University POLITEHNICA of Bucharest

Automatic Control and Computers Faculty,
Computer Science and Engineering Department

BACHELOR THESIS

Automatic Generation Of Virtual Machines

Scientific Adviser:
As. Drd. Ing. Răzvan Deaconescu

Author:
Alexandru Andrei

Bucharest, 2011

Preface

Abstract

Virtualization is one of the hot topics in today's IT industry and academia. Whether it is used for commercial or educational purposes, it's growing popularity is a testimony to the many advantages if offers.

However, the process of creating and configuring virtual machine is time consuming, as it is done manually by users. Solutions that automatize this process are rare and do not provide a common interface for different virtualization technologies.

In this paper, we present *vmgen*, a new tool which tries to fix these issues, automating and simplyfing the creation and the system and software configuration of virtual machines. It is a tool that aims at saving both time and energy, for technical and non-technical users.

Contents

List of Figures

Chapter 1

Introduction

Virtualization is certainly one of the most interesting topics in today's IT industry and academia. It is becomming more widely used by companies in various fields as it saves money, time and energy[4]. It is also an interesting topic of research and can be extremely useful for education purposes.

To give a short definition, virtualization implies the creation of a virtual version of an operating system[9]. Another way of puting it, it enables the existence of multiple operating systems on a single computer, through the creation of *virtual machines*. A virtual machine creates an *abstraction* of the physical resources into virtual resources.

There are several different types of virtualization, depending on the level of *abstraction*[6]. Some of these are:

- *full virtualization* - complete simulation of the hardware; operating systems run unmodified
- *partial virtualization* - partial simulation; applications may require some modifications
- *paravirtualization* - hardware is not simulated; virtual environments run on the same system as the host, but are isolated

They all have their advantages and disadvantage, some more than others, but each category includes virtualization solutions that are widely used nowadays. In this aspect, it is worth mentioning `VMware`, `Virtual Box`, `Kvm` (full virtualization), `OpenVZ`, `lxc` (operating system-level virtualization) and `Xen` (hypervisor).

Virtualization is an extremely useful tool that can be used in multiple scenarios. Since in enables the existence of multiple operating systems on a single machine, it takes away the necessity of running multiple machines, each with it's own operating systems. Also, because the operating systems are isolated, if one of them crashes, the others will not suffer any damage. This enables a virtual machine to act as a *sandbox*, useful for testing and development purposes.

However useful they are, the process of generating and configuring a virtual machine is a time consuming operation. Installing an operating system, configuring a system and installing a set of applications takes a lot of time as they are operations done by the user manually. This is extremely slow compared to an automated process.

This paper introduces a tool called `vmgen` which tries to solve these issues by entirely automating these operations, thus reducing the total time spent. It's purpose is also to offer a common interface for the creation and generation of virtual machines, using several virtualization technologies. This is not easy to accomplish, since virtualization technologies do not tend to offer the same features as others. Each has it's own advantages and issues, so the task of finding the common aspects is rather challenging.

`Vmgen` also tries to simplify the entire process, thus enabling non-technical users to benefit from the advantages of virtualization without having to go through all the troubles of setting up a new operating system, a process which can get tedious sometimes.

Currently, the tool offers support for both full virtualization - `VMware` - and operating system-level virtualization - `OpenVZ` and `lxc`. It enables the creation and system configuration of Linux and Windows hosts. It also provides software configuration (programs and tools installed) for Windows and some important Linux distributions - Debian, Fedora, Ubuntu.

For the rest of the work, I will describe some of the components and modules of the applications, detail certain aspects of the implementation and offer some suggestions for the tool's future development.

Chapter 2

Configuration File

The `vmgen` application requires the use of a configuration file at startup. This must be provided by the user and must contain the settings for the new virtual machine. The only setting not found in this file is the virtualizaton technology - that detail will be specified directly to the central module. The configuration file is the basis for all subsequent configurations.

The other option considered for providing the settings was an interactive prompt in which the user would specify the settings guided by the application. However, the use of a configuration file proved to be much more faster to use, although a little bit more difficult. Also, if a graphical interface would be created for the application, it would be much easier to integrate it and would not imply any modifications to the existing source code.

The following image shows the configuration file's role in the application's flow.

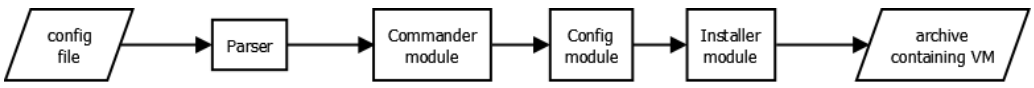

Figure 2.1: Application flow

2.1 Format

In order to make it easier to create, edit or interpret in later stages, the configuration file uses the INI file format to organize the data [1]. Another option considered was the use of the XML file format. This format uses the same basic representation (key, value pairs) and supports many of the same features as the INI format. Also, Python standard library supports both file formats (XML parsers and INI parsers). However, the XML syntax is much more complicated and dense. A configuration file in the XML file format can be 2−3 times larger than the equivalent in the INI file format. For these reasons, the INI format was preferred.

In the INI file format, each basic setting has a name and a value.

```
1 ip=192.168.1.2
2 gateway=192.168.1.0
3 nameserver=192.168.1.1
```

Listing 2.1: Basic configuration setting

Related settings are grouped in sections. The major sections are:

- hardware
- network
- users
- config
- devel
- services
- gui

These sections may contain nested sections - subsections, forming a hierarchy. A new subsection is marked by an extra pair of sorrounding brackets ("[...]"). Each major section will be presented in detail in section 2.2. For more details about

[1] http://en.wikipedia.org/wiki/INI_file

the content of the configuration file, please consult the project's wiki pages [1].

```
1 [hardware]
2         ...
3         [[hdds]]
4                 ...
5                 [[[hdd0]]]
6                         ...
7                         [[[[partitions]]]]
8                                 ...
```

Listing 2.2: Nested sections

The syntax is flexible, allowing the following features:

- empty values
- white spaces
- empty lines
- list values
- multiple line values
- comments.

. However, options outside the major sections will not be taken into account. Also, the user must take into account the fact that one section(major section or subsection) lasts until a token marking another section is found ("["). Identation is not required, but is highly recommended as it makes the configuration file much easier to interpret visually.

2.2 Supported Features

As mentioned earlier, related settings are grouped in *major* sections. The *hardware* section contains the settings used for the actual creation of the virtual

[1] http://ixlabs.cs.pub.ro/redmine/projects/vmgen/wiki/SupportedOptions

machine. It must contain a virtual machine identifier (name for VMware or lxc/numeric ID for OpenVZ) and an operating system identifier. Also, it can specify the amount of memory available to the virtual machine or the number of CPUs.

Furthermore, this section specifies the disk structure, by setting the hard drives, their parameters and the partitions. The cd drives can also be included in this section, as well as the network interfaces.

The rest of the sections include options used in the subsequent system configuration (networking, users) or software configuration (applications).

The `network` section contains the network parameters for each interface: IP address, gateway (on Windows), network mask or global parameteres: nameservers, gateway (on Linux). This section also includes firewall settings, by including a subsection that specifies the open ports and allowed applications. Also, for Linux, the section can include a subsection where the user can submit custom firewall rules (iptables). Besides these system settings, the section also includes a set of programs which can be categorised as networking tools (nmap, tcpdump, etc). These can be installed in the virtual machine.

The `users` section contains a list of custom groups and a list of new users with the corresponding parameters: password, group(s), home directory (Linux).

The `config` section contains some additional options for system configuration: hostname, root password, additional repositories (some Linux distributions).

The remaining three sections contain lists of programs which are to be installed in the virtual machine. These programs are grouped in three major categories:

- `devel` - development tools
- `services` - services to be configured
- `gui` - tools with graphical interfaces

The `development tools` category includes:

- text editors and IDEs - Vim, Emacs, Eclipse
- programming languages - Python, Php, Tcl
- debugging tools - valgrind
- support libraries - build-utils, kernel-devel, mpich2, openmp, openjdk

The `services` category includes:

- servers: http, mail, dns, ...
- subversion systems: git, svn, mercurial
- network file systems: NFS, LustreFS
- additional services: ntp, squid proxy, bittorent

The `gui` section includes:

- web browsers: Mozilla Firefox, Google chrome
- mail clients: Mozilla Thunderbird
- network analyzers: Wireshark

For more details on the supported configuration file options, please consult the project's wiki pages[1].

2.3 Editing and Parsing

The python standard library includes a module `ConfigParser` that can be used to parse configuration files that have a structure similar to INI files. However, the major disadvantage of this solution is that it does not support nested sections.

The structure of the configuration file without the use of nested sections can get extremely complicated. As a result, the application uses for parsing the `ConfigObj` library which also adds some useful features, while maintaing the simplicity of use[3].

The parsing module `vmgParser` just passes the name of the configuration file to `ConfigObj` which builds a data structure recursively from the content of the configuration file. Because of the way the options are specified (list of pairs key=value), an associative array must be used. This type of data structure is known in `Python` as a dictionary and contains a list of values indexed by keys. For this application, both the keys and the values will be represented as *strings*. The object that represents the entire data strucure is, therefore, a dictionary which contains a list of dictionaries indexed by the name of the major sections.

[1] http://ixlabs.cs.pub.ro/redmine/projects/vmgen/wiki/SupportedOptions

Each nested dictionary will also contain a list of dictionaries corresponding to it's nested sections.

To simplify the access to specific data in this structure, the application uses a wrapper class - `vmgSection` - which stores the information in a section (the corresponding dictionary) and provides an interface for handling basic operations.

The data structure built from the information in the configuration file is then serialized to be passed to subsequent modules. This means that the structure is converted to a format and stored in a file that can be later used to restore the original structure. This enables multiple modules to use the data structure without requiring parsing and building the structure each time. Also, the Python standard library supports object serialization through the use of the `pickle` library. The operations of creating the object serialization and writing it to a file (a process called *dumping*), as well as restoring it is extremely fast and easy to use.

```
1  (ivmgStruct
2  vmgStruct
3  p0
4  (dp1
5  S'data'
6  p2
7  (dp3
8  S'network'
9  p4
10 (ivmgStruct
11 vmgSection
12 p5
13 (dp6
14 ...
```

Listing 2.3: Serialized representation of a configuration file

After the parsing stage, the *dump file* is used by the *commander* modules to restore the content of the configuration file and use it for the creation and subsequent configuration of the virtual machines.

```
1 class CommanderBase:
2         def __init__(self, dumpFile):
3                 self.loadStruct(dumpFile)
4
5         def loadStruct(self, dumpFile):
6                 with open(dumpFile, 'rb') as f:
7                         self.data = pickle.load(f)
```

Listing 2.4: Restoring the data structure

Another important aspect is the editing of the configuration file. The structure is designed as to allow easy interpretation and editing of complex configurations. As a result, the configuration file is fairly easy to modify manually, by the user. However, it is rather difficult to make any modifications from another application. Doing this would require a re-parsing and familiarity with the data structure.

The `ConfigObj` library provides an interface for configuration file edition. However, `vmgen` modules wrap the library's features, masking it outside the application. For this reason, the editing feature of the library can not be directly used.

To solve this, another module was created in order to wrap around the editing feature of the library. It is easy to use by users, or by application, having an intuitive usage (it is similar to the use of *sysctl* command in Linux systems).

```
1         vmgCtl.py network.eths.eth0.address=192.168.1.23
```

Listing 2.5: Editing the configuration file

The above command accesses the *network* section and the network interfaces subsection to set the IP address of interface eth0.

Similar commands can be used to alter or add any value in the configuration file, in any section or subsection.

One important notice is that the parsing modules of the application *do not check* the validity of the data inside the configuration file. This means that user can set any options to invalid values without this being detected in the parsing stage. This kind of checking is rather complex for this stage and the only use would

be to fix user errors. Instead, the application will assume the data is correct. Of course, the actual error will be detected later on, by the appropriate module (commander, configuration module or installer).

Also, the addition of unrelated settings in the configuration file will not interfere with the operations of the other modules. For example, the user can add an option, specifying that he wishes a certain unsupported application installed.

```
1 [gui]
2         ...
3         visual_studio=1
4         ...
```

Listing 2.6: Editing the configuration file

However, this option, although present in the data structure used by all the modules, will be ignored by the corresponding module (the installer module).

The conclusion is that the configuration file's syntax and content is highly flexible, but the options actually used by the modules of the application are restricted to a specific subset.

For more details about the structure and content of configuration files, please consult the various examples included in the annex B.

Chapter 3

OpenVZ Commander Module

The `vmgen` application wishes to provide support for container virtualization solutions and operating system layer virtualization. `OpenVZ` is currently an open-source container virtualization technology built for Linux. The virtual environment (known as *container*) is essentialy a complete Linux file system tree mounted in the host file system and isolated using `chroot` command. This operation allows processes to run using a different root directory, which means it can not access files outside of specified directory tree. As a result, each container can act like a stand-alone system, with it's own:

- users
- IP addresses
- files
- applications

Container virtualization implies the creation of the directory tree and configuration file. A large number of precreated `templates` are available on the OpenVZ download site[1] which can be easily imported using the OpenVZ API (`vzctl`). In regards to the aspect of creating new containers, `OpenVZ` is fairly easy to use, by comparison with the other important container virtualization technology, `lxc` (linux containers). However, `lxc` has the advantage of having native support in the Linux kernel. Instead, `OpenVZ` requires a new kernel, compiled to offer support for container virtualization. Some Linux distributions, like Debian or Fe-

[1] http://download.openvz.org/template/precreated/

dora, make this operation easier for the user, by providing the compiled kernels in their repositories. The user needs only to install the new kernel and the API tools.

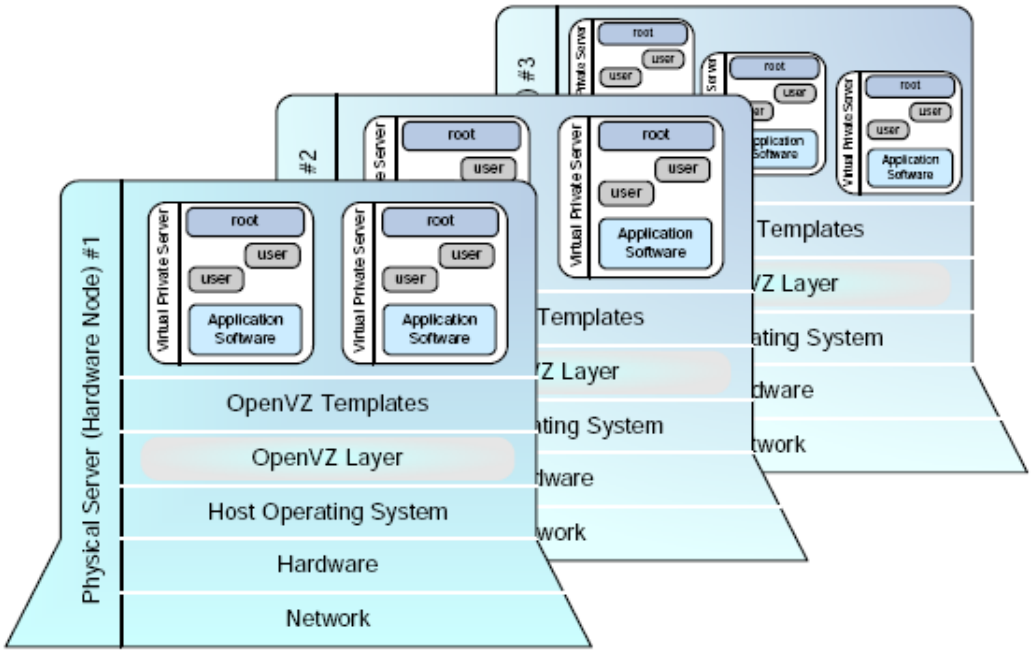

Figure 3.1: OpenVZ Arhitecture

However, most of the distributions (most notably, Ubuntu) do not offer such support and the user must manually download the kernel, compile it with container virtualization support and install it. This operation can be fairly complicated. Considering these aspects, the system used as a host for the containers, during testing was a `Fedora 14`.

3.1 General Design

This section describes the basic implementation of the `OpenVZ commander module` and its connection with the other modules of the application. A commander module is the application component that creates the virtual machine and it's hardware configuration and instantiates the proper modules to handle further configurations (networking, users, applications, etc). This module is written in `Python` but also makes use of `bash scripts` for certain operations.

A common interface (`vmgCommanderBase`) is used for all the commander modules and contains the basic operations that need to be implemented by any commander. This application architecture facilitates the addition of new commander modules for new virtualization technologies without having to alter the existing modules.

```
1  class CommanderBase:
2          def startVM(self):
3                  pass
4          def shutdownVM(self):
5                  pass
6          def connectToVM(self):
7                  pass
8          def disconnectFromVM(self):
9                  pass
10         def setupHardware(self):
11                 pass
12         def setupPartitions(self):
13                 pass
14         def setupOperatingSystem(self):
15                 pass
16         def setupServices(self):
17                 pass
18         def setupDeveloperTools(self):
19                 pass
20         def setupGuiTools(self):
21                 pass
22         def getConfigInstance(self):
23                 return None
24         def getInstallerInstance(self):
25                 return None
```

Listing 3.1: Commander module interface

As seen above, the commander must provide methods for starting or shuting down the virtual machine, for hardware setup (including partitions and operating

system, if necessary) and for applications setup (services, developer tools, gui tools).

Also, the commander is responsible for instantiating an `installer module` that will install the programs requested by the user and a `config module` that will perform additional configuration to the virtual machine (networking, users).

The `OpenVZ commander` overrides only a subset of these methods. For example, the `setupOperatingSystem` method does not have a meaning in the context of container virtualization. The following sections will describe in detail how the `OpenVZ module` implements each of the methods.

3.1.1 Module Control Flow

The `OpenVZ commander module` is responsible for creating and configuring a new `OpenVZ container` using the settings specified in the configuration file. As shown in the previous section, the configuration file is parsed, and the subsequent data structure is serialized and dumped in a file. Also, the central module vmgen is responsible for instantiating the corresponding commander module (OpenVZ, lxc, VMware).

```
1  class CommanderBase:
2          def __init__(self, dumpFile):
3                  self.loadStruct(dumpFile)
4
5          def loadStruct(self, dumpFile):
6                  with open(dumpFile, 'rb') as f:
7                          self.data = pickle.load(f)
8          ...
9
10 class CommanderOpenvz(CommanderBase):
11         def __init__(self, dumpFile):
12                 CommanderBase.__init__(self, dumpFile)
13         ...
```

Listing 3.2: Retrieve settings from the dump file

The first thing the commander does is to restore the data structure containing all the user-specified setting from the dump file and store it locally for further user. After retrieving the settings data structure, the commander will perform the basic setup in the following order:

```
1         self.setupHardware()
2         self.setupPartitions()
3         self.setupOperatingSystem()
4
5         self.startVM()
6         self.connectToVM()
7
8         self.config = self.getConfigInstance()
9         self.config.setupConfig()
10        self.root_passwd = self.config.getNewRootPasswd()
11
12        self.installer = self.getInstallerInstance()
13        self.setupServices()
14        self.setupDeveloperTools()
15        self.setupGuiTools()
16
17        self.disconnectFromVM()
18
19        self.shutdownVM()
```

Listing 3.3: Operations performed by the commander module

The commander first creates the virtual machine and the `setup hardware` and then starts the virtual machine. Subsequent operations require the vritual machine running. Next, it will instantiate a `configuration module` and an `installer module` that will complete the configuration. After these operations are completed, the vritual machine is ready and is packed in order to be transferred to the user. The directory will be packed in an archive file and this will also be packed together with the container configuration file in the final archive which will be passed to the user.

3.1.2 OpenVZ API

The `OpenVZ technology` provides an extensive tool for container management: `vzctl`[5]. This facilitates the creation and removal, startup and shutdown, and the setting of a wide range of parameters for the container[7]. However, the `OpenVZ commander` uses only a subset of the options available, since many of them are used for extreme low-level configurations. For example, the option to set the maximum size of the TCP receive buffer (–tcprcvbuf) is highly unlikely to appear in the user requests. Also, the inclusion of these options would have made the creation of a common interface for all the virtualization solutions extremely difficult.

The subset of options used by the commander is shown below:

- vzctl `create` CTID –ostemplate name : creates a new container with the specified id, from a precreated template
- vzctl start | stop | destroy CTID : starts, stops or destroys the container
- vzctl set CTID –cpus num : number of CPUs available in the container
- vzctl set CTID –privvmpages value : controls the amount of memory available
- vzctl set CTID –diskspace value : controls the amount of harddisk available
- vzctl set CTID –netif_add ifname : adds a new virtual interface (veth)

Also, the module does not use some useful options that can be set using the `vzctl` API, like IP addresses, nameservers, hostname. These are set by the `configuration module` (configLinux.py) in a later stage, using Linux commands instead. It is much easier to use a configuration module for Windows and Linux to handle all possible virtualization solutions, instead of creating configuration modules for each technology.

3.1.3 Communication Interface

Each commander uses a `communicator module` to handle the communication with the virtual machine instance. A separate communicator is required for

each virtualization solution and must offer a certain interface, defined by the `vmgCommunicatorBase` class.

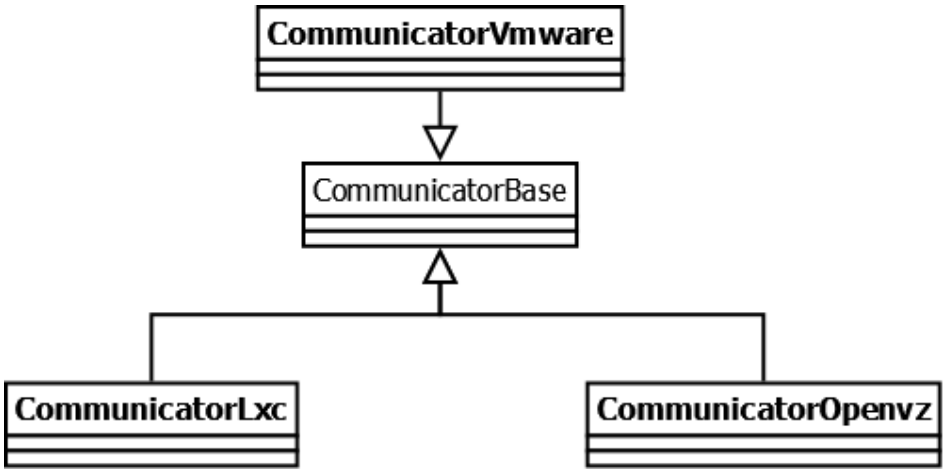

Figure 3.2: Common commander interface

This interface must provide functions for executing commands in the virtual machine (container), for copying and deleting files inside the virtual machine.

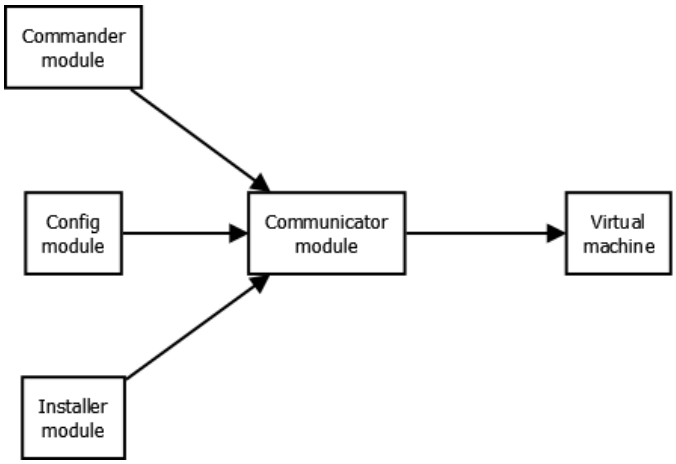

Figure 3.3: Communication with the virtual machine

Each communicator is instantiated with a set of communication parameters specific to each virtualization technology. For `OpenVZ`, these parameters are:

- `vmx`: path to the VMware virtual machine used as a host for the container
- `host`: user and hostname for the VMware virtual machine
- `id`: the id of the container

The communicator will use `ssh` and the `vzctl` API to execute commands in the VMware host and `scp` for file transfers:

```
1 def runCommand(self, cmd):
2         executeCommandSSH("vzctl␣enter␣" + self.id + "␣--
             exec␣" + cmd + ";logout")
3
4 def copyFileToVM(self, localPath, remotePath):
5         executeCommand("scp␣" + key + "␣" + localPath + "
             ␣" + self.host + ":$VZDIR/root/" + self.id + "
             /" + remotePath)
6
7 def deleteFileInGuest(self, remotePath):
8         executeCommandSSH("rm␣-rf␣$VZDIR/root/" + self.id
              + "/" + remotePath)
```

Listing 3.4: Communication with the container

The file transfers actually represent a simple copy operation to the container directory tree which is mounted in a specific location in the host file system. The location where the containter is mounted is given by the `$VZDIR` variable, the `root/` subfolder and then the subfolder corresponding to the container, which has the name of the `container id`.

3.2 Basic Configurations

The settings for the hardware setup are listed in the `hardware` section of the configuration file:

```
1 [hardware]
2         vm_id = 123
3         os = fedora-14-x86
4         num_cpu = 1
5         ram = 1024
6         [[hdds]]
```

```
7                   [[[hdd0]]]
8                   size = 1G
9           [[eths]]
10                  [[[eth0]]]
11                  type = nat
12                  connected = 1
```

Listing 3.5: Hardware section example

For the creation of the container, the module uses a `precreated template` in the form of a gzip archive, containing the directory tree. The name of the template is set using the `os` option. The module will use a `bash script` to make sure the template is present in the OpenVZ cache (/vz/template/cache). If the template is not found, it will try to retrieve it from the OpenVZ download page[1].

```
1  #!/bin/bash
2  ...
3  get_template()
4  {
5          pack=$1.tar.gz
6          # possible download paths
7          precreated="http://download.openvz.org/template/
              precreated/"
8          beta=$precreated"beta/"
9          contrib=$precreated"contrib/"
10         unsupported=$precreated"unsupported/"
11         # wget: set quite mode && output directory
12         flags="-q␣-P␣/vz/template/cache"
13         for link in $precreated $beta $contrib
              $unsupported
14         do
15                 echo "␣*␣try␣$link$pack"
16                 wget $flags $link$pack
17                 if [ $? -eq 0 ]; then
```

[1] http://download.openvz.org/template/precreated

```
18                          exit 0
19                  fi
20          done
21          echo "template not found"
22          exit 1
23  }
24  ...
```

Listing 3.6: Bash script to retrieve template

For a complete view of the bash script used for template retrieval, please consult the annex A.

If the container was succesfully created, the setup continues by starting the container and setting the following options, using the `vzctl` API mentioned earlier:

- number of CPUs available inside the container
- amount of memory available
- amount of harddisk available

The most interesting aspect of the hardware setup is the network interfaces setup. OpenVZ provides `veth`(Virtual Ethernet)[8] and `venet`(Virtual Network) devices for networking. The differences between the two are shown below [1]:

Feature	veth	venet
MAC address	Yes	No
Broadcasts inside CT	Yes	No
Traffic sniffing	Yes	No
Network security	Low[2]	High[3]
Can be used in bridges	Yes	No
Performance	Fast	Fastest

The module can setup up networking using any of these two, according to the option `type` specified for each interface. `Venet` interfaces only require an IP address. However, `veth` interfaces needs additional configuration. The commander uses `netif_add` options in the `vzctl` API to create an interface in

[1] http://wiki.openvz.org/Differences_between_venet_and_veth
[2] Independent of host. Each CT must setup its own separate network security.
[3] Controlled by host.

the container, an interface in the host and, most importantly, a bridge between them. Furthermore, `IP forwarding` and `proxy_arp` must be enabled for both interfaces, `IP addresses` must be assigned to each interface (must be in the same network) and a new default route added in the container. In order to make these persistent, a script including these operations is set to execute at container start-up[1].

3.3 Additional Configurations

Besides the hardware setup, other additional configurations are required for the container. These include:

- `Users setup`: groups, users, passwords
- `Network setup`: IP addresses, nameservers, hostname
- `Firewall setup`: open ports, allowed programs, additonal firewall rules
- `Applications setup`: install programs

These configurations are not done explicitly by the `OpenVZ commander` module, but by additional components. The reason is that these additional configurations are not related to the virtualization solution. For example, network setup is similar on Linux virtual machines, even if they are OpenVZ container, lxc containers or VMware Linux guests. Also, the application installing does not depend on the virtualization technology, but on the operating system installed:

- Windows virtual machines use `InstallerWindows`
- Linux virtual machines use `InstallerApt`/`InstallerYum` depending on the distribution (Debian/Fedora/Ubuntu)

However, the commander is responsible for instantiating these additional components with the correct parameters (configuration file settings, list of programs to be installed, communication module).

The following subsection will describe the `configuration module` for Linux. For the `Linux installer modules`, please consult chapter 4.

[1] http://www.linuxweblog.com/blogs/sandip/20080814/bridge-networking-on-openvz-containers-using-veth-devices

Linux Configuration Module

As with all the modules of the application, the Linux configuration module is required to implement a specific interface.

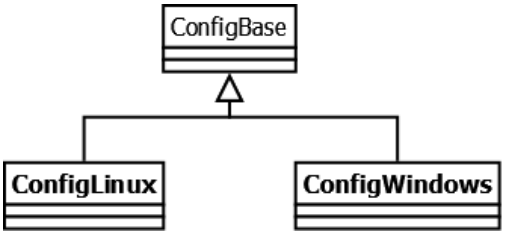

Figure 3.4: Common configuration interface

The `vmgConfigBase` class specifies that each configuration module must provide methods for system configuration, groups and user setup, network and firewall setup.

```
1         def setupConfig(self):
2                 self.setupSystem()
3                 self.setupGroups()
4                 self.setupUsers()
5                 self.setupNetwork()
6                 self.setupFirewall()
7
8                 self.applySettings()
9
10        def setupSystem(self):
11                pass
12        def setupGroups(self):
13                pass
14        def setupUsers(self):
15                pass
16        def setupNetwork(self):
17                pass
18        def setupFirewall(self):
19                pass
```

```
20          def applySettings(self):
21                  pass
22          def getNewRootPasswd():
23                  return None
```

Listing 3.7: Configuration module interface

The methods will be automatically called by the `setupConfig` in the correct order (for example, groups before users). The modifications made by all these methods will be applied to the virtual machine by the `applySettings` method. Furthermore, each configuration module must provide the new access credentials (`getNewRootPasswd`) for the virtual machine, in order to be used by the subsequent modules (the installer modules).

In detail, the Linux configuration module uses Linux specific commands to handle the necessary configurations. In some cases, multiple commands can be used to achieve the required result. However, the commands used were the ones who have a *permanent* or *non-interactiv behavior*. For example, multiple commands can be used to set the `hostname`:

```
1 sysctl kernel.hostname=new_hostname
2 hostname new_hostname
3 edit /etc/sysconfig/network and modify hostname value
```

Listing 3.8: Set hostname on Linux systems

From the listed commands, the second does not modify the system permanently, while the third is harder to use non-interactively. So, the first command is preferable to be used.

For group and user setup, the `groupadd` and `useradd` commands are used, as they are both permanent and can be used non-interactively. For modifying the user passwor, the `passwd` command can be used, using a standard input redirect and the `echo` command.

```
1          echo new_pwd | passwd --stdin user
```

Listing 3.9: Set user password non-interactively

In regards to the network setup, the `ip` command (iproute2 package) can be used for most of the settings (*ip addr*, *ip link*). However, in some cases file editing is also used - for example, nameserver setup requires the editing of file */etc/resolv.conf*.

For firewall setup, open ports and firewall rules can be both configured by using `iptables`.

Each of the previous configurations are added to a script file, which in the end will contain all the requested settings. To apply these settings, the script file is copied and executed inside the virtual machine.

This operation completes the system configuration and the virtual machine can subsequently be used for software configuration - application install.

3.4 Testing Results

The commander module has been thoroughly tested to generate various containers for several Linux distributions. Since the time of creating a container is mostly spent on generating the directory tree, and OpenVZ offers the set of precreated templates for this, the time results vary around the 1-2 minutes margin.

The configuration module has been tested both as a stand-alone component and in connection to the commander module with good results. The settings are succesfuly applied to the virtual machine in under 1 minute.

Chapter 4

Linux Installers

The final stage of the machine configuration is the the installation of programs inside the virtual instance. These are a set of applications that are commonly used on these system (for example, development tools) and are likely to be requested by the users. The set was completed using our previous experience in using virtual machines for development purposes. The list of programs is defined in the configuration file:

```
1  [devel]
2          vim = 1
3          valgrind = 1
4          python = 1
5          openjdk = 1
6  [services]
7          mail_server = 1
8          dns_server = 1
9          sshd = 1
10         git = 1
11         mercurial = 1
```

Listing 4.1: User defined list of pograms

The structure of the configuration file is flexible and the user can add various new programs, but `vmgen` will only consider a specific set of programs. Any other values added to the configuration file will be ignored.

The list of programs available for Linux is extensive and includes:

- `Networking tools`: nmap, traceroute, netcat, etc
- `Development tools`: vim, emacs, valgrind, etc
- `Services`: httpd, dns server, sshd, git, etc
- `GUI tools`: web browsers, wireshark, etc

For the complete list of programs please consult the project's wiki page [1].

The application provides installers for the distributions using `apt-get` or `yum` package manager.

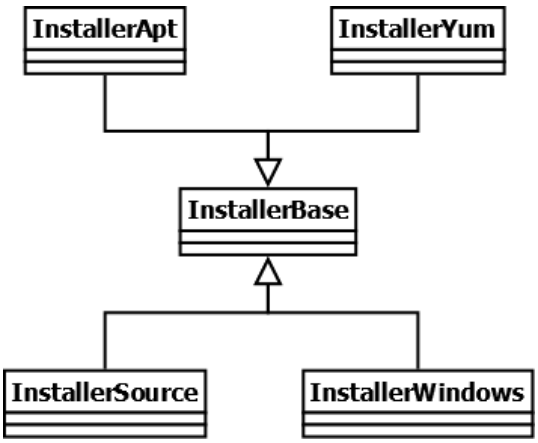

Figure 4.1: Installer architecture

Both use a common interface, extending the `InstallerBase` class and overriding it's `install` method.

```
1 class InstallerBase:
2         def __init__(self, communicator):
3                 pass
4
5         def install(self, program):
6                 pass
```

Listing 4.2: Common installer interface

[1] http://ixlabs.cs.pub.ro/redmine/projects/vmgen/wiki/SupportedOptions

The architecture is flexible, allowing the addition of new installers for other package managers or a source installer that will compile and install the programs from their source code.

Each installer uses a communicator passed by the commander module. The communicator is used by the installer to execute commands and copy files inside the virtual machine. This means that the installer does not need to distinguish between the virtualization solutions. The transparency is ensured by the communicators.

4.1 Apt-get Installer

The module `InstallerApt` is designed to install programs for Linux distributions that use the `apt-get` package manager. It uses the `Debian/Ubuntu repository` to retrieve packages for each program and install it accordingly. Each program is identified in the repository by the name of the package. This means that the installer will have to keep a data structure that maps each program to a certain package name. This data structure is stored as a `Python dictionary` inside the `InstallerApt` module.

```
1  packages = {
2          "mozilla-firefox" : {
3                  "package" : "iceweasel␣firefox" },
4          "mozilla-thunderbird" : {
5                  "package" : "icedove␣thunderbird" },
6          "pidgin" : {
7                  "package" : "pidgin" },
8          "wireshark" : {
9                  "package" : "wireshark" },
10         ...
11 }
```

Listing 4.3: Apt package name mapping

As seen above, the package name can have multiple values for a certain program, as some packages change their name in time or differ between `Debian` and

Ubuntu. For example, the Mozilla Firefox browser package is known as iceweasel on Debian and firefox on Ubuntu.

Some programs (for example, eclipse or mpich2) that can be found in the Ubuntu repository don't have a correspondent in the Debian repository. Also, the repository version of eclipse is old and the correct way to install it would be to do it manually instead of using the repository.

As a result of these problems, a small set of programs are not available on some systems.

Regarding the module control flow, the installer will first check the list of programs received from the commander module and will only keep those that can be found in the packages mapping. Any other programs are not supported and considered invalid. Each programs will then be installed by executing the apt-get command inside the virtual machine using the communicator as a mediator.

```
1 install_cmd = "␣/usr/bin/apt-get␣install␣-y␣-q␣"
2 class InstallerApt:
3         def install(self, programs):
4         ...
5         # Retrieve only the list of valid programs
6         packs = [packages[p] for p in programs if p in
             packages]
7         if packs:
8                 [self.comm.runCommand(install_cmd + p['
                    package']) for p in packs]
```

Listing 4.4: Programs install

The apt-get command is run in quiet mode (" -q ") since the output is not relevant. Also, the command will run non-interactively (" -y ") and will assume yes as answer to all prompts[1].

Another problem encountered in the testing stages on some systems is the presence of an option in the apt-get sources file that enables the command to search the requested package on the installing CD-ROM. However, if this is not found, the command will block waiting for user actions, thus blocking the entire module.

To fix these, a `sed` command will disable any such options by commenting the correspoding lines in the sources file.

```
1 self.comm.runCommand("sed␣-i␣'/cdrom/s/^/#␣/'␣/etc/apt/
    sources.list")
```

Listing 4.5: Disable cdrom option

Testing results

This installer has been tested more as a stand-alone module, independently from the central module. This has the advantage of not wasting time on each test with the virtual machine creation and system configuration.

The tests included various sets of applications, including both small and large packages. The time result can vary significantly as a result, ranging from several seconds, for sets of small packages, to several minute for lage packages.

To install all the applications supported, the application required up to 10 minutes. This figure can also vary, depending on the Internet connectivity and the host system. The time can not be improved by the application as it is only related to package manager operations, the retrieval of packages and the actual install.

The module has also been tested, with good results, in the context of the entire application, to check the connectivity between this module and the corresponding commander module.

The module has been tested on Debian 6 and Ubuntu 10.10 distributions. For some older distributions, the installer does not guarantee the succesful install of all the programs specified. This can easily be fixed with an extended testing stage.

4.2 Yum Installer

The module `InstallerYum` is designed to install programs for Fedora distributions that use the `yum` package manager. This is similar to the `apt-get` repository mentioned in the previous section 4.1. Each program is identified by

a package name. In addition, a set of programs (for example, development tools) can be installed together as a group. In this case, the identifier is the name of the group[2].

Considering the similarities, the structure of the `Yum installer` will basically be the same, mapping each program to a package name. However, each install entry will also require a `type` value to distinguish the groups of packages from simple packages. Another option `local` is supported to enable the install of programs from local `rpm` files, but still using the `yum` package manager.

```
1  packages = {
2          "mozilla-firefox" : {
3                  "type" : "simple",
4                  "package" : "firefox" },
5          "mozilla-thunderbird" : {
6                  "type" : "simple",
7                  "package" : "thunderbird" },
8          "build-utils" : {
9                  "type" : "group",
10                 "package" : "\"Development␣Tools\"␣\"
                      Development␣Libraries\"" },
11         ...
12 }
```

Listing 4.6: Yum package name mapping

Some programs are not present by default in the `yum repository`. These programs require the addition of a specific `repo` file to the `yum` configuration files (the default location is `/etc/yum/repos.d`). The path to such `repo` files must be provided, as the installer needs to copy it inside the virtual machine. These programs can be identified in the packages dictionary by their type, which is set to `repo`.

Almost the entire set of available programs for the Linux guests is available in the `yum` repository, the most important exception being the `Chrome` web browser. The best way to install it is to use `Google's own yum repository`, using a `repo` file. Also, `dynamips` is not supported by the yum repository, but it can be installed by using a `rpm` file.

The control flow of this installer is similar to the `Apt installer`. This means that the module will first check the programs requested by the user and keep only the ones that are supported (the program name is a key in the `packages` dictionary. Afterwords, each program will be installed according to it's type.

```
1 local_cmd = "␣yum␣-y␣-d␣0␣-e␣0␣localinstall␣--nogpgcheck␣
    "
2 simple_cmd = "␣yum␣-y␣-d␣0␣-e␣0␣install␣"
3 group_cmd = "␣yum␣-y␣-d␣0␣-e␣0␣groupinstall␣"
```

Listing 4.7: Install commands

The `yum` command is run non-iteractively (" -y") and assumes `yes` to all prompts. Also, the output of the command is irrelevant and is therefore suppressed (the debug level and error level are set to 0).

```
1  if p['type'] == 'simple':
2          self.comm.runCommand(simple_cmd + p['package'])
3  if p['type'] == 'group':
4          self.comm.runCommand(group_cmd + p['package'])
5  if p['type'] == 'repo':
6          # copy repo file in container
7          self.comm.copyFileToVM(p['repo'], p['package'] +
              ".repo")
8          # install package
9          self.comm.runCommand(simple_cmd + p['package'])
10 if p['type'] == 'local':
11         # copy rpm to root
12         self.comm.copyFileToVM(p['rpm'], ".")
13         # execute
14         self.comm.runCommand(local_cmd + p['package'])
15         # remove rpm
16         self.comm.runCommand("rm␣-rf␣*.rpm")
```

Listing 4.8: Install commands

The install command will differ for simple packages, group packages and local

packages. In addition, the local installation requires the `rpm` file to be copied inside the virtual machine, before running the `yum` command with a specific flag. Also, the `repo` type programs need the configuration file copied to the guest, but are installed as simple packages afterwords.

Testing results

Similar to the `Apt` installer, this component has also been tested more from a stand-alone perspective.

The tests included various sets of applications, including both small and large packages. To install all the applications supported, the application required up to 12 minutes. This figure can vary, depending on the Internet connectivity and the host system.

The connectivity with the commander module has also been tested, by running the entire application with all the modules included.

The module has been tested on Fedora 14 distribution. However, tests for older distributions are required in order to fix any problems that may arise because of the changes in the repository.

Chapter 5

Conclusion

Despite the limited amount of time, we managed to provide most of the features we intended for the application. The tool succesfully generates and configures virtual machines for several important virtualization technologies. It is rather easy to use and logs details about each operation being performed.

However, there is always room for improvement, so there are a few aspects that can be improved and some features that can be added.

In order to improve the usability aspect, the application definitely would benefit from the use of a *graphical user interface* which would make it easier for the user to generate the configuration file.

Also, the application requires a public framework which would enable users to access it, test it and provide valuable feedback.

Regarding the software configuration, some programs are not supported on some systems, due to repository issues. Also, a source installer module would greatly benefit vmgen , as it would allow it to install programs on any Linux distribution.

One of the best features of vmgen is the application architecture which makes it easy to extend. This means that new commander modules can be added fairly easy, in order to provide support for other technologies, like `Virtual Box` or `Kvm`. Also, new Linux installer modules can be added to offer support for other package managers, like Gentoo's emerge.

Another aspect that can be improved is the complexity of the system configuration. The application manages to support a large set of settings, which we

felt were more likely to be requested by users. However, this is some room for expansions in this area, too. This can also be applied to the set of supported programs which can easily be extended.

Another nice feature would be the use of templates for the virtual machine configurations. This way, users that intend to use the virtual machine for development purposes could use a *Development* template to configure a virtual machine with text editors, system libraries and other development tools installed. Users that are going to use the virtual machine for networking purposes could use a *Networking* template to configure the system with a http server, nmap, wireshark or other networking tools. This kind of options will certainly speed up the tool, since the generation of the configuration file can be rather time consuming.

In retrospect, the work on this project has been both challenging and interesting. The application is a useful tool for various types of users and can be considered a good starting point for future development.

Appendix A

Template retrieval script

```
1  #!/bin/bash
2
3  check_cache()
4  {
5          pack=$1.tar.gz
6          cache_path="/vz/template/cache/"
7
8          # check if the openvz cache already contains the
               template
9          if [ -e $cache_path$pack ]; then
10                 exit 0
11         fi
12 }
13
14 get_template()
15 {
16         pack=$1.tar.gz
17
18         # possible download paths
19         precreated="http://download.openvz.org/template/
               precreated/"
20         beta=$precreated"beta/"
21         contrib=$precreated"contrib/"
22         unsupported=$precreated"unsupported/"
```

```
23
24          # wget: set quite mode && output directory
25          flags="-q␣-P␣/vz/template/cache"
26
27          for link in $precreated $beta $contrib
               $unsupported
28          do
29                  echo "␣*␣try␣$link$pack"
30                  wget $flags $link$pack
31                  if [ $? -eq 0 ]; then
32                          exit 0
33                  fi
34          done
35          echo "template␣not␣found"
36          exit 1
37  }
38
39  usage()
40  {
41          cat <<EOF
42  $1 -h|--help -t|--template=<name>
43  EOF
44          return 0
45  }
46
47  options=$(getopt -o ht:n -l help,template:,name -- "$@")
48  if [ $? -ne 0 ]; then
49      usage $(basename $0)
50      exit 1
51  fi
52  eval set -- "$options"
53
54  while true
55  do
56      case "$1" in
57          -h|--help)      usage $0 && exit 0;;
```

```
58          -t|--template)  check_cache $2;  get_template $2;
                 break;;
59          --)             usage $0 && exit 0 ;;
60          *)              break ;;
61      esac
62  done
```

Listing A.1: The source code of the script that retrieves new templates

Appendix B

Configuration file samples

B.1 OpenVZ configuration file sample

```
1  [hardware]
2          vm_id = 123
3          os = fedora-14-x86
4          num_cpu = 1
5          ram = 1024
6          [[hdds]]
7                  [[[hdd0]]]
8                  size = 1G
9          [[eths]]
10                 [[[eth0]]]
11                 hw_addr = 01:00:de:ad:be:ef
12                 type = venet
13
14 [network]
15         gateway = 192.168.1.254
16         netcat = 1
17         tcpdump = 1
18         [[eths]]
19                 [[[eth0]]]
20                         address = 192.168.1.2
21                         network = 255.255.255.0
```

```
22          [[nameservers]]
23                  nameserver0 = 192.168.1.1
24
25  [users]
26          [[groups]]
27                  [[[group0]]]
28                          name = julius
29          [[users]]
30                  [[user0]]
31                          name = caesar
32                          passwd = alesia
33                          group = julius
34                          home_dir = /home/caesar
35
36  [config]
37          hostname = HAL-9000
38          root_passwd = student
39  [devel]
40          vim = 1
41          emacs = 0
42          build-utils = 1
43          valgrind = 1
44          openmp = 1
45  [services]
46          httpd = 0
47          sshd = 1
48          git = 1
49          bittorrent = 1
50  [gui]
51          firefox = 0
52          chrome = 1
53          wireshark = 1
```

Listing B.1: OpenVZ configuration file sample

B.2 Windows XP configuration file sample

```
[hardware]
        vm_id = TestMachine
        os = winxppro
        num_cpu = 2
        ram = 512
        [[hdds]]
                [[[hdd0]]]
                size = 8GB
                type = ide
                scsi_index = 0
                pos = 0:0
                name = hdd.vmdk
                [[[[partitions]]]]
                        [[[[[partition0]]]]]
                        type = primary
                        fs = ntfs
                        size = 5000

                        [[[[[partition1]]]]]
                        type = extended
                        size = 3072

                        [[[[[partition2]]]]]
                        type = logical
                        fs = ntfs
                        size = 2048

                        [[[[[partition3]]]]]
                        type = logical
                        fs = ntfs
                        size = 1024

        [[eths]]
                [[[eth0]]]
```

```
35                          hw_addr = 01:00:de:ad:be:ef
36                          type = nat
37                          connected = 1
38                  [[[eth1]]]
39                          type = nat
40                          connected = 1
41
42  [network]
43          ip_routing = 1
44          nat = 1
45          [[eths]]
46                  [[[eth0]]]
47                          type = static
48                          address = 192.168.1.2
49                          network = 255.255.255.0
50                          gateway = 192.168.1.1
51                          dns = 192.168.1.254
52                  [[[eth1]]]
53                          type = static
54                          address = 10.0.0.2
55                          network = 255.255.255.0
56                          gateway = 10.0.0.1
57                          dns = 10.0.0.254
58
59          [[open_ports]]
60                  [[[port0]]]
61                          proto = tcp
62                          port = 22
63                          description = ssh
64
65                  [[[port1]]]
66                          proto = all
67                          port = 65000
68                          description = myport
69
70  [users]
```

```
71          [[groups]]
72                  [[[group0]]]
73                          name = julius
74                  [[[group1]]]
75                          name = vmg
76          [[users]]
77                  [[[user0]]]
78                          name = caesar
79                          passwd = alesia
80                          groups = julius
81                          home_dir = /home/caesar
82                  [[[user1]]]
83                          name = vv
84                          passwd = P4ssw0rd
85                          groups = vmg
86
87  [config]
88          root_passwd = pass2
89          hostname = xp-gen
90
91  [devel]
92          vim = 0
93          emacs = 0
94          eclipse = 1
95          python = 1
96          jdk = 0
97
98  [services]
99          ftp_server = 0
100
101 [gui]
102         mail_client = mozilla-thunderbird
103         wireshark = 0
```

Listing B.2: Windows XP configuration file sample

Bibliography

[1] apt-get(8) - linux man page. http://linux.die.net/man/8/apt-get, 2011. [Online; accessed 05-July-2011].

[2] Guides - yum - trac. http://yum.baseurl.org/wiki/Guides, 2011. [Online; accessed 05-July-2011].

[3] Michael Foord and Nicola Larosa. Reading and writing config files. http://www.voidspace.org.uk/python/configobj.html, 2011. [Online; accessed 05-July-2011].

[4] Kenneth Hess and Amy Newman. *Practical Virtualization Solutions: Virtualization from the Trenches*. Prentice Hall, 1st edition, 2009.

[5] Benjamin Poulain. Linux certif - man vzctl(8). http://www.linuxcertif.com/man/8/vzctl/, 2011. [Online; accessed 05-July-2011].

[6] Abraham Silberschatz, Peter Bayer Galvin, and Greg Gagne. *Operating System Concepts*. John Wiley and Sons, 7th edition, 2005.

[7] OpenVZ Wiki. User guide/operations on containers - openvz wiki. http://wiki.openvz.org/User_Guide/Operations_on_Containers, 2011. [Online; accessed 05-July-2011].

[8] OpenVZ Wiki. Virtual ethernet device - openvz wiki. http://wiki.openvz.org/Virtual_Ethernet_device, 2011. [Online; accessed 05-July-2011].

[9] Wikipedia. Virtualization — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Virtualization, 2011. [Online; accessed 05-July-2011].