Hard Drive
VM-50	ESXi, Hyper-V,KVM	2	4.5 GB (4 GB in Lite mode)	32 GB (60 GB at boot)
VM-100 **VM-200**	AWS, Azure, ESXi, Google Cloud Platform, Hyper-V, KVM, NSX	2	6.5 GB	60 GB
VM-300 **VM-1000-HV**	AWS, Azure, ESXi, Google Cloud Platform, Hyper-V, KVM, NSX	2 4	9 GB	60 GB
VM-500	AWS, Azure, ESXi, Google Cloud Platform, Hyper-V, KVM, NSX	2 4 8	16 GB	60 GB
VM-700	AWS, Azure, ESXi, Google Cloud Platform, Hyper-V, KVM, NSX	2 4 8 16	56 GB	60 GB

Table 9.- Hardware Resources for KVM installation

9.1.2. Software versions [6]

Software	Version
UBUNTU	14.04 LTS (QEMU-KVM 2.0.0 and libvirt 1.2.2) 16.04 LTS (QEMU-KVM 2.5.0 and libvirt 1.3.1)
CentOS / RedHat Enterprise Linux	7.2 (QEMU-KVM 1.5.3 and libvirt 2.0.0)

Table 10.- Software versions supporting KVM

9.2. Requirements for libvirt installation

9.2.1. Hypervisor drivers

There are so many hypervisor drivers currently supported by libvirt. The most common ones are:

LXC (Linux Containers)	QEMU	UML (User Mode Linux)
Bhyve (The BSD Hypervisor)	Microsoft Hyper-V	VMWare ESX
VirtualBox	Xen	IBM PowerVM (phyp)

10. Appendix III

10.1. Openhuaca Commands

10.1.1. Bases

base_create Create a new base (to use as a template to create containers).

usage: `openhuaca base-create kvm -n NAME --iso miiso.iso`

usage: `openhuaca base-create lxc -n NAME -t TEMPLATE`

base_rename Change the name of a base to another new one.

usage: `openhauca base-rename NAME NEW_NAME`

base_destroy Destroy a base and ask you if you are sure about it.

usage: `openhuaca base_destroy -n NAME [-f FORCE]`

base_list Show a list of the existent bases.

usage: `openhuaca base_list`

10.1.2. Containers

autostart Manage auto-started containers.

usage: `openhuaca autostart [-d DOMAIN] [--kill] [--list] [--reboot] [--shutdown] [--all] [--ignore-auto] [--groups] [-t TIMEOUT] [--quiet]`

certification [KO] To be revised in the next Openhuaca version

cgroup Gets or sets the value of a state-object in the container's control group. It's main porpouse is to manage the resources offered to every huaca container.

usage: `openhuaca cgroup -n H_CONTAINER -d H_DOMAIN {state-object} [value]`

checkconfig It shows the whole information related to the environment configuration. (Inherited from LXC)

usage: `openhuaca checkconfig`

checkpoint Checkpoints and restores a container. Serialize a conteiner's running state to a disk to allow restoring it on its running state later. (Inherited from LXC)

usage: `openhuaca checkpoint -n H_CONTAINER -d H_DOMAIN [-r]`

usage: `openhuaca checkpoint -n H_CONTAINER -d H_DOMAIN [-D CHECKDIR]`

console Login inside huaca conteiner console.

usage: `openhuaca console -n H_CONTAINER -d H_DOMAIN`

copy Create a copy of a huaca container.

usage: `openhuaca copy -n H_CONTAINER -d H_DOMAIN -n NEWNAME -p NEWPATH`

create To create a new container.

usage: `openhuaca create -n H_CONTEINER -d H_DOMAIN`

destroy Destroy a huaca container. Asks you if you are sure about.

usage: `openhuaca destroy -n H_CONTAINER [-d H_DOMAIN] [--snapshot] [--force]`

10.1.3. Domains

Domain Command to manage the domains. Lets the user to create, delete, list, start, stop or restart domains.

usage: `openhuaca domain [-h] [--create H_DOMAIN] [--destroy H_DOMAIN][--start H_DOMAIN] [--start-all] [--stop H_DOMAIN] [--stop-all] [--restart H_DOMAIN] [--restart-all] [--list] [--status]`

10.1.4. Miscellaneous

attach Execute the specified COMMAND in the lxc container NAME DOMAIN

usage: `openhuaca attach -n H_CONTAINER -d H_DOMAIN - COMMAND`

top Shows statistics in real time.

usage: `openhuaca top -d H_DOMAIN`

start Start a specific container.

usage: `openhuaca start –d domain –n instance`

stop Stop a specific container.

usage: `openhuaca stop –d domain –n instance`

freeze Freeze the specific conteiner.

usage: `openhuaca freeze –d domain –n name`

unfreeze Unfreeze the specific conteiner.

usage: `openhuaca unfreeze -d domain -n name`

info Display some information about the specific container.

usage: `openhuaca info -n H_CONTAINER -d H_DOMAIN`

ls Show a summary of the created domains and containers.

usage: `openhuaca info [-d H_DOMAIN]`

monitor Actively monitors the state of the container. You can be able to create a register of the states of every machine.

usage: `openhuaca monitor -n H_CONTAINER [-d H_DOMAIN] [--quit]`

rebase [KO] To be revised in the next Openhuaca version

snapshot Manages the snapshots. Can copy, restore or delete them.

usage: `openhuaca snapshot -n NAME [-d H_DOMAIN] [--list] [[-r RESTORE][-N NEWNAME]] [-D DESTROY] [-c COMMENT] [-C SHOWCOMM] [--rcfile RCFILE]`

Wait Waits for a huaca container state to reach a STATE. (Inherited from LXC)

usage: `openhuaca wait -n H_CONTAINER -d H_DOMAIN --state STATE --timeout TMO`

10.2. Detailed Openhuaca Usage Commands

```
OPENHUACA attach [-h] -n NAME [-d DOMAIN] [- COMMAND] [-e EPRIV] [-a ARCH]
[-s NAMESPACES] [-R] [--clear-env] [--keep-env] [-L PTYLOG] [-v SETVAR]
[--keep-var KEEPVAR] [-f RCFILE] [- - [- ...]] [-q]


OPENHUACA autostart [-h] [-d DOMAIN] [-k] [-L] [-r] [-s] [-a] [-A] [-g]
[-t TIMEOUT] [-q]


OPENHUACA start [-h] -n NAME [-d DOMAIN] [-D] [-p PIDFILE] [-F] [-f RCFILE]
[-c CONSOLE] [-L CONSOLELOG] [-C CLOSE_ALL_FDS] [-s DEFINE] [--share-net
SHARENET] [--share-ipc SHAREIPC] [--share-uts SHAREUTS]


OPENHUACA stop [-h] -n NAME [-d DOMAIN] [-r] [-W] [-t TIMEOUT] [-k] [--
nolock] [--nokill] [-f RCFILE]
```

11. Appendix IV

11.1. Remote access with Secure Shell (SSH)

The goal of this chapter is to study the security access management when accessing to guests.

11.1.1. Login with private key vs password

SSH keys are longer and complex than user's passwords, by default. Besides, private keys are not transmitted to the remote system which passwords needs to be. Due to this fact, it makes password logging more vulnerable than private key logging due to, using some sniffing tools like Wireshark, it is possible to hear the password (Man in the middle). [16] Passwords are so vulnerable to force attack and its protection directly depends on the protection of the server or whatever it uses to verify the passwords (e.g. the `/etc/passwords` file).

12. Appendix V

12.1. Raft Algorithm

Designed [20] as an alternative to the Paxos complexity by the Stanford University laboratory, Raft is another consensus algorithm for managing log replication in a cluster of nodes. While in the Paxos algorithm the problems are mixed in a confusing manner and solved in the same way, Raft tried to separate each subject clearly di_erentiated in order to solve it in phases. However, the general operation of the algorithm is equal to Multi-Paxos. Again, a client executes a command and a cluster node receives it, passing it to its Consensus Module. If this node meets some requirements, it will pass it to other nodes to accept and choose, replicating it in an entry of its log and passing it to its State Machine for execution. At that point, the initial node returns the result that has been generated to the client. We remember that this model of failures only accepts fails or delays in the nodes, but not that there is a malicious one that tries to harm the correct operation of the cluster. In that case, Raft, like Paxos, would fail. In this variant, a new way of dividing the nodes of the cluster was included. The diagram in below figure shows the three possible states of a server, which would be:

1. Leader, which would handle all client requests and the correct replication of the logs. In each round, this server must notify the other nodes that is up and properly operating by means of RPCs, and in case a timeout is exceeded without a response, the cluster would enter a new election period. There can only be one leader in the cluster to avoid con_icts (as it happened with Basic Paxos).

2. Follower, a passive node whose sole purpose is to respond the requests it receives from leaders and candidates for leader. In case of a crashed leader node, it becomes a candidate.

3. Candidate, nodes who have previously been followers and choose to become cluster leaders. If the majority vote them, they increase their rank; if they see that another has been chosen, they return to be followers.

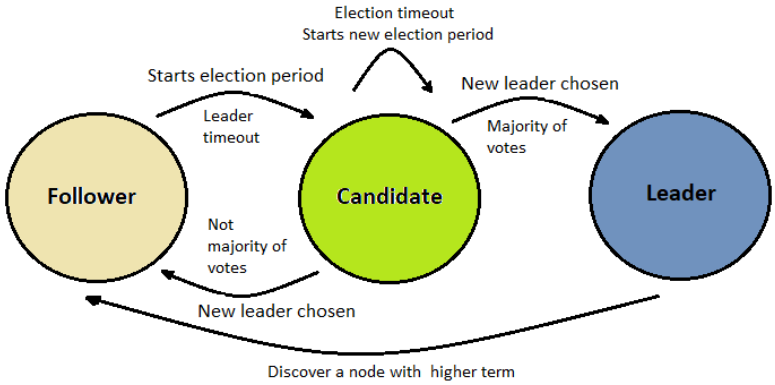

Figure 8.- Raft Algorithm

In Raft, a client contacts the cluster leader to send a command. In case he did not know which server is the leader, it is sent to any node, which would redirect him to the current leader of that term. There is a timeout, which once exhausted without response from the node that it trying to contact, the client would start its request again with another node in the cluster. Then the consensus algorithm would be executed, where the leader will not return the command to the client until it has been committed and executed by its own State Machine. But what would happen if after executing the command, the leader fails and does not return the result to the client? The client again waits until timeout expires, and once exceeded, retries the process with the new leader of the cluster. It is important in order to save resources that this command do not run twice. But in the practice, this would not happen, as future leaders have the old logs as long as they have been committed (and if executed by the State Machine, this is true). So, the new leader, before accepting the new command, would check his id to see if there is a matching log entry. If so, it would ignore the new command and return the result directly to the client, so that it was executed only once.

13. Network Block Device

13.1. What is NBD?

Linux can make use of a remote server as one of its block devices if NBD is compiled into the kernel. Whenever the client computer wishes to read /dev/nd0, a request is sent to the server through TCP. The server then responds with the requested data. This is useful for stations having low disk space (or maybe even diskless, if booted from a floppy) since it allows them to use other computers' disk space.

In contrast to the Network File System (NFS), it is possible to use any file system with NBD. However, if another user has already mounted NBD read/write, one must make sure that no one else mounts it again.

Even though NFS, SMB/CIFS and other similar protocols are useful, they may not be ideal for some requirements. [25]

Glossary

In the below table can be found a list of all acronyms and the meaning they stand for.

Acronym	**Meaning**
LxC	Linux Container
KVM	Kernel-based Virtual Machine
VM	Virtual Machine
QCOW	QEMU Copy-On-Write
OS	Operative System
SLIP	Serial Line Protocol
VDA	Virtual Disk Agent
LKM	Loadable Kernel Module