Automated Administration of Virtual Machines and Networks on the VMware ESX Platform

Master Thesis

Dong Yan

Network and System Administration
Oslo University College

May 23, 2011
Automated Administration of Virtual Machines and Networks on the VMware ESX Platform

Dong Yan

Network and System Administration
Oslo University College

May 23, 2011
Abstract

This thesis considers a novel solution to deploy and administrate virtual machines and networks on the VMware ESX platform. The new solution is able to decrease network deployment complexity and increase automatic administration and coherence. Virtual machines and networks are described by MLN language. A plug-in esx.pl is developed to associate MLN to create and manage virtual machines and networks. Advanced features including network upgrade and migration are also implemented. Creating and administrating two different networks are demonstrated via MLN and plug-in esx.pl. More upgrade and migration cases are analysed as well. After that, data is collected to compare the ordinary and new solution to select a more efficient solution among several scenarios. Furthermore, a user interface is developed to make deployment and administration more convenient.
Acknowledgements

First and foremost, I would like to be grateful to my supervisor Kyrre M. Beg-nnum for the project guidance and technical support. Thank him for providing project devices and VMware vSphere ESX license. Besides, thank him for the paper writing guide.

Secondly, I owe Hårek Haugerud a great debt of gratitude since he creates the Zen and Seine groups to practise presentations and helps me order a new Intel network adapter in time.

Thirdly, I would like to say thanks to Ismail Hassan for providing two physical servers and a network adapter. Next my gratitude is expressed to a system administrator Amir M. Ahmed for supplying me valid IP addresses. And I would like to be grateful to my nice friend Stephan Bjoraa for his patient PHP assistance. In addition I am extremely grateful to Isaac B. Liney who assists me in examining the thesis and correcting my English grammar mistakes.

Lastly my appreciation is expanded to all the lecturers, professors, classmates and friends. Because of their companions and academic enlightenment, my achievement is obtained during the two years master study.

May, 2011
Dong Yan
Contents

1 Introduction ............................................. 1
  1.1 Problem Statement .................................. 2
  1.2 Summary of the Results ........................... 3
  1.3 Thesis Outline ................................... 4

2 Background ............................................. 5
  2.1 VMware Virtualization Technology ............... 5
    2.1.1 VMware Product Family ........................ 5
    2.1.2 VMware Virtualization System Architecture .......... 7
  2.2 VMware Terminology ................................ 9
    2.2.1 Datacenter .................................. 9
    2.2.2 Datastore .................................. 10
    2.2.3 Virtual Machine File System (VMFS) ............... 10
    2.2.4 Virtual Switch ................................ 10
    2.2.5 Virtual Disk Development Kit (VDDK) ............. 11
  2.3 vSphere Command Line Interface (vCLI) ........... 11
    2.3.1 Manage Virtual Machines ...................... 12
    2.3.2 Manage Virtual Switches ...................... 14
    2.3.3 Manage Virtual Disks ........................ 15
    2.3.4 Manage Files ................................ 16
  2.4 Virtual Machine Administration Tools ............. 17
    2.4.1 MLN (Manage Large Network) .................. 17
    2.4.2 MLN Programming Language ................... 17

3 Approach .............................................. 21
  3.1 System Design ................................... 21
    3.1.1 Topology .................................. 21
    3.1.2 Procedure .................................. 22
    3.1.3 Package Control .............................. 23
    3.1.4 Enable VT (Intel Virtualization Technology) .... 24
  3.2 Obtain VMX and VMDK Files ........................ 25
  3.3 Edit VMX File ................................... 25
    3.3.1 Modified Items ............................... 25
    3.3.2 Emptied Items ............................... 26
  3.4 Deploy Virtual Machine only by Template and vCLI .. 26
    3.4.1 Upload VMX and VMDK files .................. 26
    3.4.2 Register and Power on Virtual Machine .......... 27
## CONTENTS

3.4.3 Power off Virtual Machine and Cancel Register ................................. 27
3.4.4 Remove Virtual Machine ................................................................. 28
3.5 Investigate MLN Source Code ................................................................. 28
3.6 Principle of Developing New Plug-in esx.pl ........................................... 29
3.7 Advanced Feature: Network Upgrade ................................................... 30
3.8 ESX Keyword Implemented in MLN Project File ...................................... 30
3.9 Use MLN and Plug-in esx.pl to Administrate VMware ESX Platform ................. 31
  3.9.1 Simple Network ........................................................................ 31
  3.9.2 Complex Network ..................................................................... 32
3.10 User Interface ................................................................................. 33
3.11 Data and Analysis ........................................................................... 34

4 Result One: MLN Analysis and Plug-in Design ........................................... 37
  4.1 Analysis of MLN Functions and Operations ......................................... 37
  4.2 Plug-in Design ................................................................................. 41
    4.2.1 Common Variables .................................................................. 41
    4.2.2 Create File System .................................................................. 41
    4.2.3 Mount VMDK File System on the Linux .................................... 43
    4.2.4 Umount VMDK File System on the Linux .................................. 43
    4.2.5 Create Start and Stop Script for Virtual Machine ......................... 43
    4.2.6 Generate VMX Configuration File .............................................. 45
    4.2.7 Check Virtual Machine Status .................................................... 48
    4.2.8 Check Virtual Switch Status ....................................................... 49
    4.2.9 Remove a Virtual Machine ......................................................... 49
    4.2.10 Remove a Virtual Switch .......................................................... 50
    4.2.11 Start a Virtual Switch ............................................................... 50
    4.2.12 Stop a Virtual Switch ............................................................... 51
    4.2.13 Manage Virtual Switches During Upgrade ................................. 52
  4.3 Chapter Summary ............................................................................. 53

5 Result Two: Automatic Network Administration ......................................... 55
  5.1 Simple Topology .............................................................................. 55
    5.1.1 Network Topology .................................................................. 55
    5.1.2 Design Project File .................................................................. 55
    5.1.3 Build Network ......................................................................... 58
    5.1.4 Start Network ......................................................................... 60
    5.1.5 Network View from the VMware ESX Server ............................... 60
    5.1.6 Start Only One Host ................................................................. 62
    5.1.7 Stop Only One Host ................................................................. 62
    5.1.8 Stop Network .......................................................................... 62
    5.1.9 Remove Network ..................................................................... 64
  5.2 Complex Topology ........................................................................... 64
    5.2.1 Network Topology .................................................................. 64
    5.2.2 Design Project File .................................................................. 66
    5.2.3 Build Network ......................................................................... 66
    5.2.4 Start Network ......................................................................... 67
List of Figures

2.1 A system formed by VMware products ........................................ 8
2.2 How VMware vShield products secure a system .............................. 9
2.3 A vSS connects a virtual machine to the physical network ............... 10
2.4 A vDS connects a virtual machine to the physical network ............... 11
2.5 Inheritance demonstration ....................................................... 18

3.1 Topology graph ................................................................. 21
3.2 Procedure graph .................................................................. 22
3.3 Call graph .......................................................................... 23
3.4 Design diagram of combining user interface with MLN and plug-in ........ 33

4.1 Sub function sequence diagram of MLN when building network ......... 39
4.2 MLN’s sub function sequence diagram when running mln status .......... 40

5.1 Simple network topology ........................................................ 56
5.2 Network view from VMware vSphere Client .................................. 61
5.3 Network view without host Oslo ............................................... 63
5.4 Small business network topology ............................................... 65
5.5 Network view from VMware vSphere Client .................................. 68

6.1 Check disk size from Datastore Browser ...................................... 75
6.2 Check memory size from host London ......................................... 76
6.3 Verify new user from host Paris ............................................... 77
6.4 Verify network configuration in host Paris ..................................... 78
6.5 Upgrade network topology ........................................................ 79
6.6 New topology from VMware ESX server ...................................... 81
6.7 Migrate schematic diagram ...................................................... 83
6.8 Virtual machine city displayed in datastore1 ................................. 84
6.9 Virtual machine city displayed in new VMware ESX server ............... 86
6.10 User interface function graph ................................................... 86
6.11 User interface index web page .................................................. 87
List of Tables

2.1 Main products provided by VMware Corporation .......................... 5
2.2 Explanation of MLN keywords ............................................. 17

3.1 Package name ................................................................. 24
3.2 Basic functions in the plug-in ............................................. 30

4.1 Characteristics of sub functions of MLN that calls plug-ins .......... 38
4.2 Variables designed and roles played in the plug-in ................. 41

6.1 Items supported in upgrade process ...................................... 73

7.1 Number of wizard clicks and code lines ............................... 89
7.2 Number of commands to construct network ............................ 90
7.3 Number of commands to boot network .................................. 91
7.4 Number of commands to stop network .................................. 91
7.5 Number of commands to remove network ............................... 92
Chapter 1

Introduction

Nowadays virtualization technologies have become increasingly popular and powerful. Virtualization technology has many potential advantages over server consolidation [1, 2], model/software simulation [3, 4], education/training [5, 6] and high-performance computing [7]. Server consolidation utilizes server virtualization technology to improve the efficient use of computer resources and solve the server sprawl problem in businesses [8, 9]. Model simulation is widely used to imitate the biological world by biologists and more real-time and granular information is achieved with the trend of virtualization computing [10, 11]. In the field of education and training, virtualization technology brings not only the same effects as using real machines before, but also lower cost and simpler recovery mode [6, 12]. Using physical machine for high-performance computing may bring some challenges, like heterogeneity and security, which can be mitigated by virtualization technology [13, 14].

Generally, virtual machine deployments are divided into three groups: KVM [15], Xen [16, 17] and VMware [18]. KVM is an open source software. It is a full-virtualization solution for Linux on x86 hardware [19]. Xen was also open source software before being bought by Citrix System in October 2007. The Xen uses para-virtualization technology and provides a thin layer called Hypervisor between hardware and virtual machines. VMware is the dominant commercial virtualization software today and occupies the biggest market share.

VMware has a huge family of products including vSphere, vShield, vCloud and vCenter, which provides virtualization and cloud computing solutions. Through vSphere, all hardware covering CPU, memory, network and storage is virtualized completely. VShield has two main components: One is vShield Manager. Another is vShield Edge. VShield Manager is the centralized network management component that is responsible for security service and firewall/routing appliances [20]. VShield Edge runs on hosts as virtual appliances [21]. VCloud is built over the vSphere and acts as a logical abstraction and management interface of vSphere resources [22]. VCenter Chargeback collects resource usage data, measures and workloads to generate cost and usage re-
1.1. PROBLEM STATEMENT

ports [23]. Using VMware virtualization technology brings some additional advantages, like economics of scale, elastic resources and self service provisioning. Therefore, VMware is very powerful and offers many management tools. However, all of them are commercial products. Thus it is potentially uneconomical for a company with low budget, minor work force or cost reduction to use VMware’s administration tools.

VMware offers a friendly user interface and it is simple to deploy or manage several virtual machines. However, the flowery user interface is only designed to do operations on one virtual machine, not on groups of virtual machines. Therefore it becomes tedious when a system administrator needs to administrate hundreds of virtual machines, which requires adequate patience. For example, if a network with one hundred virtual machines and fifty virtual switches are required to deploy into data center, it is a hard and complicated work. Next, administration of these virtual machines is indispensable. Booting one hundred virtual machines at one time via the VMware command is infeasible. Machines may only be booted individually which of course is very inefficient. In addition, stopping all of them is also a significant task. Therefore in this case developing an excellent administration tool is vital for system administrators to deploy network efficiently and administrate networks automatically.

Many projects have focused on the issue of virtual machine administration and a lot of tools are developed to make virtual machine administration more efficient, which helps system administrators save time on virtual machine configuration and creation [24, 12, 25]. Some of the tools are prepared for the purpose of Xen and have already altered and simplified the manner to manage virtual machines on the Xen platform. For example a tool called MLN (Manage Large Networks) makes complex virtual machines management easier than its management directly by Xen command line [26, 27]. However, few projects are aimed at the VMware platform which means that it still remains a lengthy operation to manage virtual machines on the VMware platform.

Consequently the problem is clarified distinctly that my final thesis will pay close attention to the virtual machines administration on the VMware ESX platform.

1.1 Problem Statement

The given problem statement is as follows:

Reduce deployment complexity and increase automatization as well as coherence of virtual machines and networks administration on the VMware ESX platform.

The deployment complexity means how complicated it is to design and to cre-
1.2. SUMMARY OF THE RESULTS

ate large number of virtual machines or intricate networks. It is simple to create a single virtual machine on VMware ESX but it is tedious to construct hundreds of virtual machines because they have to be created one by one. So the reduction of deployment complexity means to reduce the number of operations to create virtual machines.

Currently, manner to start/stop one virtual machine on the VMware ESX platform is to go to start/stop menu and click corresponding button. So start/stop one hundred virtual machines needs one hundred clicks. Thus the word automatization means automation degree of administrating virtual machines and networks. And the degree of automation needs to be increased. The aim of the automation is to manage large number of virtual machines by automatic operations.

If many virtual machines are almost identical clones, they should have some identical attributes. If these common attributes can be extracted and collected into a parental class then other clones can inherit from this class. So the attributes of virtual machines can fit each other well, which states the implication of coherence. To be exact coherence means how to integrate configurable attributes of virtual machines and fit them together well which expresses a clear pattern to organize the configurable attributes of virtual machines.

Next keyword is the virtual machine. The virtual machine is not a real machine. It is simulated by software. Several virtual machines are running on one physical machine at the same time and isolated from each other.

Any operations to virtual machines including creating, starting/stopping, and removing are understood as virtual machine administration as stated in the problem statement.

The VMware ESX platform refers to the enterprise level virtualization product VMware ESX 4.0. This project will be implemented on this platform.

1.2 Summary of the Results

The results are divided into four sections. In the first section hooks of MLN preparing for plug-ins are analysed and its call graphs are drawn. Essential code of the plug-in esx.pl and its design are explained. Next section new method of automatic administrating networks is demonstrated through two topologies. The first topology is simple and its goal is to introduce how to utilize the method. The second topology is complex. It is used to elaborate how to administrate a complicated network automatically. In the third section two advanced features upgrade and migration are developed and integrated into the plug-in and their functionality is demonstrated one by one based on the former topology. In addition, a user interface is created to order and deploy virtual machines and networks more conveniently. The last section the
network deployment complexity and coherence and automatic administration degree are analysed. The deployment complexity is dropped. While the automatic administration degree and coherence are increased.

1.3 Thesis Outline

This paper will be structured in the following manner: Chapter one will state motivation and problem statement. Chapter two will introduce VMware virtualization technology, vCLI and MLN. Complete solution design, principle and analysis are elaborated in chapter three. Since result is enormous, so that it is divided into four chapters, from chapter 4 to chapter 7. MLN analysis and plug-in design is explained in chapter 4. Two real live cases are demonstrated in chapter 5. Two notable features and user interface are illustrated in chapter 6. Further, data is collected, compared and analysed in chapter 7. In chapter 8, design principle and feasible approach are discussed. Solved and remaining problems are listed. Moreover, project achievement, influence and future work are stated. Final chapter is conclusion to summarise the whole thesis.
Chapter 2

Background

In this chapter the VMware technologies and terminology are introduced. Next, the virtual machine administration tool MLN is presented briefly.

2.1 VMware Virtualization Technology

2.1.1 VMware Product Family

VMware is the biggest virtualization software provider in the world. According to different demands it offers various of products. So it is clear to list most of the critical products in a table below [28].

<table>
<thead>
<tr>
<th>Function</th>
<th>Product Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Centre &amp; Cloud Infrastructure</td>
<td>VMware vSphere&lt;br&gt; vSphere Hypervisor(ESX/ESXi)&lt;br&gt; VMware Go&lt;br&gt; vCloud Director&lt;br&gt; vCloud Request Manager</td>
</tr>
<tr>
<td>Desktop &amp; End User Computing</td>
<td>VMware ThinApp&lt;br&gt; VMware ACE&lt;br&gt; VMware Workstation&lt;br&gt; VMware Player&lt;br&gt; VMware MVP</td>
</tr>
<tr>
<td>Infrastructure &amp; Operations Management</td>
<td>VMware vCenter Server&lt;br&gt; vCenter Server Heartbeat&lt;br&gt; vCenter Orchestrator&lt;br&gt; vCenter Converter&lt;br&gt; Site Recovery Manager</td>
</tr>
<tr>
<td>Security</td>
<td>VMware vShield Manager&lt;br&gt; VMware vShield App&lt;br&gt; VMware vShield Edge&lt;br&gt; VMware vShield Endpoint</td>
</tr>
</tbody>
</table>

Table 2.1: Main products provided by VMware Corporation
2.1. VMWARE VIRTUALIZATION TECHNOLOGY

VSphere is an industrial level virtualization platform to build cloud infrastructures. VSphere Hypervisor is installed with VMware ESX/ESXi [29]. VMware Go is a web based management interface. Through it the hypervisor can be created or updated on physical machine. And system administrators can create, migrate virtual machines or install new patches [30]. Coupled with vSphere, vCloud Director can add virtual applications into a cloud helping users build secure private clouds and control services over a multi-tenant environment [22]. Next product is VMware vCloud Request Manager. It can assist VMware vCloud Director to improve private cloud control by enforcing business policies and procedures. Besides, it can avoid virtual sprawl on private cloud infrastructure and provide a simple cross-browser compatible portal to cloud consumers [31].

With VMware ThinApp, applications can be packaged into many single executables that are completely isolated from operating systems. Moreover, these executables can be deployed into new platforms effortlessly. For example, re-coding cost and regression testing are able to be eliminated on Windows 7 [32]. The second product is VMware ACE (Assured Computing Environment). It treats desktops as a managed service and extends corporate resource to form an united computing environment [33].

The third product is called VMware Workstation. VMware released the first Workstation in May, 1999. Until now it has evolved into the seventh edition. Nowadays, it has become a popular desktop virtualization software. Besides it is a good platform to develop and test demo software for technical professionals. For hardware it supports maximum 4 virtual processes and 32 gigabyte memory and over 200 different kinds of operating systems which means that almost all the applications can run on the Workstation 7 even included VMware vSphere or Hypervisor ESX/ESXi. Moreover, more flexible snapshots can be taken and 256 bits AES encryption is supported to achieve high security [34].

With VMware Player users can run multiple operating systems in one machine at the same time and VMware Player makes it effortless for anyone to try out Windows 7, Chrome OS or the latest Linux releases [35]. Next product is the VMware MVP (Mobile Virtualization Platform). It provides a freedom way to work from their own smart-phone, while it brings security control and eases management to IT department [36].

Next another collection of products belonging to Infrastructure and Operations Management. The first product is VMware vCenter Server. It forms the foundation of virtualization management. It manages vSphere environment centrally and allows system administrators to improve control dramatically over virtual environment [37]. Second product is vCenter Server Heartbeat. It delivers high reliability to VMware vCenter server against planned and unplanned downtime with rapid fail-over and fault recovery on both physical and virtual platforms [38]. The third product is VMware vCenter Orchestrator.
2.1. VMWARE VIRTUALIZATION TECHNOLOGY

It is utilized to capture daily routine work and transferred them into work flow which can be executed automatically by VMware vSphere. VMware vCenter Orchestrator also provides some sample work flows for system administrators to design blueprints to satisfy their own work demands [39]. The fourth product is VMware vCenter Converter. It is a robust and enterprise class migration tool that converts local or remote physical machines into virtual machines without any disruption or downtime. Moreover, it is able to convert some special formats such as Microsoft Hyper-V [40, 24], Microsoft Virtual PC [41, 42] and Microsoft Virtual Server [43] or backup images created by Symantec Backup Exec System Recovery or Norton Ghost to virtual machines [44]. Last product is VMware vCenter Site Recovery Manager. It can build, manage and execute disaster recovery plans reliably and help system administrators take problem away from system [45].

VMware vShield family comprises four products related security. First product called VMware vShield Manager offers role based access control mechanism and acts as a management interface for all vShield products. It can also help the third party’s security component to integrate into system [20]. Next product is called VMware vShield App. It can protect applications from network based threats in virtual datacenter keeping watch over network and enforce granular policies within security groups. Therefore, it provides a cost-effective protection solution which is cheaper than physical security and eliminates hardware and policy sprawl. Third product is VMware vShield Edge. By employing VMware vShield Edge, virtual datacenter can achieve comprehensive bound network security. Detailed log messages are provided to simplify IT compliance. Moreover, a web load balancer is integrated into VMware vShield Edge that can manage inbound web traffic across virtual machines [21] efficiently. Last product is VMware vShield Endpoint. It can streamline and accelerate antivirus and anti-malware software deployment and eliminate susceptible attacks. Audit is also provided from detailed log [46].

2.1.2 VMware Virtualization System Architecture

In this section, virtualization system architecture and security components are presented.

Figure 2.1 describes the architecture of VMware virtualization system briefly. It is divided into four layers. The bottom layer is storage that is sorted into private cloud and public cloud. Private cloud refers to any proprietal data servers or disks. Public cloud means any rented storage infrastructures from service providers. Upwards the second layer is vSphere layer. It is separated into two sub-layers. One is the infrastructure sub-layer which is separated into three parts. First part is the computing part which contains ESX, DRS and DRM. ESX is the hypervisor of VMware virtualization system. Second, aiming to a company’s business goal, DRS (Distributed Resource Scheduler)[47] is used to allocate and balance computing resource dynamically. Third part is DRM (Digital Rights Management) [48]. It is an access control system to limit
2.1. VMWARE VIRTUALIZATION TECHNOLOGY

Figure 2.1: A system formed by VMware products

the use of digital content and devices. Next component is storage. VMware has a file system called VMFS (Virtual Machine File System) [49] optimized for virtual machines. Last component is network that has two kinds of virtual switches. One is VSS (vNetwork Standard Switch). Another is VDS (vNetwork Distributed Switch).

Upwards second sub-layer is application layer. It is divided into three sections as well. First part is availability. It offers four main functions that are virtual machines migration (vMotion) [50, 51], storage migration (storage vMotion), fault tolerance [52] and data recovery. VMotion is used to transfer virtual machines from one machine to another machine or storage. Fault tolerance keeps system running in the event of one or more components of the system in failure. Besides, it is the security part which comprises VMware vShield App, VMware vShield Edge and VMware vShield Endpoint. The last part is DRS and Hot Add to increase system scalability. vSphere supports hot-plug vCPU and virtual network adapter (NIC) as well as hot-add RAM and virtual disk. Virtual network adapter and virtual disk can be recognized by guest operating system without downtime. However, vCPU and RAM can not be detected until guest operating system reboots itself. Over the vSphere layer, it is vCenter layer upon which virtual machines and applications are running.

Second picture figure 2.2 demonstrates how VMware vShield products are devoted to secure system. New layer, vShield Manager is located over vCenter
2.2 VMware Terminology

VMware has many new terminology. Therefore, it is necessary to state them exactly.

2.2.1 Datacenter

A datacenter contains many clusters and a cluster contains many ESX/ESXi hosts. Usually a datacenter is used to apportion computing resource by geographical location or IT department. The default datacenter is called *ha-datacenter* in ESX/ESXi host. A big company usually has many datacenters that are located in different parts of a city or country. However, a small company may has only one datacenter.
2.2. VMWARE TERMINOLOGY

2.2.2 Datastore

A datastore is the storage container for virtual machines in the ESX/ESXi host. Storage device can be a local SCSI/IDE disk or remote disk, such as fiber channel or iSCSI. The datastore file system is *vmfs3*.

2.2.3 Virtual Machine File System (VMFS)

VMFS is designed for VMware high performance cluster. It is used to store virtual machine disk image or snapshot. Disk image consists in two files. One file is small with extension *vmdk*. It can be treated as the configuration file of disk image. Another file is big with suffix *-flat.vmdk*, which has the same size of virtual disk.

2.2.4 Virtual Switch

VMware provides virtual switch to connect virtual machines to form a network. A virtual switch is divided into two kinds. One is vSS (vNetwork Standard Switch). Another is vDS (vNetwork Distributed Switch). Subnets connected by virtual switch are called PortGroup. Next two pictures illustrate what kind of role vSS and vDS play in network.

![vSS and vDS](image)

Figure 2.3: A vSS connects a virtual machine to the physical network

From picture 2.3, virtual machine is able to connect to internet from vSS that connects to physical network adapter. Subnet where virtual machine is located is called Port Group. One host can deploy more than one vSS or Port Group. And one vSS can connect to more than one physical network adapter to implement fault tolerance or load balance. Next picture demonstrates what role
vDS plays.

VDS is a distributed switch that spans more than one host. A subnet is called dvPort Group and it can also span many hosts. Virtual machines running on different hosts can communicate each other through vDS. VDS connects to a physical network adapter through a special network adapter dvUplink. It plays as the single virtual switch associated with hosts. Thus consistent network configuration is able to be maintained when virtual machines migrate from one host to another.

2.2.5 Virtual Disk Development Kit (VDDK)

In order to mount VMware file system, VDDK [53] is necessary to be installed. It is a collection of virtual disk and disk mount libraries. They are sets of C functions to manipulate virtual disk files. Disk mount utility can be used to access files and file system when virtual disks are in offline mode. Besides, virtual disk manager can be used to manipulate offline virtual disk which includes clone, create, relocate, rename, grow, shrink and defragment virtual disk.

2.3 vSphere Command Line Interface (vCLI)

VCLI (vSphere Command Line Interface) is the console interface for Linux and Windows to manage VMware vSphere ESX platform. System administrators can manage virtual machines, storage, vSphere network and users by vCLI.
2.3. VSPHERE COMMAND LINE INTERFACE (VCLI)

However, in order to manage virtual machines by vCLI, an additional package, VMwareTools is required to install on the guest operating system.

2.3.1 Manage Virtual Machines

VCLI provides some commands to administrate virtual machines and they are illustrated in this section.

Display Virtual Machines

```
vmware-cmd --server SERVER_NAME --username USERNAME --password PASSWORD -l
```

Running this command can list registered virtual machines. Created virtual machines stored in the datastore but unregistered are not listed. This command can also be written like this expressed below.

```
vmware-cmd -H SERVER_NAME -U USERNAME -Q PASSWORD -l
```

This command lists the VMX file of each registered virtual machine. A demo result can be like this.

```
/vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/cfmanager.cf/cfmanager.vmx
/vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/toy.cf/toy.vmx
```

Two virtual machines, cfmanager.cf and toy.cf are listed here, which are stored in the volume labelled with 4c74cc2d-f3d16a79-36d5-b8ac6f21f590.

Register a Virtual Machine

A virtual machine needs to be registered before powering it on.

```
vmware-cmd --server SERVER_NAME --username USERNAME --password PASSWORD -s register VIRTUAL_MACHINE.vmx
```

This command is used to register a virtual machine into Inventory. If it is registered successful, register() = 1 will be returned.

Obtain a Virtual Machine Running Status

```
vms-cmd --server SERVER_NAME --username USERNAME --password PASSWORD /vmfs/volumes/VMFSLABEL/VIRTUAL_MACHINE/VIRTUAL_MACHINE.vmx getstate
```

Using getstate can obtain the state of a virtual machine. If a virtual machine is running, the output will be getstate() = on. If it does not run, the output will be getstate() = off.

Power on a Virtual Machine

```
vms-cmd --server SERVER_NAME --username USERNAME --password PASSWORD /vmfs/volumes/VMFSLABEL/VIRTUAL_MACHINE/VIRTUAL_MACHINE.vmx start
```

Using start can power on a virtual machine immediately. If it boots successfully, start() = 1 will be returned.
Power off a Virtual Machine

Comparing the starting of a virtual machine to the powering off of a virtual machine is more complicated. Except using the `stop` command, a power mode flag, either `soft` or `hard` is required.

Adding the `soft` flag simulates the normal way to shut down a virtual machine. It attempts to shut down the guest operating system, and then power off the virtual machine.

```
vmware-cmd --server SERVER_NAME --username USERNAME --password PASSWORD /vmfs/volumes/VMFSLABEL/VIRTUAL_MACHINE/virtual_machine.vmx stop soft
```

Another way is like to pressing the power button for a long time to shut down a virtual machine immediately by using the `hard` flag. It shuts the virtual machine down instantaneously without caring about the guest operating system. The command looks like this below.

```
vmware-cmd --server SERVER_NAME --username USERNAME --password PASSWORD /vmfs/volumes/VMFSLABEL/VIRTUAL_MACHINE/virtual_machine.vmx stop hard
```

If a virtual machine is stopped successfully an output `stop() = 1` will be printed.

In some cases, when virtual machines are not response to the normal shutdown, it might be necessary to forcibly shut them down. Forcibly shutting down a virtual machine is similar to pulling the power cable on a physical machine. The command to forcibly shut down a virtual machine is `esxcli vms vm kill`.

```
esxcli --server SERVER_NAME --username USERNAME --password PASSWORD vms vm kill --type KILL_TYPE --world-id WORLD_ID
```

This command supports three –type options.

- `soft` - Use the VMX process to shut down the virtual machine
- `hard` - Stop the VMX process immediately
- `force` - Kill the VMX process immediately

There three options are advised to try sequentially to shut down a virtual machine. In order to shut down a virtual machine, its World ID should be obtained first. Running this command can get the World ID.

```
esxcli --server SERVER_NAME --username USERNAME --password PASSWORD vms vm list
```

And the output is like this. From the rich output, the World ID 1689298 is achieved.

```
cfmanager.cf
   World ID: 1689298
   VMX Cartel ID: 1689297
   UUID: 56 4d 58 3b 7b f2 4e 78 25 c7 ed 5a fd bf 14 00
   Display Name: cfmanager.cf
   Config File: /vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/cfmanager.cf/cfmanager.vmx
```
2.3.2 Manage Virtual Switches

The virtual network consists of virtual switches and subnets. A virtual switch may be called vSwitch and the subnet may be called port group. vCLI provides a powerful command `vicfg-vswitch` to manage vSwitch and port groups which are introduced here.

**Check Virtual Switch Existence**

A switch is used to connect computers to form a network so that it is an indispensable device. So it is necessary to check a switch existed or not before creating a switch.

```
vicfg-vswitch --server SERVER_NAME --username USERNAME --password PASSWORD --c VIRTUAL_SWITCH
```

This command can check the `VIRTUAL_SWITCH` exists or not. If it does not exist, zero will be returned. Otherwise one will be returned.

**Check Subnet Existence**

It is also necessary to check a subnet existence before creating a subnet.

```
vicfg-vswitch --server SERVER_NAME --username USERNAME --password PASSWORD --check-pg 'VM_NETWORK' VIRTUAL_SWITCH
```

This command is used to check whether virtual switch `VIRTUAL_SWITCH` owns a subnet called `VM_NETWORK` or not. If no subnet `VM_NETWORK` pertains to `VIRTUAL_SWITCH`, a zero will be returned. Otherwise one will be returned.

**Create a Virtual Switch**

```
vicfg-vswitch --server SERVER_NAME --username USERNAME --password PASSWORD --add VIRTUAL_SWITCH
```

This command is used to create a virtual switch. If it is created successfully, no output will be printed. Otherwise an error message will be printed.

The specified key, name, or identifier already exists.

**Create a Subnet**

```
vicfg-vswitch --server SERVER_NAME --username USERNAME --password PASSWORD --add-pg 'VM_NETWORK' VIRTUAL_SWITCH
```

This command is used to create a subnet called `VM_NETWORK` belonging to virtual switch `VIRTUAL_SWITCH`. If it is created successfully, no output will be printed. Otherwise an error message will be printed to say an object is missing.

The object or item referred to could not be found.
2.3. VSphere Command Line Interface (VCLI)

Delete a Subnet

```
vicfg-vswitch --server SERVER_NAME --username USERNAME --password PASSWORD --
del-pg 'VM_NETWORK' VIRTUAL_SWITCH
```

This command removes a subnet VM_NETWORK appertaining to a virtual switch VIRTUAL_SWITCH. If it is removed successfully, no output will be printed. Otherwise the subnet can not be found and an object missing alert will be printed to the console.

The object or item referred to could not be found.

Delete a Virtual Switch

```
vicfg-vswitch --server SERVER_NAME --username USERNAME --password PASSWORD --
delete VIRTUAL_SWITCH
```

This command is used to delete a virtual switch. If it is removed successfully, no output will be printed. Otherwise an object missing error will be printed as the same as the error of removing a subnet.

2.3.3 Manage Virtual Disks

VMware provides a package called VMware Virtual Disk Manager that can mount, convert, expand, defragment and shrink virtual disk.

Convert a Virtual Disk

```
vmware-vdiskmanager --r FILE_SYSTEM.vmdk --t TYPE ESX_FILE_SYSTEM.vmdk
```

This command is provided by VMware Virtual Disk Manager. For TYPE, it provides seven options, from zero to six.

- 0 - Create a growable virtual disk contained in a single file
- 1 - Create a growable virtual disk split into 2GB files
- 2 - Create a preallocated virtual disk contained in a single file
- 3 - Create a preallocated virtual disk split into 2GB files
- 4 - Create a preallocated virtual disk compatible with ESX server
- 5 - Create a compressed disk optimized for streaming
- 6 - Create a thin provisioned virtual disk

If the TYPE is given to 4, this command will convert a file system to a ESX file system. And if the connection information is provided, such as server name, username and password, this command can convert the file system remotely.
2.3. VSphere Command Line Interface (VCLI)

Expand a Virtual Disk
VMware Virtual Disk Manager offers an option -x to grow a virtual disk. The size of the new disk should be specified and the unit can be in gigabytes (GB), megabytes (MB) and kilobytes (KB).

```
vmware-vdiskmanager -x SIZE [GB|MB|KB] NEWDISK.vmdk
```

2.3.4 Manage Files
Like Unix, virtual machines consist in files and directories in the VMware ESX. Disk, firmware, memory, CPU and network card can be treated as files and virtual machine can be treated as a directory. So creating and moving virtual machine can be treated as operations to files and directories.

Create a Directory

```
vi fs --server SERVER_NAME --username USERNAME --password PASSWORD --mkdir '[DATASTORE] VIRTUAL_MACHINE'
```

This command is used to create a directory to contain files for a virtual machine. If a directory is created successfully, an output followed will be printed.

```
Created directory '[DATASTORE] VIRTUAL_MACHINE' successfully.
```

Upload a File

This command is used to upload a file (vmx, vmdk or flat file) to a remote VMware ESX host.

```
vi fs --server SERVER_NAME --username USERNAME --password PASSWORD --put ./ FILE '[DATASTORE] VIRTUAL_MACHINE/FILE'
```

Download a File

This command is used to download a file from a remote host.

```
vi fs --server SERVER_NAME --username USERNAME --password PASSWORD --get '[DATASTORE] VIRTUAL_MACHINE/FILE' ./ FILE
```

Delete a Directory

This command with -force option can remove a directory without any prompts.

```
vifs --server SERVER_NAME --username USERNAME --password PASSWORD --rm '[DATASTORE] VIRTUAL_MACHINE' --force
```
2.4 Virtual Machine Administration Tools

2.4.1 MLN (Manage Large Network)

MLN is a virtual machine administration tool that is designed to build and manage virtual machines or networks in a simple and efficient way. It can be utilized to manage virtual machines on Xen, UML (User Mode Linux), VMware Server platform, and Amazon EC2 (cloud computing platform), but except KVM and VMware vSphere ESX/ESXi. Using MLN is able to ease construction and configuration of virtual machines and networks so that it is necessary to introduce MLN.

2.4.2 MLN Programming Language

Keyword

Keywords are used to define virtual machine attributes or virtual switches. They are explained in the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>Name of a built project</td>
</tr>
<tr>
<td>host</td>
<td>Name of a virtual machine</td>
</tr>
<tr>
<td>switch</td>
<td>Name of a virtual switch</td>
</tr>
<tr>
<td>template</td>
<td>File system running in a virtual machine</td>
</tr>
<tr>
<td>size</td>
<td>Disk capacity of a virtual machine</td>
</tr>
<tr>
<td>free_space</td>
<td>Extra disk capacity added to a virtual machine</td>
</tr>
<tr>
<td>memory</td>
<td>Memory of a virtual machine</td>
</tr>
<tr>
<td>nameserver</td>
<td>IP address of the DNS server known by a virtual machine</td>
</tr>
<tr>
<td>network</td>
<td>Network configuration description</td>
</tr>
<tr>
<td>address</td>
<td>IP address assigned to a virtual machine</td>
</tr>
<tr>
<td>netmask</td>
<td>Subnet mask of the subnet</td>
</tr>
<tr>
<td>gateway</td>
<td>IP address of the gateway</td>
</tr>
<tr>
<td>superclass</td>
<td>Common attributes of a host</td>
</tr>
<tr>
<td>global</td>
<td>To state the name of the project here</td>
</tr>
<tr>
<td>users</td>
<td>Users added to the virtual machine</td>
</tr>
</tbody>
</table>

Table 2.2: Explanation of MLN keywords

From table 2.2, it displays some common keywords. According to their position in project file, they are divided into two groups. Block keywords belong to the first group, including global, host, switch, network and users. Other keywords belong to another group. Next, it is clear to give some clips of code to express how they work.
There is a piece of code using keyword `superclass` with value `attribute`. Since it is a block keyword, more attributes can be defined within curly brackets. First, type of the template, `hvm` is declared and template `Windows-Server-2003.ntfs` is specified. Next, memory, disk size and IP address of DNS server are assigned to 3GB, 10GB and “128.73.30.8”. Last, another block keyword is `network`. It represents network adapter and its value is `eth0`. In this block, IP address, subnet mask and gateway are assigned by DHCP protocol.

### Inheritance

Usually a machine has many attributes that can be described here. And in a network many machines often have the same attributes which can be summarised and stated in a superclass block. Moreover, the attributes can be extracted and abstracted and distributed into different superclass blocks. When a host is built, it can inherit attributes from different superclasses. Furthermore, superclass is able to inherit attributes from other superclasses to make up a bigger and clearer structure, like a tree or a net, which has the same concept of inheritance as OOP (Object Oriented Programming). Next, an example is given to show how it works.

![Inheritance Diagram](image)

**Figure 2.5: Inheritance demonstration**

This picture explains the relationship among superclasses and host. Three superclasses are defined to represent operating system, disk and memory. Next, a superclass called `common-machine` inherits the three superclasses to form a
2.4. VIRTUAL MACHINE ADMINISTRATION TOOLS

machine. Finally, DNS-server inherits the common-machine to become a name server. Next, a piece of code is demonstrated to implement the structure.

```c
superclass common-OS {
    template Windows-XP.ntfs
}
superclass common-disk {
    size 10GB
    free_space 1GB
}
superclass common-memory {
    memory 512M
}
superclass common-machine {
    superclass common-OS
    superclass common-disk
    superclass common-memory
}
host DNS-server {
    superclass common-machine
    template Windows-Server-2008.ntfs
    memory 4GB
}
```

From this piece of code, three father superclasses are defined. One child superclass inherits father’s superclasses. In the end, a host is constructed by inheriting the child superclass. Besides, variables can be rewritten locally that means any variables can be rewritten into the children’s block. For example, template and memory are updated in the DNS-server.

Moreover, in order to build larger network, another important keyword include is offered by MLN. By using include, Other network or system configuration can be distributed into different files or imported into current project seamlessly. It is simple to utilize include by adding one line showed below.

```c
#include another-network.mln
```
Chapter 3

Approach

In this chapter, system design is stated. Technologies are illustrated in detail. Plug-in esx.pl is structured. Moreover, the user interface is contrived. The last section data metrics are listed and analysis method is explained.

3.1 System Design

3.1.1 Topology

Two physical machines and one rack server are deployed for this project. One physical machine and rack server are used for VMware ESX servers. Another physical machine running Linux acts as a manager who sends commands to two ESX servers and receives results from them. MLN, plug-in esx.pl and vCLI are also installed on this machine. Topology of final project is expressed in the following graph.

Figure 3.1: Topology graph
3.1. SYSTEM DESIGN

Two ESX servers and a Linux server are connected together by a switch. Three public IP addresses are assigned to each of them. A laptop running VMware vSphere Client, putty and VPN connects to these machines through internet.

Console is removed from new generation ESX server. An external console is needed to connect to ESX server. And VMware vSphere Client is a dedicated console designed for ESX server. Thus it is selected and installed on the laptop. Since port 443, 902 and 903 are blocked by the firewall, a VPN is necessary to run on the laptop. MLN is able to run only on the Linux platform, so Linux is chosen for the machine. MLN is written in Perl and its plug-ins are only accepted in Perl, so Perl is selected to develop the plug-in.

3.1.2 Procedure

From command sending to result receiving, whole procedure will span approximately five steps. Plug-in esx.pl plays a critical role to execute commands and receives feedback.

![Procedure graph]

The first step, commands are sent to the Manager machine. Commands include operations to build, start, stop, check or remove network. The second step, MLN receives a command and executes specific functions of esx.pl. For example, if building network is executed, the network will be created and a copy will be stored locally. If starting network is executed, a local starting script will be called. Third step esx.pl transmits command to ESX server. It is executed remotely and meanwhile the network is transmitted. Fourth step, return value is sent back to esx.pl. Fifth step, final state is decided by esx.pl
3.1. SYSTEM DESIGN

and sent back to the console.

The function of esx.pl should be stated clearly. If one goes deeper, the next graph expresses how MLN, esx.pl and ESX server call each other in the bottom level.

![Call graph]

Figure 3.3: Call graph

From figure 3.3 red arrow means that commands or data is sent to next station. Blue arrow means that return value or feedback is transferred back. In the center of call graph, a big square is filled with name of sub functions. These functions in esx.pl is designed to do a specific administration task. For example the function `esx_createFilesystem` is designed to create, convert and expand file system.

How to divide function feature and function name is decided from MLN principal code. Features are divided and distributed in different functions in MLN. For example, when a virtual switch is built, MLN will call `buildSwitch` to do it. Furthermore, call windows of plug-in sub functions have been defined as well, which is used to extend features of MLN to different platforms. On the demand of ESX plug-in, call windows are examined and selected to be a function member of ESX plug-in esx.pl. For function name, according to MLN naming rule, `esx` is added as a prefix in front of each function.

3.1.3 Package Control

Except Linux operating systems, some additional packages are necessary to install. Since operating system selected is 64 bit, thus other packages advised to use are 64 bit. Since a package has many versions, so it is better to show the
3.1. SYSTEM DESIGN

exact version installed in this project. A table below shows each package’s full name contained version number.

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian</td>
<td>debian-6.0.0-amd64-CD-1.iso</td>
</tr>
<tr>
<td>VMwareTools</td>
<td>VMwareTools-8.3.2-257589.tar.gz</td>
</tr>
<tr>
<td>vSphere ESX</td>
<td>VMware-VMvisor-Installer-4.1.0-260247.x86_64.iso</td>
</tr>
<tr>
<td>vCLI</td>
<td>VMware-vSphere-CLI-4.1.0-254719.x86_64.tar.gz</td>
</tr>
<tr>
<td>VDDK</td>
<td>VMware-vix-disklib-1.2.1-323406.x86_64.tar.gz</td>
</tr>
<tr>
<td>vSphere Client</td>
<td>VMware-viclient-all-4.1.0-258902.exe</td>
</tr>
<tr>
<td>MLN</td>
<td>mln-1.0.7.tar.gz</td>
</tr>
<tr>
<td>Fuse</td>
<td>fuse-2.8.5.tar.gz</td>
</tr>
</tbody>
</table>

Table 3.1: Package name

First software is the stable 64 bit debian operating system. Second package is VMwareTools. It is required to install on guest operating system to shut virtual machine down fluently. Otherwise the virtual machine has to be shut down forcibly as equivalent of pulling the power from a physical machine. Since VMwareTools is dependent on library linux-headers, but linux-headers is unavailable on unstable 64 bit debian operating system. So stable debian is chosen to install. Next package is VMware vSphere ESX that is hypervisor of virtualization platform. It is needed to install definitely. Since the server running ESX has Intel Core2 Duo CPU that supports 64 bit virtualization technology, so 64 bit ESX package is selected. Fourth package is vCLI. From figure 3.3, vCLI plays a role in executing final commands to storage. Since ESX server is 64 bit on the other end, so vCLI is chosen 64 bits as well to provide seamless support. Fifth package is VDDK installed on the Manager machine. It is involved to create, convert, expand, shrink and mount virtual disk on the Manager machine during executing MLN administration commands. Since debian and guest operating system are 64 bit, VDDK has to be 64 bit to support them. Next package is VMware vSphere Client. Since new generation ESX package removes console from hypervisor, an external console is required, but to install on another machine. So it is installed on laptop showed in figure 3.1. Seventh package is MLN which latest version is installed. Last package is fuse that is a loadable kernel module for Linux operating system to run operating system in user space, which is called by VDDK.

3.1.4 Enable VT (Intel Virtualization Technology)

VT is hardware virtualization technology. Intel CPU Core2 Duo supports VT and it is disabled by default. However, it is necessary to be enabled to run 64 bit ESX and guest operating system. Moreover the debian operating system and other packages installed are 64 bit, so VT is required to be enabled.
3.2 Obtain VMX and VMDK Files

Within preparation, one VMX file and two VMDK files are necessary to be obtained. Others like VMSD and VMXF files are generated automatically on the basis of content of VMX and VMDK files. VMX file is virtual machine hardware description file. VMDK is virtual disk description file. They are used to determine a virtual machine. So it is critical to obtain these files and two VMDK files acting as template will be used by MLN.

It will take approximate four steps to obtain these files. First step is to create a virtual machine by wizard. It is straightforward, but several places need to be chosen correctly. 64 bits operating system needs to be chosen. Virtual machine version needs to be chosen 7, since it is compatible with ESX 4.0. Second step is to install the debian operating system. It is also straightforward. One powers on virtual machine, insert ISO image from local disk and shift to console to install it. Third step is to install VMwareTools on the debian operating system. In order to stop virtual machine normally, VMwareTools is required to install. VMwareTools is installed from source code downloaded from VMware official website. After that, final step is to execute following VMware commands to download VMX and VMDK files to Manager machine from console on laptop.

File system consists of two files. First one with vmdk extension is file system configuration file. Second one encapsulated in -flat.vmdk is the real disk file. Both of them are downloaded and stored in /opt/mln/template/ which is specified by MLN to achieve file system. Third file is virtual machine’s configuration file with vmx extension and some items of this file needs to be modified.

3.3 Edit VMX File

VMX file needs to be edited. Some items needs to be emptied while some items needs to be modified.

3.3.1 Modified Items
3.4. DEPLOY VIRTUAL MACHINE ONLY BY TEMPLATE AND VCLI

Above a list with five items need to be modified. The value VMTEST should be changed to the name of virtual machine system administrators expected. For example, System administrator expects a virtual machine called Monitor. So displayName should be Monitor and other proper places should be filled with Monitor as well. Then after this virtual machine is registered from Datastore to Inventory, Monitor will be displayed on the screen of vSphere Client.

3.3.2 Emptied Items

<table>
<thead>
<tr>
<th>uuid.location</th>
<th>&quot;56 4d a5 30 30 02 45 08−c9 b8 a0 bb b1 23 05 30&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid.bios</td>
<td>&quot;56 4d a5 30 30 02 45 08−c9 b8 a0 bb b1 23 05 30&quot;</td>
</tr>
</tbody>
</table>

Above two items uuid.location and uuid.bios are generated every time when virtual machine is registered. And their value is unique among the whole ESX server. Thus emptying these items is necessary before registering new virtual machine. Although they are emptied, they are generated automatically when virtual machine is registered.

3.4 Deploy Virtual Machine only by Template and vCLI

Template including VMX and VMDK files have been obtained. A virtual machine called Node will be built from these files. If it can be done successfully virtual machine does not need to be built from scratch, which brings a new way to create a virtual machine. On the ESX platform virtual machines consist of files. Operation of virtual machines is transformed to operation of files. Thus creating and administrating virtual machines is to create and administrate these files. Upgrade virtual machine is to modify files. Migrate virtual machine is to move files. And remove virtual machine is to delete files.

3.4.1 Upload VMX and VMDK files

In order to accept files, a directory acting as a container for virtual machine’s files is needed to be created first. vCLI provides command vifs and option mkdir to create directory. Since console and ESX server are separate, so server IP address, username and password are needed to provide.

```
vifs --server 128.39.73.231 --username root --password password --mkdir '[$datastore1] Node'
```

Then VMX and VMDK files are able to be uploaded to Node directory.

```
vifs --server 128.39.73.231 --username root --password password --put /opt/mln/template/Node.vmx '[$datastore1] Node/Node.vmx'
vifs --server 128.39.73.231 --username root --password password --put /opt/mln/template/Node.vmdk '[$datastore1] Node/Node.vmdk'
vifs --server 128.39.73.231 --username root --password password --put /opt/mln/template/Node-flat.vmdk '[$datastore1] Node/Node-flat.vmdk'
```
3.4. DEPLOY VIRTUAL MACHINE ONLY BY TEMPLATE AND VCLI

If they are uploaded successfully, system administrators can find them in Datastore Browser. Or running following command can print file list, which has the same feature as shell command ls.

```bash
vifs --server 128.39.73.231 --username root --password password --dir '/datastore1'] Node'
```

So far files of virtual machine can be operated as normal files. Option put provides upload feature. Other features such as downloading and copying files are also provided by option get and copy.

3.4.2 Register and Power on Virtual Machine

So far the files are ready. Next step is to register a virtual machine Node to make it visible to ESX server before starting it. In ESX server, two layers are divided. One is storage that is usually called Datastore. Another is front called Client. Registering virtual machine is to bring it from storage to front.

```bash
vmware-cmd --server 128.39.73.231 --username root --password password --register /vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/Node/Node.vmx
```

Above command is to register virtual machine to front. 4c74cc2d-f3d16a79-36d5-b8ac6f21f590 is a special value that indicates which disk it stores in. So it can be treated as a constant value, since so far commands are operated merely to one disk. If register() = 1 is returned to console, it means that virtual machine Node is registered successfully. The virtual machine is displayed in the left side of client. But it is in stop status. Next step is to power it on.

```bash
vmware-cmd --server 128.39.73.231 --username root --password password /vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/Node/Node.vmx start hard
```

Both of the two commands above can start virtual machine. First command with hard flag will power on virtual machine immediately. Second command with soft will examine virtual machine before starting it. If everything is going normally, then it will power on virtual machine. Since second command simulates the most real condition of starting physical machine, so it is selected in this project. If start() = 1 is returned, starting virtual machine Node is successful. Then system administrators can check its status by executing following command.

```bash
vmware-cmd --server 128.39.73.231 --username root --password password /vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/Node/Node.vmx getstate
```

getstate can return virtual machine status. It is necessary to check its status to guarantee that it is running before next step. If getstate() = on is printed, it indicates virtual machine Node is running fluently.

3.4.3 Power off Virtual Machine and Cancel Register

The vCLI also provides two ways to shut virtual machine down, showed below. One is with flag hard. Another is with flag soft. Stopping virtual machine
3.5. INVESTIGATE MLN SOURCE CODE

with flag \textit{hard} will shut it down immediately and forcibly, like pressing power button long. However, if use flag \textit{soft}, it will perform normally to shut virtual machine down.

\begin{verbatim}
vmware-cmd --server 128.39.73.231 --username root --password password /vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/Node/Node.vmx stop hard
vmware-cmd --server 128.39.73.231 --username root --password password /vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/Node/Node.vmx stop soft
\end{verbatim}

In this project, flag \textit{soft} is chosen to simulate the truest way to stop virtual machine. After executing the command, if \texttt{stop()} = 1 is returned, then stopping virtual machine Node is successful.

Although virtual machine Node is shut down, it is still showed on the left side of client. In this project client is designed to only show running virtual machines. Stopped virtual machines should be invisible in the client so that it should be cancelled register. Command below is used to cancel Node’s register.

\begin{verbatim}
vmware-cmd --server 128.39.73.231 --username root --password password unregister /vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/Node/Node.vmx
\end{verbatim}

If \texttt{unregister()} =1 is returned to console, then register of virtual machine is cancelled successfully. If system administrator goes back to client, virtual machine Node becomes invisible there. But it is invisible in the Datastore.

3.4.4 Remove Virtual Machine

When virtual machine is obsolete, it needs to be either upgraded or removed. vCLI provides no methods to upgrade virtual machine. However, vCLI provides an option \texttt{rm} to delete a file or directory. And a directory indicates a virtual machine. So virtual machine is able to be removed. When a file or directory is removed, prompt will pop out to confirm. If \texttt{--force} is added, file or directory will be removed without interrupted by prompt.

\begin{verbatim}
vifs --force --server 128.39.73.231 --username root --password password --rm '[datastore1] Node'
\end{verbatim}

Above command is to remove directory of virtual machine Node without prompt. Supposing ten virtual machines needs to be remove at one time, so ten prompts are not necessary to verify. Therefore option \texttt{--force} is added in command. If output \texttt{Deleted file '[datastore1] Node' successfully} is printed, Node is removed successfully.

3.5 Investigate MLN Source Code

MLN (Manage Large Network) source code is investigated to understand how to build, start, stop, check status and remove network. For example, when network is built by MLN, no matter where the network is located, it will be built locally at the beginning. If it is a local network, then MLN has finished the
task. If it is a remote network, then MLN will transfer it to remote end. For network starting and stopping, MLN will call bash script created to manipulate network. Thus two bash scripts for virtual machine or virtual switch are created during building process. One script is to start virtual machine. Another script is to stop virtual machine. Secondly, windows to call plug-in sub functions are investigated and recorded as well. All windows are selected and corresponding feature is examined. For example, createFilesystem() is an important function called before mount function to create virtual disk and operating system. Thirdly, since plug-in vmware.pl is developed for VMware former platform called VMware Server 1.0, so it is also investigated. And it is examined carefully to realize what kinds of basic features are necessary for the new generation of VMware ESX platform.

MLN is a virtual machine and networks administration tool. Originally it is designed to administrate virtual machine merely on Xen and UML platforms. As concept of plug-in is imported to MLN, plug-in extends MLN’s feature to administrate virtual machines and switches on different platform. For example, ec2.pl is a plug-in for MLN to manage virtual machines on the Amazon EC2 cloud computing platform. Building virtual machines is supported. Starting, stopping and removing virtual machines are supported as well. However, virtual machine upgrade is not supported. Therefore supposing to use MLN to administrate virtual machines and switches on VMware ESX platform, developing a plug-in is feasible.

3.6 Principle of Developing New Plug-in esx.pl

After investigation, to develop a new plug-in esx.pl for VMware vSphere ESX platform is ready. The plug-in is called esx.pl since new generational VMware platform is called ESX. General principles to develop the plug-in is stated here. First of all, since some basic features are implemented by some basic functions, they can be developed at the beginning. For example, function mountFilesystem is responsible for mounting created file system before MLN modifies it, such as altering IP address or adding user. Therefore, it can be developed since it is a basic verified feature. Basic functions are selected and listed in a table below. In this table, ten basic functions are listed. And features are explained from the second column. Next, vCLI commands are selected and inserted into corresponding function. For example, function createStartStopScripts requires vCLI command to start a virtual machine. Therefore, start command should be inserted into the function. Another example, function mountFilesystem needs commands to mount file system from VDDK so that it should be added into the function as well. In a word, selecting proper commands into corresponding function is complicated because it is required to be familiar with both MLN and vCLI. Thus, it is the reason why these two parts are investigated before starting to develop the plug-in. Do the same procedures to fill vCLI and VDDK commands into other functions until all basic functions are fulfilled.
### 3.7 Advanced Feature: Network Upgrade

Basic functions are responsible for starting, stopping, checking status and removing virtual machines or virtual switches. However, more modifications in basic function and more functions are added into plug-in when upgrade and migrate features are developed, which is the most complicated section in this project. When executing MLN command, no matter whether its option is build or upgrade, MLN will call the same function `createFilesystem` to do it. Thus many conditional statements and flags are required to add into this function to distinguish current status whether it is in build or in upgrade. Next, according to the status, plug-in `esx.pl` will perform different action to finish its task. Network upgrade is an advanced feature in virtual machines and networks administration. Goal of upgrade is to alter attributes of virtual machines, increase/decrease number of virtual machines/virtual switches to alter network topology without rebuilding network.

Steps to fulfil this goal are stated below.

- Add and modify source code of `esx.pl` plug-in, but do not modify MLN source code as far as possible
- Do thorough testing including build, start, stop, check status, remove and upgrade to make sure each feature works
- Repeat above two steps until all upgrade features are fulfilled

Supposing modifying `esx.pl` source code merely can not fulfil all upgrade features, then modifying MLN source code is considered.

### 3.8 ESX Keyword Implemented in MLN Project File

If virtual machines and switches are deployed on Xen platform, keyword `xen` is necessary to be marked in MLN project file to make MLN realized that this
3.9. USE MLN AND PLUG-IN ESX.PL TO ADMINISTRATE VMWARE ESX PLATFORM

network will be constructed on xen platform. Thus if network is built on VMware ESX platform, keyword esx should be provided. Since new generational VMware virtualization platform ESX shifts console out of hypervisor, and running MLN is merely feasible in console, so network is always deployed in remote end, which means only one esx keyword is not enough. More information is required by MLN. So solution is to extend esx keyword from just a single keyword to a block keyword where more information is fulfilled.

A list below is to demonstrate how the esx block is designed.

```
esx {
  datastore datastore1
  vc_server 128.39.73.231
  username root
  password password
  vmfslabel 4c74ec2d-f3d16a79-36d5-b8ac6f21590
  vc_uuid 52 46 f5 2e 65 33 35 04-e1 0e bf f5 aa 6c 8f 13
}
```

This is an esx block filled with enough information for MLN to administrate virtual machines and switches on VMware ESX platform. First value is datastore indicating which disk is used. Next three values are IP address, user name and password used to access VMware ESX server. Next value is a global value to indicate which volume of the disk is used to store virtual machines and switches. Last value is an unique vCenter identifier required in VMX file. Above six values are critical for MLN to perform administration of virtual machines and switches on VMware ESX platform.

3.9 Use MLN and Plug-in esx.pl to Administrate VMware ESX Platform

In order to solve the problem stated in the introduction, two networks are designed, created and administrated by MLN and plug-in esx.pl to illustrate that new tool brings low deployment complexity, high automatic administration and high coherence among virtual machines. First network is simple network with project name simpleTopology. Three virtual machines and one virtual switch are involved in the network. It is demonstrated to system administrators to make them realized how to use MLN and plug-in to administrate virtual machines and switches on VMware ESX platform. Moreover simplicity of using MLN is expressed as well. Second network is complex network with project name firewallTopology. Six virtual machines and three virtual switches are involved in the network. This network is demonstrated to show how MLN is able to administrate complex network with low deployment efforts, high automation administration degree and high coherence among virtual machines.

3.9.1 Simple Network

Detailed scheme to administrate simple network is stated in this section. First step is to write a MLN project file describing network topology. This net-
work contains both virtual machine and virtual switch. Although it is a simple topology, it is ample to demonstrate how to use MLN to administrate network. Second step is to build whole network by executing following command.

```
mln build -f simpleTopology.mln
```

It will print a lot of output. Some are from MLN, others are from plug-in esx.pl. In order to distinguish where output comes from, label ESX: is added in front of each output from plug-in. From output, whether network is built successfully or not is able to be judged. Next step is to start whole network or start only one virtual machine.

```
mln start -p simpleTopology
mln start -p simpleTopology -h vm
```

First command is to start the whole network and second command is to start only one virtual machine called vm. Next it is necessary to check its status to verify that it is running by executing following command.

```
mln status -p simpleTopology
mln status -p simpleTopology -h vm
```

MLN provides two ways to check status. One is to check whole network’s status by executing the first command. Another is to check one single virtual machine’s status by executing the second command. Up or down is printed to indicate virtual machine’s status. Supposing starting them is fluent, then system administrator logs into virtual machine to check its attributes, such as disk size, memory size, DNS location, network configuration and users. Next step is to shut them down.

```
mln stop -p simpleTopology
mln stop -p simpleTopology -h vm
```

Two ways to stop either whole network or one virtual machine are provided. Friendly output will be printed to indicate whether they are shut down completely or not. Next it is advised to run `mln status -p simpleTopology` command again to check its status. If they are all down, removing whole network is executed by following.

```
mln remove -p simpleTopology
```

A copy of network stored in local will be removed first. Next virtual machine stored in remote ESX server will be removed. Finally, virtual switch stored in remote ESX server will be removed. If everything goes well, it illustrates that using MLN can administrate virtual machines and networks on VMware ESX platform.

### 3.9.2 Complex Network

Second network called `firewallTopology` is much more complicated than the first network. Process of building and managing the second network is similar to the first one. Following commands are executed to build, start, stop, check and remove whole network. Management operations to single virtual machine is
also supported, but they are the same as demonstrated in last section. Thus they are omitted here.

```bash
mln build -f firewallTopology.mln
mln start -p firewallTopology
mln stop -p firewallTopology
mln status -p firewallTopology
mln remove -p firewallTopology
```

If this case is fulfilled, it illustrates that MLN with esx.pl plug-in is enable to create and administrate big and complex network.

3.10 User Interface

User interface is intended to develop to make virtual machine and switch deployment and administration more convenient. It is designed in PHP. Figure 3.4 below displays the whole procedure operated by user interface.

![Design diagram of combining user interface with MLN and plug-in](image)

First, configure information, such as ESX server IP address, username and password is filled into text field of PHP web page. After submitting web page, configure information is filled into MLN project file. Next, MLN command is sent to a daemon that is running by root locally. The daemon will receive MLN command and “OK” will be replied to user interface. After that, machine Manager will execute MLN command. Output generated from MLN and plug-in esx.pl will be saved in `/tmp/TopologyName`. Finally, the output will be captured and printed in real time by user interface.
3.11 Data and Analysis

So as to certify that using MLN with esx.pl plug-in can decrease deployment complexity, increase automatic administration and coherence to administration of virtual machines and networks, some data is collected by different sorts [54].

Sorts of data is collected and divided into five groups.

First group:

- Number of VMware commands to build and administrate simpleTopology
- Number of MLN commands to build and administrate simpleTopology
- Number of VMware commands to build and administrate firewallTopology
- Number of MLN commands to build and administrate firewallTopology

For system administrator, few commands to fulfil same task is better than more commands, since more commands might increase probability of error. Meanwhile, few commands can bring higher administration automation. Thus number of VMware commands and MLN commands are collected and compared to achieve which method is better on two topologies.

Second group:

- Lines of code describing simpleTopology in MLN project file
- Wizard clicks of building simpleTopology
- Lines of code describing firewallTopology in MLN project file
- Wizard clicks of building firewallTopology

Lines of code in MLN project file and wizard clicks are collected and compared to obtain which method is more complicated. Supposing lines of MLN code is fewer than number of wizard clicks, it means that former method is able to decrease network deployment complexity.

In MLN project file, since hierarchy of virtual machine attributes is introduced and connection between network interface and virtual switch is visible, so coherence among virtual machines and virtual switches is achieved. However, coherence among virtual machines and virtual switches is invisible from wizard clicks.

Third group:

- Number of VMware commands to start simpleTopology
3.11. DATA AND ANALYSIS

- Number of MLN commands to start *simpleTopology*
- Number of VMware commands to start *firewallTopology*
- Number of MLN commands to start *firewallTopology*

Fourth group:
- Number of VMware commands to stop *simpleTopology*
- Number of MLN commands to stop *simpleTopology*
- Number of VMware commands to stop *firewallTopology*
- Number of MLN commands to stop *firewallTopology*

Fifth group:
- Number of VMware commands to remove *simpleTopology*
- Number of MLN commands to remove *simpleTopology*
- Number of VMware commands to remove *firewallTopology*
- Number of MLN commands to remove *firewallTopology*

Under the same condition, each group of data is analysed and compared to find optimal solution. First group, Number of VMware and MLN commands are compared to explore whether administration automation is increased. Second group, code lines and wizard clicks are compared to achieve whether deployment complexity is decreased and coherence among virtual machines and virtual switches is introduced or not. Other three groups, from third to fifth group, number of VMware and MLN commands to administrate network is gathered and compared as well to explore whether they are able to lead to high degree of automatic administration.
Chapter 4

Result One: MLN Analysis and Plug-in Design

This is the first chapter of results. In this chapter MLN source code and call procedure are documented. Plug-in design spirit and essential code are explained.

4.1 Analysis of MLN Functions and Operations

There are 231 sub functions in MLN that are divided into two parts. One part is called non-unique sub functions, because during its execution it may call another sub function from a plug-in to finish its process. Another is called unique sub function, since it is able to accomplish the task itself. Most sub functions can finish their own functions by itself. However 18 sub functions are found that they may call other sub functions from a plug-in to help themselves. So these 18 sub functions are so exclusive that they should be picked out to show the detailed information in table.

The table are comprised of two columns. The first column is the name of the sub function from MLN. The second column is the name of sub function from plug-ins called from MLN. For example, when MLN wants to configure the file system, it will call not only the sub function configureMLNFilesystem, but also call the sub function pluginName_configureMLNFilesystem from a plug-in. Secondly, some sub functions of MLN may need to call more than one sub functions of plug-ins. For example, the sub function buildSwitch may call both _buildSwitch and _configureSwitch when MLN is ordered to build switches. Thirdly, for sub function runPlugins, it will call a sub function of a plug-in called pluginName_$[0], and the $[0] presents the first parameter of sub function runPlugins. Fourthly, the pluginName_$name will be called when MLN invokes sub function executePlugin or executePluginExclusive. And the $name refers to the parameter incomingLiveVM.

When one runs this command, mln build -f projectFile.mln, MLN will build a network. And it will call many sub functions to fulfil the task. And these sub
4.1. ANALYSIS OF MLN FUNCTIONS AND OPERATIONS

<table>
<thead>
<tr>
<th>Name of Sub Function in MLN</th>
<th>Name of Sub Call in Plug-ins</th>
</tr>
</thead>
<tbody>
<tr>
<td>configureMLNFilesystem</td>
<td>_configureEntireFilesystem</td>
</tr>
<tr>
<td>writeXenConfig</td>
<td>_getFilesystemPath</td>
</tr>
<tr>
<td>buildSwitch</td>
<td>_buildSwitch_configureSwitch</td>
</tr>
<tr>
<td>buildHostFilesystem</td>
<td>_configure</td>
</tr>
<tr>
<td>writeConfig</td>
<td>_version</td>
</tr>
<tr>
<td>removeHost</td>
<td>_removeHost</td>
</tr>
<tr>
<td>removeProject</td>
<td>_removeHost</td>
</tr>
<tr>
<td>checkIfUp</td>
<td>_checkIfUp</td>
</tr>
<tr>
<td>upgradeMLNHosts</td>
<td>_liveUpgrade</td>
</tr>
<tr>
<td>removeOldHosts</td>
<td>_removeHost</td>
</tr>
<tr>
<td>getStatus</td>
<td>_getDiskResource _checkIfSwitchIsUp</td>
</tr>
<tr>
<td>runPlugins</td>
<td>_$(0)</td>
</tr>
<tr>
<td>runPostParsePlugins</td>
<td>_postParse</td>
</tr>
<tr>
<td>getFilesystems</td>
<td></td>
</tr>
<tr>
<td>runDaemon</td>
<td>_getImportExportFiles</td>
</tr>
<tr>
<td>getImagePath</td>
<td>_getFilesystemPath</td>
</tr>
<tr>
<td>executePlugin</td>
<td>_$name</td>
</tr>
<tr>
<td>executePluginExclusive</td>
<td></td>
</tr>
<tr>
<td>importMLNproject</td>
<td>_importFilesystem</td>
</tr>
<tr>
<td>exportMLNproject</td>
<td>_exportFilesystem</td>
</tr>
</tbody>
</table>

Table 4.1: Characteristics of sub functions of MLN that calls plug-ins

functions are drawn in a time line diagram in UML (Unified Modelling Language) showed below.

From the chart, the process of building a network is very complex. Firstly, MLN reads and parses the configuration file. Then if it finds all the file system images and plug-ins MLN needs, it will build the MLN project’s handler. The handler function can not finish the task itself. It has to call other sub function buildMLNproject() and some other sub functions that mainly include buildNetwork() and buildMLNHosts() to construct and configure switches and file systems. After finishing these procedures, MLN goes back to its main function and creates some scripts to manage the network. Each virtual machine has two Bash scripts, start_hostName.sh and stop_hostName.sh to start or stop virtual machine. The same as virtual switch, it also has two Bash scripts, start_vSwitchName.sh and stop_vSwitchName.sh to manipulate vSwitch.

After building a project successfully system administrators can run mln start -p projectName to start the whole project. It runs start_vSwitchName.sh first to set vSwitch up and then run start_hostName.sh to boot virtual machines. If only a virtual machine is expected to boot, system administrators can run mln start -p projectName -h hostname to boot that virtual machine exclusively and keep the vSwitch down. Corresponding to the start commands, there are two kinds of stop commands for system administrators. One is mln stop -p projectName that
4.1. ANALYSIS OF MLN FUNCTIONS AND OPERATIONS

Figure 4.1: Sub function sequence diagram of MLN when building network
4.1. ANALYSIS OF MLN FUNCTIONS AND OPERATIONS

stops the virtual machines first and then shuts the vSwitch down. Another is 
mln stop -p projectName -h hostName which stops the specific virtual machine only but keep the vSwitch running.

At any time, system administrators can run mln status to check all the projects’ status or add -p projectName to check the specific project’s status. Moreover the command option -h hostName can be added to check the virtual machine’s status. Next picture figure 2.6 states how mln status works.

![Diagram](image)

Figure 4.2: MLN’s sub function sequence diagram when running mln status

From figure 4.2 the starting point is mln status. It calls printStatus() that calls getStatus() to get the status of virtual machines and virtual switches. Then a temporary data set is built by buildDataTree() to storage conditions of virtual machines and virtual switches. Then two sub functions checkIfUp() and checkIfSwitchIsUp() are called to check their status. Later it will decide whether there exists a distributed project or not. If it exists, MLN will connect to the remote server and require the project’s status. If it does not exist, it will return to the main function. Finally it reports and prints the status to the console.

When a network or some parts of the network needs to be altered, MLN provides upgrade command to support this task. The detailed steps are to modify the project file first, then run mln upgrade -p projectName. MLN will parse the new project file and pick the updated part out to replace the old version.

Finally, if a network is prepared to be erased, it is simple to run mln remove -p
4.2 Plug-in Design

The essential parts of the code and its design spirit are stated.

4.2.1 Common Variables

For the first part of the plug-in, some default variables are defined to prevent user to forget to define them and to make the plug-in robust.

```plaintext
# Tested on VMware vSphere ESX 4.1.0
my $ESX_PLUGIN_VERSION = 0.1;
my $DEFAULT_DATASTORE = "datastore1";
my $DEFAULT_VC_SERVER = "128.39.73.231";
my $DEFAULT_ESX_HOST_USERNAME = "root";
my $DEFAULT_ESX_HOST_PASSWORD = "corenetwork";
my $DEFAULT_VMFSLABEL = "4c74cc2d-f3d16a79-36d5-b8ac6f21f590";
my $DEFAULT_VC_UUID = "52 46 f5 2e 65 33 35 04-e1 0e bf f5 aa 6c 8f 13";
my $DEFAULT_PORT_GROUP_NAME = "VM Network";
```

The plug-in is tested on the VMware vSphere ESX 4.1.0 platform. According to the programming standard, the $ESX_PLUGIN_VERSION is declared. Next seven important variables are defined and their roles are stated in the table below.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_DATASTORE</td>
<td>The default storage volume in which virtual machines are resided</td>
</tr>
<tr>
<td>DEFAULT_VC_SERVER</td>
<td>The IP address of the default storage server</td>
</tr>
<tr>
<td>DEFAULT_ESX_HOST_USERNAME</td>
<td>The default administrator used in the ESX is root</td>
</tr>
<tr>
<td>DEFAULT_ESX_HOST_PASSWORD</td>
<td>The password used for the default ESX server</td>
</tr>
<tr>
<td>DEFAULT_VMFSLABEL</td>
<td>The volume label referred to the default storage server</td>
</tr>
<tr>
<td>DEFAULT_VC_UUID</td>
<td>The default UUID of the vCenter system</td>
</tr>
<tr>
<td>DEFAULT_PORT_GROUP_NAME</td>
<td>The default subnet where virtual machines are resided</td>
</tr>
</tbody>
</table>

Table 4.2: Variables designed and roles played in the plug-in

Each of the seven variables will be used when they are missed or defined in the MLN project file.

4.2.2 Create File System

This function `esx_createFilesystem()` will create the operating system and adjust or expand the disk size.
4.2. PLUG-IN DESIGN

Get Variables

```perl
my $DATASTORE = getScalar("/host/$hostname/esx/datastore");
$DATASTORE = $DEFAULT unless $DATASTORE;
out("DATASTORE: $DATASTORE\n");
my $VC_SERVER = getScalar("/host/$hostname/esx/vcserver");
$VC_SERVER = $DEFAULT unless $VC_SERVER;
out("VC_SERVER: $VC_SERVER\n");
my $ESX_HOST_USERNAME = getScalar("/host/$hostname/esx/username");
$ESX_HOST_USERNAME = $DEFAULT unless $ESX_HOST_USERNAME;
out("ESX_HOST_USERNAME: $ESX_HOST_USERNAME\n");
my $ESX_HOST_PASSWORD = getScalar("/host/$hostname/esx/password");
$ESX_HOST_PASSWORD = $DEFAULT unless $ESX_HOST_PASSWORD;
out("ESX_HOST_PASSWORD: $ESX_HOST_PASSWORD\n");
```

This piece of code will capture some necessary variables before creating the operating system. The `getScalar()` is a very useful function that it gets the value from the built data tree. And if the value is empty, the default value will be assigned. Next the `out()` function prints the value to the console which helps system administrators to monitor the building process.

Convert or Expand Disk

This part of code shows how to convert or expand the file system image and some trivial codes like variable assignments or transformations are omitted.

First piece of the code is used to achieve the size of the file system template. It is achieved by a long regular express. Regular express provides a concise and flexible means to match a text or a string. The source text of the template is like this.

```plaintext
−rw−r−r— 1 root root 2147483648 Mar 8 10:18 debian6.2GB−flat.vmdk
```

From the text, the size of the template, `2147483648` is going to pick out. So a regular express (line 3) is written to match it and extracted from the fourth brackets which is assigned to the variable `TemplateSize`.

```perl
while( my $templateSizeInfo = <$TEMPLATESIZE> ){
  chomp $templateSizeInfo;
  if($templateSizeInfo =~ /'(\S+)'s1'(\w+)'s'(\w+)'s'(\d+)\s'(\s)'s'(\s+)−flat
  \s\s.*/ ){
    $TemplateSize = $4;
  }
}
```

Next piece of the core code is used to convert (line 1) and expand (line 4) disk. If the `freeSpace` is declared and assigned a value that is not zero, the `newDiskSize` is recomputed and the disk is expanded.

```perl
system("vmware–vdiskmanager −r $TEMPLATEDIR/$tem
−t 4 $PROJECT_PATH/$PROJECT/images/$hostname.vmdk");
if( $freeSpace != 0 ){
  $newDiskSize = $newDiskSize + $newFreeSpace;
  system("vmware–vdiskmanager −x $newDiskSize $PROJECT_PATH/$PROJECT/images/$hostname.vmdk");
} else{
  verbose("freeSpace: $freeSpace\n");
}
```
4.2. PLUG-IN DESIGN

If the freeSpace is zero, the disk will not be resized. It prints the amount of free space by verbose() function instead. Besides there are another two critical variables in the process of resizing the disk. One is the sizeDeclaredFromScript which is the disk size declared from the MLN project file. Another is the GBTemplateSize which is the size of the file system template presented in GB unit. In order to make the disk creation robust, three conditions are considered by comparing the size of GBTemplateSize and sizeDeclaredFromScript. This part of code is omitted but can be found from the Appendix C.

4.2.3 Mount VMDK File System on the Linux

```perl
sub esx_mountFilesystem {
  my $hostname = $_.[0];
  if( getScalar("/host/$hostname/esx"){
    system("vmware-mount $PROJECT PATH/$PROJECT/images/$hostname.vmdk $MOUNTDIR");
    system("df -h");
    return 1;
  }
}
```

This piece of code is implemented to mount a VMDK file system on the Linux machine. Since MLN may do modification to the file system, for example, adding a user, the VMDK has to be mounted during the building process. First the host name is got and one checks whether the esx block is existed or not. If it existed, the VMDK will be mounted. Finally, the df -h is run to check whether the VMDK is mounted successfully or not.

If it is mounted smoothly, the line should be showed below that is mounted under the .mln_mountdir directory.

```
/dev/loop0 1.9G 1016M 791M 57% /root/.mln_mountdir
```

4.2.4 Umount VMDK File System on the Linux

```perl
sub esxUnmountFilesystem {
  my $hostname = $_.[0];
  if( getScalar("/host/$hostname/esx"){
    out("ESX plugin is unmounting $hostname on $MOUNTDIR")
    system("vmware-mount -d $MOUNTDIR");
    system("df -h");
    return 1;
  }
}
```

Before importing the operating system to VMware ESX server, MLN needs to umount it. Line 5 is used by MLN to umount a operating system.

4.2.5 Create Start and Stop Script for Virtual Machine

MLN uses Bash code to manipulate virtual machines. So Bash codes that insert into Perl code are generated during the building process.
4.2. PLUG-IN DESIGN

```bash
out("creating : $PROJECT/stop \$\{bo\}_hostname.sh\")
open(STOP, ";$PROJECT/stop \$\{bo\}_hostname.sh\")
print STOP "#!/bin/bash\n"
print STOP "echo \"Stopping hostname...\" \n"
print STOP "vmware-cmd --server SVC_SERVER --username $ESX_HOST_USERNAME -- 
password $ESX_HOST_PASSWORD --s register /vmfs/volumes/$VMFSLABEL/Hostname . $PROJECT/vmx > $PROJECT/stop \$\{bo\}_hostname.sh \$
VMREGISTERSTATUS\n"
# Check the virtual machine has been registered or not
print STOP "if [ \"\"cat $PROJECT/stop/Hostname.$PROJECT . VMREGISTERSTATUS \"] ; then\n"
print STOP "echo \"The virtual machine $hostname . $PROJECT has been started , abort...\" \n"
print STOP "exit 1\n"
print STOP "fi\n"
# Make sure it is registered
print STOP "while [ \"\"cat $PROJECT/stop/Hostname.$PROJECT . VMREGISTERSTATUS \"] do\n"
print STOP "vmware-cmd --server SVC_SERVER --username $ESX_HOST_USERNAME -- 
password $ESX_HOST_PASSWORD --s register /vmfs/volumes/$VMFSLABEL/Hostname . $PROJECT/vmx > $PROJECT/stop/Hostname.$PROJECT . $PROJECT . VMLIVE
VMREGISTERSTATUS\n"
print STOP "echo \"Register the virtual machine $hostname . $PROJECT again\"\n"
print STOP "done\n"
# Start the virtual machine
print STOP "vmware-cmd --server SVC_SERVER --username $ESX_HOST_USERNAME -- 
password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/Hostname.$PROJECT/Hostname.vmx start soft\n"
close (STOP);
system("chmod +x $PROJECT/stop/\$\{bo\}_hostname.sh\")
```

Above it is the code to register and start a virtual machine. Line 5 is going to register a virtual machine and print its output to a file VMREGISTERSTATUS. If the virtual machine has been registered before, next a if block can check, alert and quit the program safely. If the virtual machine has not been registered fluently, a while block can guarantee to register it. Next, line 17 starts the virtual machine softly which usually takes some time, but since it is the last line in the Bash code, it makes no effects to other codes. Finally, chmod command is used to give the execution permission to the file. From here, many conditions are considered to make sure the register and start process robustly.

```bash
out("creating : $PROJECT/start \$\{bo\}_hostname.sh\")
open(START, ";$PROJECT/start \$\{bo\}_hostname.sh\")
print START "#!/bin/bash\n"
print START "echo \"Starting $hostname...\" \n"
print START "vmware-cmd --server SVC_SERVER --username $ESX_HOST_USERNAME -- 
password $ESX_HOST_PASSWORD --s register /vmfs/volumes/$VMFSLABEL/Hostname . $PROJECT/vmx > $PROJECT/start \$\{bo\}_hostname.sh \$
VMREGISTERSTATUS\n"
# Check the virtual machine has been registered or not
print START "if [ \"\"cat $PROJECT/start/Hostname.$PROJECT . VMREGISTERSTATUS \"] ; then\n"
print START "echo \"The virtual machine $hostname . $PROJECT has been stopped , abort...\" \n"
print START "exit 1\n"
print START "fi\n"
# Get status of virtual machine
print START "vmware-cmd --server SVC_SERVER --username $ESX_HOST_USERNAME -- 
password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/Hostname.$PROJECT/Hostname.vmx getstate > $PROJECT/start/Hostname.$PROJECT.VMLIVE
"
print START "if [ \"\"cat $PROJECT/start/Hostname.$PROJECT.VMLIVE \"] = \n\"\"
]\ ; then\n"
print START "echo \"The virtual machine $hostname . $PROJECT has been stopped, abort...\" \n"
print START "exit 1\n"
print START "fi\n"
```

44
4.2. PLUG-IN DESIGN

4.2.6 Generate VMX Configuration File

In VMware ESX platform, a VMX file illustrates the hardware of a virtual machine. It can describe the hostname, memory, firmware, CPU, swap, disk, CD-ROM, PCI devices and network. The code in the plug-in are divided into three parts shown below. The first part is the hardware description to a guest virtual machine. The second part is about the network setting. And the third part is about the file transfer.

```java
open(VMX, ">$PROJECT_PATH/$PROJECT/hostname.vmx");
print VM " . encoding = "UTF-8" \n";

# Virtual Machine Communication Interface (VMCI)
print VM "vmci0.present = " \n"
print VM "vmci0.id = " \n"
print VM "vmci0.pciSlotNumber = " \n"

# Guest Information
print VM "displayName = " \n"
print VM "extendedConfigFile = " \n"
print VM "nvram = " \n"
print VM "memsize = " \n"
print VM "sched.swap.derivedName = " \n"
print VM "guestOS = " \n"
print VM "uuid.location = " \n"
print VM "uuid.bios = " \n"
print VM "vc.uuid = " \n"

# Hardware Virtualization
print VM "virtualHW.version = " \n"
print VM "virtualHW.productCompatibility = " \n"

# Power
print VM "powerType.powerOff = " \n"
print VM "powerType.powerOn = " \n"
print VM "powerType.suspend = " \n"
print VM "powerType.reset = " \n"

# Storage Devices
print VM "floppy0.present = " \n"
```

# Make sure it is stopped
print STOP "while [ " \n" cat $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMLIVE \n" ! -f " \n" getstate () = off" ] ; do
print STOP "echo \" Stopping the virtual machine $hostname.$PROJECT again\" \n"
print STOP "vmware-cmd --server $VC_SERVER --username $ESX_HOST_USERNAME -- 
password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/hostname.$PROJECT/ 
hostname.vmx stop soft \n";

# The stop takes some time
print STOP "sleep 10s\n"
print STOP "vmware-cmd --server $VC_SERVER --username $ESX_HOST_USERNAME -- 
password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/hostname.$PROJECT/ 
hostname.vmx getstate > $PROJECT_PATH/$PROJECT/hostname.$PROJECT.VMLIVE \n"
print STOP "done\n";

# Unregister the virtual machine
print STOP "vmware-cmd --server $VC_SERVER --username $ESX_HOST_USERNAME -- 
password $ESX_HOST_PASSWORD -- unregister /vmfs/volumes/$VMFSLABEL/ 
hostname.$PROJECT/hostname.vmx \n"

close (STOP);

system ("chmod +x $PROJECT_PATH/$PROJECT/stop.sh hostname.sh");

This is the VMX configuration file above. Since it is so long that some duplicate parameters are omitted, but the whole file can be found in Appendix C. The whole file is encoded in UTF-8. First it is the virtual machine communication interface (VMCI) which is used to communicate between a virtual machine and the host operating system or among virtual machines. If the VMCI is not in use, virtual machines can use network layer of TCP/IP protocol to communicate which adds more overhead.

Next part of the VMX file is the fundamental information for a virtual machine. The machine name, size of memory, NVRAM, name of the swap file, operating system type, UUID (Universal Unique Identifier), PCI bridge, floppy, CD-ROM and SCSI disk can be modified here. This profile can be treated as a machine template that can be modified to create different machine.

The next part is the network configuration. A virtual machine is deployed into the correct location according the MLN file and topology.
First, all network interfaces are achieved by function `getHash()` and assigned to a hash table `interfaces`. Next the interface is picked out one by one and connected to the corresponding subnet presented by `PORT_GROUP_NAME`. The `PORT_GROUP_NAME` is the subnet of a virtual switch. If no virtual switch is referred in the MLN project file, the virtual machine will be connected to the default network, `VM Network`. Otherwise it will connect to the virtual switch mentioned in the MLN project file. The MAC address is left empty and generated automatically by VMware ESX. The network type is `e1000` referred to the 1000 Mbps network adapter.

Then the next part is three commands to transfer critical files to VMware ESX server. The function `out()` can prints friendly and useful message to the console.

In the end of the above code, line 13 and 14 are commented that are used to register the virtual machine. If only the running virtual machines are expected to show on the networking view of VMware ESX server, line 13 and 14 should be kept commented. Otherwise, VMware ESX will show all the created virtual machines on the networking view.
4.2. PLUG-IN DESIGN

4.2.7 Check Virtual Machine Status

This piece of code is used to check a virtual machine that is up or down. On the VMware ESX platform, before power on a virtual machine, it needs to be registered first. In order to display running virtual machines merely on the network view of VMware ESX, they should not be registered until they are powered on. So in the virtualMachineCheckUp function below, first the register should be checked. Second one should check the machine is powered or not.

```perl
# The registerFlag, 0 means a VM down and 1 means a VM up.
my $registerFlag = 0;

# Get register list
my $command = "vmware-cmd --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --list";
open(REGISTER, "$command |") or die "Error: failed to open register data... $!

# The first line is a newline. Ignore it.

while ($line = <REGISTER>) {
    chomp $line;
    # /vmfs/volumes/4c74cc2d-f3d16a79-36d5-b8ac6f21f590/ball.m8/ball.vmx
    if ( $line =~ /\vmfs/\volumes/\(/w+)-(\w+)-(\w+)-(\w+)/(\w+)/(\w+).vmx/ ) {
        $virtualMachine=$5;
        $vmxFile=$6;
        if ( ( $virtualMachine eq "$hostname . $project" ) and ( $vmxFile eq "$hostname" ) ) {
            $registerFlag=1;
            verbose("The $hostname . $project has been registered.
"");
        }
    }
}

# Check the VM is start or not
$command = "vmware-cmd --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD /vmfs/volumes /VMFSLABEL/$hostname . $project/$hostname . vmx getstate";
open(RESULT, "$command |") or die "Error in esx host check up... $!

while ( $line = <RESULT> ) {
    chomp $line;
    if ($line =~ /\vmfs/\volumes/\(/w+)-(\w+)-(\w+)-(\w+)/(\w+)/(\w+).vmx/ ) {
        # A host is up
        verbose("A host is up\n");
        return 1;
    } elsif ($line =~ /\!/ ) {
        # A host is down
        verbose("A host is down\n");
        return -1;
    } elsif ( $line =~ /No\svirtual\smachine\sfound\.$/ ) {
        # No virtual machine found.
        verbose("A host is non-existed\n");
        return 256;
    }
}

close(RESULT);
}
else{
    $registerFlag=0;
    verbose("The $virtualMachine has been registered, but not the VM wanted.
");
}
else{
    $registerFlag=0;
    verbose("The $hostname . $project has not been registered. Run \"mln start -p $project\" to start it.\n");
}
```
4.2. PLUG-IN DESIGN

```perl
47 } close ( REGISTER ) ;
48
# Tell MLN the status
49 if ( $registerFlag == 0 ) {
50   verbose ( "Finally, after a search, the $hostname.$project has not been
51             registered. Run \"mln start −p $project\" to start it.\n"");
52   return −1;
53 }
54 }
```

First a flag called registerFlag is set up and assigned to zero which assumes a virtual machine is down initially. Second the register list is obtained by line 5. Then the list is traversed to check each virtual machine’s register status. If a virtual machine is registered, an output representing its existence is similar to the line 11. Then it goes to the next line, a long regular expression to extract the name of virtual machine and VMX file. Next a conditional statement determines this registered machine is hit or not. If it hits, the registerFlag is assigned to one. After that the process goes to the next step to check its status by line 20. From its output, the conditional statement checks this virtual machine’s status and returns a value to MLN. Otherwise the registerFlag is assigned to zero and an alert message is printed. After the traverse if no virtual machines are matched, the registerFlag will be verified again. If it is zero, minus one will return to MLN.

### 4.2.8 Check Virtual Switch Status

```perl
1 my $switch = $_.[0];
2 my $command = "vicfg−vswitch −−server SVC_SERVER −−username $ESX_HOST_USERNAME
−−password $ESX_HOST_PASSWORD −c $switch.$project";
3 open (STATUS, "$command |") or die " Error: Failed to run $command . . . $!\n";
4 while ( my $status = <STATUS> ) {
5   chomp $status;
6   if ( $status =~ /1$/) {
7     # A switch is up
8     return 1;
9   } elsif ( $status =~ /0$/) {
10     # A switch is down
11     return −1;
12   }
13   close (STATUS);
14 }
```

This piece of code is used to check a virtual machine is up or not. Checking a virtual switch is much easier than checking a virtual machine. One receives the name of virtual switch from line one. The $_. is a special array storing parameters from the called function. Line 2 is a critical command to check the virtual switch’s status. Next the process passes through a conditional statement. If the result is one, then the virtual switch is up and value one is returned. Otherwise the value minus one is sent back to MLN and the virtual switch is down.

### 4.2.9 Remove a Virtual Machine

```perl
1 # Check virtual machine status
2 my $value = esx_checkIfUp ( $hostname , $project );
```

49
4.2. PLUG-IN DESIGN

This piece of code is used to remove virtual machines, no matter they are running or stopped. Sometimes due to their carelessness, system administrators may remove virtual machines even though they are running, which may lead a system to crash. So it is a good feature to add to remove running virtual machines.

First the status of a virtual machine is obtained and assigned to value by the first line. If it is down which means the variable value equals to minus one, it will be removed directly by line 15. However if it is up, the process will shift to a while loop until it is shut down. Then the process goes to a conditional statement to remove the host.

4.2.10 Remove a Virtual Switch

This piece of code is used to remove a virtual switch, no matter it is on or off. The structure of this code is quite similar to the code of removing a virtual machine. It checks the status of a virtual switch first, then the process determines whether to remove it or stop it. After removing it, success value one will be sent back to MLN. Otherwise minus one will be returned to MLN. Overall, in order to make the solution robust, the extreme condition should be considered and the solution should be implemented.

4.2.11 Start a Virtual Switch
4.2. PLUG-IN DESIGN

This piece of code is used to create a virtual switch. Line 5 checks the virtual switch existed or not. If it does not exist, line 9 and 11 will create the virtual switch and corresponding subnet. And the MTU (Maximum Transmission Unit) is set to 1500 bytes. Otherwise a error message will be printed and the process will be aborted.

4.2.12 Stop a Virtual Switch

This piece of code is used to stop a virtual switch. Line 5 checks the virtual switch existed or not. If it does not exist, line 9 and 11 will stop the virtual switch and corresponding subnet. And the MTU (Maximum Transmission Unit) is set to 1500 bytes. Otherwise a error message will be printed and the process will be aborted.
This piece of code is used to stop a virtual switch. The structure of this code is similar to the code of starting a virtual switch. Line 5 checks the virtual switch existed or not. If it exists, line 13 and 15 will remove the subnet and the virtual switch, whose deletion sequence should be noticed that it is reverse to the sequence of starting a virtual switch. If a virtual switch does not appear, a error message will be printed and the process will be aborted.

### 4.2.13 Manage Virtual Switches During Upgrade

```plaintext
my %vSwitchList;

# vSwitch flag: 0 means a vSwitch is not built, 1 means it has been built
# Used in the first time, since no OLD_DATA_ROOT in the first time
my $built = "$PROJECT_PATH/$PROJECT/$name.$PROJECT.VSWITCHBUILT";

if ( ! stat "$built"){
    system("echo '0' > $built");
}

open(BUILT,"$built") or die "Error: can’t open $built:\n"
while($line = <BUILT>){
    if($line == 1){
        # The block is a struct, defined in the M&N
        my $rootblock = new block;
        my $switchblock = new block;
        @{$rootblock} = OLD_DATA_ROOT;
        # Shift into switch block
        @{$switchblock} = $rootblock->blocks("switch");
        # Get all the virtual switches from the former project
        foreach $vs (keys @{$switchblock->blocks}){
            # Get all the virtual switch name from the list
            foreach $vs (keys %vSwitchList){
                if($vSwitchList{$vs} == 0){
                    out("Remove $vs.$PROJECT...\n");
                    system("rm -rf $PROJECT_PATH/$PROJECT/start_$vs.sh");
                    system("rm -rf $PROJECT_PATH/$PROJECT/stop_$vs.sh");
                    system("rm -rf $PROJECT_PATH/$PROJECT/$vs.$PROJECT.VSWITCHBUILT");
                    system("rm -rf $PROJECT_PATH/$PROJECT/$vs.$PROJECT.VSWITCHSTATUS");
                }
            }
        }
    }
}
```

52
4.3. CHAPTER SUMMARY

This piece of code is used to record and adjust virtual switches during a system upgrade. Two sets of flags are used to control the existence of virtual switches. One is the file \texttt{name.project.VSWITCHBUILT} that is used in the first time when one creates a virtual switch. And the status number is inserted into this file. After constructing the virtual switch, status number will change to one.

Another flag is a hash table \texttt{vSwitchList} which records each virtual switch’s status. The key is the name of virtual switch and the value is its status. When a system upgrades, the process will get all the former project information from \texttt{OLD_DATA_ROOT} (line 15) and select the switch block which is stored into \texttt{switchblock} (line 17). Next within a loop, each virtual switch is picked out and assigned its initial value to zero. In line 27, when a virtual switch is built, its value is set up to one.

The last part of the code is intelligent. When a virtual switch is erased from a MLN project file, it will be removed by executing this part of code. First the process checks whether the virtual switch is built or not from last time by checking the first flag \texttt{name.project.VSWITCHBUILT}. If it is built last time, one will check whether it exists or not this time by traversing the hash table \texttt{vSwitchList} and examine its value. If the value equals to zero, all files related to this virtual switch will be removed, which represents to remove this virtual switch.

4.3 Chapter Summary

This is the first section of results. Hooks of MLN providing for plug-ins are documented. Two sequence graphs are drawn when executing \texttt{mln build} and \texttt{mln status}. Next the code of the plug-in is sorted according to its functionality and explained in detail.
Chapter 5

Result Two: Automatic Network Administration

This is the second chapter of results. In this chapter, two topologies are deployed and administrated by novel solution.

5.1 Simple Topology

After MLN is installed and templates (vmdk and flat disk files) are prepared, it is time to build a network with MLN. First a simple topology is come to show how to use MLN to implement it.

5.1.1 Network Topology

Before implementing a network, it is critical to have a clear view to the topology. The topology graph should be marked with IP address, mask and gateway. The first topology is simple only with three virtual machines and a virtual switch. The topology is drew by UML and showed as following.

Three virtual machines called Oslo, London and Paris respectively connect to a virtual switch called centerSwitch. The machine Oslo has two ethernet network cards. It is the gateway of the centerSwitch subnet. One card is the eth0 connected to the internet. Another card is the eth1 connect to the subnet. The eth1 is assigned to a public IP address 128.39.73.253 and the eth0 is assigned to a private IP address 10.0.0.1. Other two virtual machines have only one ethernet network card. Both of them are called eth0 and assigned to private IP addresses 10.0.0.2 and 10.0.0.3. Other network configuration information (subnet mask and gateway IP address) can be seen from the figure 5.1.

5.1.2 Design Project File

Next a project file needs to be made to focus on each machine’s hardware and network configuration and sort of operating system. For hardware, the disk size and memory size need to be considered. For network, the IP address,
5.1. SIMPLE TOPOLOGY

Figure 5.1: Simple network topology
5.1. SIMPLE TOPOLOGY

subnet mask and gateway IP address should be considered. Any kind of Linux operating system is supported as soon as VMware ESX supports it. Next it is better to show a project file and then give an explanation.

```
5.1. SIMPLE TOPOLOGY

So this is the project file. First the name of the whole network called *simple-Topology* is defined. Second, the common attributes for a machine that includes operating system, disk size, memory size, DNS server, users and an esx block.
5.1. SIMPLE TOPOLOGY

Since the network is built into an ESX server, so the esx block is necessary. The content of the esx block is constant and how to obtain them is stated in the Approach chapter.

In this case, the stable debian operating system, updated version is used. The disk and memory is given to 2GB and 512MB respectively. The DNS location is assigned to 128.39.89.8. A new users, karl is registered in all machines. And root password is updated to a new one. Both passwords are encrypted.

Next it is a host block referred to a virtual machine called Oslo. First it inherits the common attributes from the superclass. Second it overwrites the memory size to 1024MB. Third, two ethernet network cards are increased with IP address, subnet mask and gateway configurations. The value centerSwitch is filled into the eth0 block which means this interface connects to a switch centerSwitch. While no switch value is filled into the eth1 block which means it connects to the internet.

Next two virtual machines called London and Paris come after. Since they connect to a switch, so only one ethernet network card is needed. They have the same configurations except the IP address. Finally, a switch called centerSwitch is defined in the last line.

5.1.3 Build Network

Now running this command can build this project easily.

```
mln build -f simpleTopology.mln
```

During the building process, one can receive a rich output which is so long that it is stuck in the Appendix A. However it is necessary to show some selective output and give an explanation.

```
  1 ➜ BUILDING simpleTopology
  2 → Switch centerSwitch
  3 ++ printing Superclasses ++
  4 ➜ Superclass commonConfig
  5 commonConfig: template = debian6.2GB.vmdk
  6 ++ printing Hosts ++
  7 → Host Oslo
  8 Oslo: nameserver = '128.39.89.8'
  9 Oslo: superclass = 'commonConfig'
 10 Oslo: memory = '1024M'
 11 Oslo: size = '2GB'
 12 Oslo: template = debian6.2GB.vmdk
 13 printing network
 14 Oslo: Network interfaces
 15 Interface eth1
 16  address: 128.39.73.253
 17  netmask: 255.255.255.0
 18  gateway: 128.39.73.1
 19 Interface eth0
 20  address: 10.0.0.1
 21  netmask: 255.255.255.0
 22  gateway: 10.0.0.1
 23 ### users:
 24 karl KromoKGR69Iqes
```
This is the selective output. First the project file is parsed and all configuration information (switch, superclass and host) is printed. This part of output can be divided into three pieces, host itself, network interfaces and users. The first piece includes the DNS server, inherited class name, memory size, disk size and the name of file system. The second piece is the network. Two network interfaces with IP address, subnet mask and gateway location are printed. The third piece is about the users. Two users are updated into the host’s password file /etc/passwd. Next the configurations are saved into a file simpleTopology.mln.

Secondly, MLN starts to build the network. The switch centerSwitch is constructed first. Then MLN starts to build hosts. The first host is called Oslo. Since a directory represents a host in VMware ESX server, it is built first (line 30 and 31). Second MLN copies and converts the disk file to fit the ESX server. Next the file disk is mounted on the local host to add users and network interfaces. Furthermore the file /etc/inittab is updated. After that, MLN plug-in umounts the file system. Thirdly, two files to manipulate the Oslo host are generated. One is used to start the host. Another one is used to stop the host.
5.1. SIMPLE TOPOLOGY

After building the host, next step is to deploy it to the proper location based on the topology. So the interface ethernet1 connects to the VM Network and the ethernet0 connects to the centerSwitch. Finally, the vmx file, vmdk file and flat file are uploaded to the VMware ESX server. Then the host Oslo is built successfully. Other two hosts, London and Paris follow the same way to be built, but the relevant output is omitted. After all of them are constructed, the project is fulfilled to construct.

5.1.4 Start Network

After the deployment, it is simple to run this command to start the whole network.

```
mln start -p simpleTopology
```

First the switch is powered on and then the hosts are powered on. If a `start()` = 1 is returned, it means booting a host is successful. If all of them are started fluently, the following output will be printed.

```
1 Starting switch centerSwitch.simpleTopology
2 Create virtual switch centerSwitch.simpleTopology...
3 Starting London...
4 start() = 1
5 Starting Oslo...
6 start() = 1
7 Starting Paris...
8 start() = 1
```

Next system administrator can run this command to check the whole network’s status.

```
mln status -p simpleTopology
```

If the whole network is running, a following output will be received.

```
1 simpleTopology host London up
2 simpleTopology host Paris up
3 simpleTopology host Oslo up
4 simpleTopology switch centerSwitch up
```

From the output, system administrators can see that three hosts and one switch are running.

5.1.5 Network View from the VMware ESX Server

The figure 5.2 can be divided into two sections. One section is on the left to show all the virtual machines that abide by host.project name format. On the right side it is a networking view. From this view, two virtual switches can be seen. One is the default virtual switch called vSwitch0 which is used by VMware ESX server itself and connected to the physical adapter vmnic0. Another virtual switch is the centerSwitch created by simpleTopology project. Each virtual switch owns a subnet. The VM Network belongs to the vSwitch0 and the centerSwitch.simpleTopology belongs to the centerSwitch.
5.1. SIMPLE TOPOLOGY

Figure 5.2: Network view from VMware vSphere Client
5.1. SIMPLE TOPOLOGY

On the basis of the topology, host Oslo has a network interface to connect to internet. So it is deployed in the subnet VM Network. Moreover it has another network interface to connect to the centerSwitch. So it is also deployed into the subnet centerSwitch.simpleTopology. Since other two hosts only have one network interface to connect to the centerSwitch, so they are only deployed into the subnet centerSwitch.simpleTopology. Besides, each virtual machine is represented by a small cyan icon and a tiny green triangle on the right side means this virtual machine is running.

5.1.6 Start Only One Host

MLN provides a -h option to start this identified host. For example, if system administrators expect to start host Oslo, this command should be executed.

```
mln start -p simpleTopology -h Oslo
```

And if the following output is received, the host Oslo is confirmed to be powered on.

```
Starting Oslo...
start () = 1
```

5.1.7 Stop Only One Host

MLN provides a -h option to stop an identified host. For example, if system administrators expect to stop host Oslo, this command should be executed.

```
mln stop -p simpleTopology -h Oslo
```

And if the following output is received, the host Oslo is confirmed to be shut down.

```
Stopping Oslo...
stop () = 1
unregister () = 1
```

Now let’s go back to the networking view, the host Oslo disappears on both sides. Only the running hosts, Paris and London are displayed. The ESX plug-in is designed to only display running hosts on the networking view which gives a clear view to system administrators.

5.1.8 Stop Network

By running this command below, stopping the whole network is also simple.

```
mln stop -p simpleTopology
```

If the following output is received, the network is confirmed to be stopped.

```
1 Stopping simpleTopology
2 Stopping Paris...
3 stop () = 1
4 unregister () = 1
5 Stopping Oslo...
6 stop () = 1
```
Figure 5.3: Network view without host Oslo
First step is to shut down all hosts one by one. Second step is to shut down the virtual switch to which no host is connected.

5.1.9 Remove Network

It is also simple to remove the whole network by a single following command.

```
mln remove -p simpleTopology
```

This command removes the network stored locally and deployed remotely in the VMware ESX server. If it is removed successfully, the following output will be obtained.

```
Are you sure you want to remove simpleTopology? (y/N)
y
Removing /opt/mln/projects/simpleTopology
ESX: Remove host London.simpleTopology.
ESX: Remove host Paris.simpleTopology.
Deleted file '[datastore1] London.simpleTopology' successfully.
ESX: Remove host Oslo.simpleTopology.
ESX: Remove host Oslo.simpleTopology.
Deleted file '[datastore1] Oslo.simpleTopology' successfully.
ESX: ESX plugin to remove virtual switch activated...
ESX: Virtual switch centerSwitch.simpleTopology has been removed successfully
```

5.2 Complex Topology

This section is to build and manage a big network on the VMware ESX platform by using MLN and esx plug-in.

5.2.1 Network Topology

This figure shows a complex topology that contains three virtual switches, lan1, lan2 and lan3. Each switch domains a subnet that they are 10.0.0.*, 10.0.1.* and 10.0.2.*. Two firewalls called gateway and choke separate the network into three subnets and internet. Three subnets are called DMZ (Demilitarized Zone) [55, 56], LAN and WLAN respectively.

The gateway is the unique entrance to the DMZ. It has two network interfaces that one connects to internet, another connects to a virtual switch. server1 and server2 represent masses of machines located in the DMZ. The choke is a critical machine that has three network interfaces to detach the subnet into three parts. One subnet called LAN has a virtual switch called lan3 that connects many personal computers. For example, a desktop machine with IP address 10.0.2.2 connects to this switch. Another symmetrical subnet called WLAN has
Figure 5.4: Small business network topology
5.2. COMPLEX TOPOLOGY

the same structure that a switch lan2 connects to choke and many laptops connects to this switch. For example, a laptop with IP address 10.0.1.2 resides in this subnet.

Three subnets can be recognized by three different colours. In the corner of each subnet a yellow label is marked with this domain’s IP address and mask. And beside each machine also a yellow label indicates the interface name, IP address, mask and gateway location. Black lines represent cables connecting each other. Thus this topology is much complex than the former topology in the last section.

5.2.2 Design Project File

Next step is to write a project file. The choke is an exclusive machine so that it is necessary to show how to write it. Other part of the project file is a bit long so that it is attached in the Appendix B.

```mln
1 host choke {
2     superclass commonConfig
3     network eth0 {
4         switch lan1
5         address 10.0.0.2
6         netmask 255.255.255.0
7         gateway 10.0.0.1
8     }
9     network eth1 {
10        switch lan2
11        address 10.0.1.1
12        netmask 255.255.255.0
13        gateway 10.0.0.1
14     }
15     network eth2 {
16        switch lan3
17        address 10.0.2.1
18        netmask 255.255.255.0
19        gateway 10.0.0.1
20     }
21 }
```

First the choke inherits the superclass as usual. Next the network configuration is illuminated. It has three network blocks to represent three network interfaces. Each block is filled with a different switch value, so the choke connects to three virtual switches to detach the whole subnet into three parts.

5.2.3 Build Network

After fulfilling the project file, running this command can build this network efficiently and simply.

```
mln build -f firewallTopology.mln
```

The output of building the network is fairly long, however it is similar with the output of the former network. It reports the process to build virtual switches and virtual machines.
5.2. COMPLEX TOPOLOGY

5.2.4 Start Network

Supposing the network is built successfully, running this command following can start the whole network.

```
mln start -p firewallTopology
```

If the following output is received, the network is started successfully.

```
1 Starting switch lan1.firewallTopology
2 Create virtual switch lan1.firewallTopology...
3 Starting switch lan2.firewallTopology
4 Create virtual switch lan2.firewallTopology...
5 Starting switch lan3.firewallTopology
6 Create virtual switch lan3.firewallTopology...
7 Starting choke...
8 start() = 1
9 Starting desktop...
10 start() = 1
11 Starting gateway...
12 start() = 1
13 Starting laptop...
14 start() = 1
15 Starting server1...
16 start() = 1
17 Starting server2...
18 start() = 1
```

First all virtual switches are started one by one by controlling scripts generated during the building process. Then virtual machines are powered on and connected to corresponding virtual switches. If a `start() = 1` is received, it means this machine is booted successfully.

5.2.5 Check Network Status

If booting the network fluently, running this command to check their status to make sure they are in correct status.

```
mln status -p firewallTopology
```

If the following output is received, the network is running normally.

```
firewallTopology host laptop up
firewallTopology host gateway up
firewallTopology host server1 up
firewallTopology host server2 up
firewallTopology host choke up
firewallTopology host desktop up
firewallTopology switch lan1 up
firewallTopology switch lan2 up
firewallTopology switch lan3 up
```

From the output above, six hosts and three switches are in up mode. During the checking process, MLN calls sub functions of esx plug-in to require status of machines and switches. All machines’ status are requested first and then all switches’s status are requested.
5.2. COMPLEX TOPOLOGY

Figure 5.5: Network view from VMware vSphere Client
5.2. COMPLEX TOPOLOGY

5.2.6 Network View from the VMware ESX Server

This is whole view about the network. On the left side it displays six running machines. On the right side it displays four subnets and one physical adapter. gateway has a network interface to connect to this adapter to provide internet access for other three subnets. Another network interface of gateway connects to lan1 that can be seen in the next subnet. It is also called DMZ where resides other three hosts. Next two subnets are lan2 and lan3, also called WLAN and LAN respectively. As mentioned before, choke is a critical device to conjoin three subnets together. Thus the choke can be found in lan1, lan2 and lan3.

5.2.7 Stop Network

To pause the whole network is very simple, running this following command can accomplish this goal.

```
mln stop --p firewallTopology
```

If the following output is received, it verifies the network is stopped successfully.

```
1 Stopping firewallTopology
2 Stopping server2... 
3 stop() = 1
4 unregister() = 1
5 Stopping server1...
6 stop() = 1
7 unregister() = 1
8 Stopping laptop...
9 stop() = 1
10 unregister() = 1
11 Stopping gateway...
12 stop() = 1
13 unregister() = 1
14 Stopping desktop...
15 stop() = 1
16 unregister() = 1
17 Stopping choke...
18 stop() = 1
19 unregister() = 1
20 Stopping switch lan3.firewallTopology
21 Stopping the virtual switch lan3.firewallTopology...
22 Stopping switch lan2.firewallTopology
23 Stopping the virtual switch lan2.firewallTopology...
24 Stopping switch lan1.firewallTopology
25 Stopping the virtual switch lan1.firewallTopology...
```

The stop() = 1 indicates a machine is shut down successfully. And unregister() = 1 indicates a machine is erased from the network view where it only displays active machines.

5.2.8 Flexible Network Administration

Last section two administration commands to start and stop the whole network is introduced, but they are not eligible to administrate a single host. In order to manipulate one single host, an option -h is provided to specify a host.
Three commands can be used to manage one single host. For example, if system administrators want to detach the three subnets completely, the second command (line 2) can shut down the \textit{choke} merely but keep others running. By executing the second command, only the output of stopping the \textit{choke} is printed.

```
1 Stopping choke . . .
2 stop() = 1
3 unregister() = 1
```

Then system administrators can execute the third command (line 3) to check the \textit{choke}'s status.

```
1 firewallTopology host choke down
2 firewallTopology switch lan1 up
3 firewallTopology switch lan2 up
4 firewallTopology switch lan3 up
```

Thus from the output above, the \textit{choke} is down but other three switches are in up mode. Furthermore if the command \texttt{mln status -p firewallTopology} is executed without option \texttt{-h}, the whole network status should be printed to show that only the host \textit{choke} is down. Below is the output.

```
1 firewallTopology host laptop up
2 firewallTopology host gateway up
3 firewallTopology host server1 up
4 firewallTopology host server2 up
5 firewallTopology host choke down
6 firewallTopology host desktop up
7 firewallTopology switch lan1 up
8 firewallTopology switch lan2 up
9 firewallTopology switch lan3 up
```

More flexible network administration such as increasing/decreasing/modifying a host/switch will be stated in next section.

### 5.2.9 Remove Whole Network

Removing the whole network is simple by executing the following command.

```
mln remove -p firewallTopology
```

The removing sequence is that all hosts are erased first and then all switches are erased. Another sequence is that the network stored in local is removed first (line 3) and then the network stored in the remote VMware ESX server is removed.

```
Are you sure you want to remove firewallTopology? (y/N) y
Removing /opt/mln/projects/firewallTopology
ESX: Remove host laptop.firewallTopology.
Deleted file '[datastore1] laptop.firewallTopology' successfully.
ESX: Remove host gateway.firewallTopology.
Deleted file '[datastore1] gateway.firewallTopology' successfully.
```
5.3. Chapter Summary

From this chapter novel method is used to deploy and administrate virtual machines and virtual switches. They can be described in the MLN language. Many commands are supported to administrate groups of virtual machines and switches. In addition complicated network is able to be constructed and managed by the new method as well.
Chapter 6

Result Three: Advanced Features and User Interface

This is the third part of results. In this chapter, two advanced features, network upgrade and migration are illustrated. Moreover, user interface design and structure are explained as well.

6.1 Advanced Feature One: Network Upgrade

Supposing that a host needs to increase memory size, or an extra switch needs to be added, MLN with esx plug-in can help system administrators to achieve this goal. MLN has this kind of good feature to upgrade the host or alter the network topology without rebuilding the network. Below it is a list to show what kind of features MLN with esx plug-in can undertake.

<table>
<thead>
<tr>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>superclass</td>
<td>As a father class, any attributes can be altered and influenced by hosts</td>
</tr>
<tr>
<td>host</td>
<td>Altering host name is supported</td>
</tr>
<tr>
<td>file system</td>
<td>Altering file system is supported</td>
</tr>
<tr>
<td>disk</td>
<td>Altering disk size is supported</td>
</tr>
<tr>
<td>memory</td>
<td>Altering memory size is supported</td>
</tr>
<tr>
<td>network</td>
<td>Altering interface, IP address, mask and gateway are supported</td>
</tr>
<tr>
<td>users</td>
<td>Altering user name or password are supported</td>
</tr>
<tr>
<td>DNS</td>
<td>Altering DNS location is supported</td>
</tr>
<tr>
<td>switch</td>
<td>Adding/deleting a virtual switch is supported</td>
</tr>
</tbody>
</table>

Table 6.1: Items supported in upgrade process

Unless the project name does not altered, the whole network can be upgraded, such as adding a host, increasing memory size or deleting a switch. If the
project name is changed, MLN will clone the network but not upgrade it. So next it is necessary to give some cases to show how they actually work.

6.1.1 Alter Disk Size

First case is to change the disk size. Let’s go back to the simple Topology (figure 5.1) on page 56. It is a simple topology with three hosts and a switch. The gateway Oslo is a critical machine and its disk will increase from 2GB to 5GB.

First step is to modify the project file simpleTopology.mln by adding one line into the block of host Oslo.

```
size 5GB
```

This line overwrites the size value inherited from the superclass. Then running this following command to upgrade it.

```
mln upgrade -r -f simpleTopology.mln
```

MLN fetches the older project file first and compares it with new project file to find difference. Then MLN selects the different values and upgrades them. The output is so long that only special section is selected and printed comparing the output of building the simple network.

```
fetching old config: /opt/mln/projects/simpleTopology/simpleTopology.mln

---> Upgrade Info:

Attempting upgrade to version 2
The Following Diff has been calculated
host {
    Oslo {
        size 5GB
    }
}

No hosts will be removed

------> UPGRADING simpleTopology
Collecting Status information for simpleTopology

------> Oslo

Removing the older disk...
Deleted file '/[datastore1] Oslo.simpleTopology/Oslo.vmdk' successfully.

Creating disk '/opt/mln/projects/simpleTopology/images/Oslo.vmdk'
Disk expansion completed successfully.

Done

Saving Config file: /opt/mln/projects/simpleTopology/simpleTopology.mln
```
6.1. ADVANCED FEATURE ONE: NETWORK UPGRADE

From the output above, MLN detects only the disk size is altered and prints it on the eighth line. From line 18, the host Oslo has begun to upgrade. First the disk files are erased (line 23 and 24). Then the new disk is created and expanded to 5GB (line 28 and 29). Finally a Done is printed and modified configuration file is saved.

![Figure 6.1: Check disk size from Datastore Browser](image)

Next the disk size can be checked from Datastore Browser to make sure its correct size. From figure 4.8, the Oslo.vmdk is the virtual disk file and its size is 5242MB.

Decreasing disk size is also supported here. However, it is not recommended to shrink disk size since it may cause to lose data.

### 6.1.2 Alter Memory Size

This section is to increase a host’s memory size. For example, system administrators want to increase memory size of virtual machine London from 512M to 1024M. As usual, first step is to modify the project file by adding one line in the host London block.

```
memory 1024M
```

This line overwrites the memory value inherited from the superclass. Then running this following command to upgrade it.

```
mln upgrade -r -f simpleTopology.mln
```

MLN detects the memory change and prints it below. Other part of the output is omitted since it is similar with the output of upgrading disk size.

```
1 --> Upgrade Info:
2 Attempting upgrade to version 3
3 The Following Diff has been calculated
4 host {
5   London {
6     memory 1024M
7   }
8 }
10 No hosts will be removed
```
6.1. ADVANCED FEATURE ONE: NETWORK UPGRADE

After upgrade, system administrator logs into the host London and executes command `free -m` to verify the memory size.

Figure 6.2 is the result of memory size that it displays the total memory is 1003MB. Thus this section to increase memory size is fulfilled successfully. Decreasing the memory is also supported here by altering the memory value smaller than the former host’s memory.

### 6.1.3 Add/Delete a User

This section is to add a user called `tower` into host Paris. First step is to modify the project file by adding a `users` block containing the user name and password.

```plaintext
users {
    tower TuDuxji2f08Xg
}
```

This line inserts a new user into host Paris. Then running this following command to upgrade it.

```plaintext
mln upgrade -r -f simpleTopology.mln
```

MLN detects to insert a new user and prints it below. Other part of the output is omitted since it is similar with the output of upgrading memory size.

```
1 →→→ Upgrade Info:
2 Attempting upgrade to version 4
3 The Following Diff has been calculated
4 host { Paris {
5    users {
6        tower TuDuxji2f08Xg
7    }
8 }
9 }
10 }
11
12 No hosts will be removed
```

During the upgrade process, if the following output is printed, then the new user is added successfully.

```plaintext
Adding users: tower, error: 0
```

After upgrade, system administrator logs into the host Paris by new user `tower` and executes command `less /etc/passwd` to verify the new user existing. Figure 6.3 displays the new user with uid 1002 has been added into host Paris. Thus this section to add a new user is fulfilled successfully. Removing an existing user is also supported here by erasing the user from users block and executing the upgrade command.
6.1. ADVANCED FEATURE ONE: NETWORK UPGRADE

6.1.4 Alter Network

Network configuration including IP address, subnet mask, gateway and DNS location can also be altered by upgrade. First step is to modify the project file by modifying the network block and nameserver value.

```
nameserver 128.39.89.10
network eth0 {
    switch centerSwitch
    address 10.0.0.5
    netmask 255.255.255.0
    gateway 10.0.0.2
}
```

This time the DNS address, IP address and gateway have been changed. Then running the following command to upgrade it.

```
mln upgrade -r -f simpleTopology.mln
```

MLN detects what have changed in the host Paris. Other part of the output is omitted since it is similar with former output.

```
|-> Upgrade Info:
| Attempting upgrade to version 5
| The Following Diff has been calculated
| host {
| Paris {
|    nameserver 128.39.89.10
|    network {
|        eth0 {
|            gateway 10.0.0.2
|            address 10.0.0.5
|        }
|    }
| }
| No hosts will be removed
```

After upgrade, system administrator logs into the host *Paris* to check network configuration. Figure 6.4 displays the network configuration from two files. On the basis of the two files the DNS address is 128.39.89.10, IP address assigned to interface 0 is 10.0.0.5 and the new gateway is 10.0.0.2. So the upgrade of network parameters is fulfilled successfully.

6.1.5 Increase New Network

In this section, a new subnet is created and connected to an existing network created in section 5.1. The new subnet includes a host and a switch. The new topology is painted below.
Next step is to modify the project file to add a new switch *easternSwitch*, a new host *Rome* and a new network interface *eth1* belonging to the host *Paris*.

```
host Paris {
  superclass commonConfig
  network eth0 {
    switch centerSwitch
    address 10.0.0.5
    netmask 255.255.255.0
    gateway 10.0.0.2
  }
  network eth1 {
    switch easternSwitch
    address 10.0.1.2
    netmask 255.255.255.0
    gateway 10.0.0.1
  }
}
host Rome {
  superclass commonConfig
  network eth0 {
    switch easternSwitch
    address 10.0.1.3
    netmask 255.255.255.0
    gateway 10.0.0.1
  }
}
switch easternSwitch {}
```

Then execute the following command to upgrade it.

```
mln upgrade -r -f simpleTopology.mln
```

MLN detects the adding subnet and other part of the output is omitted since it is similar with the former output.
6.1. ADVANCED FEATURE ONE: NETWORK UPGRADE

Figure 6.5: Upgrade network topology
6.1. ADVANCED FEATURE ONE: NETWORK UPGRADE

```plaintext
Rome {
  template debian6.2GB.vmdk
  size 2GB
  memory 512M
  nameserver 128.39.89.8
  superclass commonConfig
  network {
    eth0 {
      switch easternSwitch
      address 10.0.1.3
      netmask 255.255.255.0
      gateway 10.0.0.1
    }
    users {
      karl KrmoKGR69Iqes
      root 0zPt3GSTp5yc0
    }
    esx {
      datastore datastore1
      vc_server 128.39.73.231
      username root
      password password
      vmfslabel 4c74cc2d-f3d16a79-36d5-b8ac6f21f590
      vc_uuid 52 46 f5 2e 65 33 35 04-e1 0e bf f5 aa 6c 8f 13
    }
  }
}
```

No hosts will be removed.

During the upgrade process, the switch `easternSwitch` is built first. Then the host `Paris` is updated and host `Rome` is created.

```plaintext
--- Building switch easternSwitch
---
--- Paris
---
Adding interface eth1 (10.0.1.2)
Adding interface eth0 (10.0.0.5)
Writing interface ethernet1
The interface ethernet1 is going to connect to virtual switch easternSwitch.
simpleTopology
Writing interface ethernet0
The interface ethernet0 is going to connect to virtual switch centerSwitch.
simpleTopology
---
--- Rome
---
Adding interface eth0 (10.0.1.3)
---
---
Done
Saving Config file: /opt/mln/projects/simpleTopology/simpleTopology.mln
```
6.1. ADVANCED FEATURE ONE: NETWORK UPGRADE

After upgrade, system administrator logins into VMware ESX server to check the topology. From the figure 6.6 the new subnet is created and the new network interface of host Paris is extended to this subnet.

![Image of new topology from VMware ESX server]

Figure 6.6: New topology from VMware ESX server

6.1.6 Remove a Host

This section is to remove a host London from the network created in the last section. This piece of code is removed from the project file first.

```mln
host London {
    superclass commonConfig
    network eth0 {
        switch centerSwitch
        address 10.0.0.2
        netmask 255.255.255.0
        gateway 10.0.0.1
    }
}
```

Then execute this following command to upgrade it.

```mln
mln upgrade -r -f simpleTopology.mln
```

MLN detects to remove host London and prints it below. Other part of the output is omitted since it is similar with the former output.

```
---
Upgrade Info:
2
3 Attempting upgrade to version 8
4 No hosts needed to be rebuild
The following hosts will be removed:
5 London
6
7
8
9 ESX: Remove host London.simpleTopology
10 Deleted file ' [datastore1] London.simpleTopology ' successfully.
---
```
6.2. ADVANCED FEATURE TWO: NETWORK MIGRATION

After upgrade, system administrator executes command `mln status -p simpleTopology` to check whether it is removed or not.

```
simpleTopology host Paris down
simpleTopology host Rome down
simpleTopology host Oslo down
simpleTopology switch centerSwitch down
simpleTopology switch easternSwitch down
```

From the output above, it shows that the host London has been removed successfully.

6.1.7 Remove a Switch

This section is to remove a switch `easternSwitch` from the network created in the section 6.1.5. This piece of code is removed from the project file first.

```
host Paris {
    network eth1 {
        switch easternSwitch
        address 10.0.1.2
        netmask 255.255.255.0
        gateway 10.0.0.1
    }
}
host Rome {
    network eth0 {
        switch easternSwitch
    }
}
switch easternSwitch {}
```

Since the switch is removed, two interfaces from host Paris and Rome are removed as well. Then execute this following command to upgrade it.

```
mln upgrade -r -f simpleTopology.mln
```

MLN detects to remove the virtual switch and prints a similar output as before. So it is omitted here. After upgrade, system administrator executes command `mln status -p simpleTopology` to check whether they are removed or not.

```
simpleTopology host Paris down
simpleTopology host Rome down
simpleTopology host Oslo down
```

From the output above, it shows that the switch `easternSwitch` has been removed successfully.

6.2 Advanced Feature Two: Network Migration

Migrating network can be treated as special upgrade. Migration means network including virtual machines and virtual switches are moved from one VMware ESX server to another VMware ESX server.
6.2. ADVANCED FEATURE TWO: NETWORK MIGRATION

6.2.1 Schematic Diagram

Figure 6.7 is schematic diagram of migration. Virtual machine and virtual switch will migrate from one VMware ESX server with IP address 128.39.73.231 to another VMware ESX server with IP address 128.39.73.235.

![Schematic Diagram](image)

Figure 6.7: Migrate schematic diagram

6.2.2 Network

A simple network containing one virtual machine called city and one virtual switch called center is used for migration in this section. The project file is displayed below.

```plaintext
global {
    project country
}

host city {
    esx {
        datastore datastore1
        vc_server 128.39.73.231
        username root
        password corenetwork
        vmfslabel 4c74cc2d-f3d16a79-36d5-b8ae6f21f590
        vc_uuid 52 46 f5 2e 65 33 35 04-e1 0e bf f5 aa 6c 8f 13
        guestos debian5-64
    }
    template debian6.2GB.vmdk
    size 2GB
    memory 512M
    nameserver 128.39.89.8
    network eth0 {
        switch center
        address 128.39.73.236
        netmask 255.255.255.0
        gateway 128.39.73.1
    }
}
```
6.2. ADVANCED FEATURE TWO: NETWORK MIGRATION

It is a simple network. Virtual machine city has one network interface connected to virtual switch center. Virtual machine uses Debian6 as operating system with 2GB disk and 512MB memory. IP address of ESX server is 128.39.73.231 referring which ESX server they are located. Next one executes command `mln build -f country.mln` to construct the network and it can be found in datastore1 showed below.

![Figure 6.8: Virtual machine city displayed in datastore1](image)

From datastore1 two files can be seen. One is virtual machine’s configure file `city.vmx`. Another is virtual machine’s virtual disk `city.vmdk`.

### 6.2.3 Preparation for Migration

In order to migrate, some values in esx block needs to be altered. At least, `vc_server`, `password`, `vmfslabel` and `vc_uuid` are need to be changed and they can be found from new VMX configure file. New esx block is showed below.

```plaintext
esx {
  datastore datastore1
  vc_server 128.39.73.235
  username root
  password corenetwork2
  vmfslabel 4db83218−04c4d991−28f4−000c2908bab8
  vc_uuid 52 e2 33 a9 0b 6a e9 e1−01 3f 78 70 d1 c0 f3 db
}
```

Comparing the former esx block, except username and datastore, other four values are altered. Datastore1 and root are default value for datastore and username respectively. IP address of ESX server is altered definitely. Remaining same password is acceptable, but because of security and plug-in testing, it is changed. For other two values, vmfslabel and vc_uuid can be found and copied from VMX configure file. So far, preparation is done and it is time to migrate.

### 6.2.4 A Single Command to Migrate Network

One mere command `mln upgrade -r -f country.mln` is executed to migrate network. A long output with rich information will be generated. It is so long that it is attached in the Appendix D. However, some important parts are selected...
and explained here.

Firstly, difference between former and latter project file is calculated. Block esx including vmfslabel, password, vc_uuid and vc_server are detected and printed successfully.

```
host {
  city {
    esx {
      vmfslabel 4db83218-04c4d991-28f4-000c2908bab8
      password corenetwork2
      vc_uuid 52 e2 33 a9 0b 6a e9 e1-01 3f 78 70 d1 c0 f3 db
      vc_server 128.39.73.235
    }
  }
}
```

Secondly, series of actions are operated to former network. New directory city.country is created in new VMware ESX server. Old VMX file and local disk are removed. Former virtual disk is downloaded to machine Manager that acts as a transit station. Former virtual disk and directory located remotely are removed.

```
ESX: Create the virtual machine directory...
Created directory '[datastore1] city.country' successfully.
ESX: Removing old remote vmx file...
Deleted file '[datastore1] city.country/city.vmx' successfully.
ESX: Removing old local disk...
ESX: Downloading old remote disk...
Downloaded file to /opt/mln/projects/country/images/city-flat.vmdk successfully.
Downloaded file to /opt/mln/projects/country/images/city.vmdk successfully.
ESX: Remove old remote disk...
Deleted file '[datastore1] city.country/city-flat.vmdk' successfully.
Deleted file '[datastore1] city.country/city.vmdk' successfully.
ESX: Remove host city.country
Deleted file '[datastore1] city.country' successfully.
```

Thirdly, new scripts to start and stop virtual machine are updated.

```
ESX: Creating: /opt/mln/projects/country/start_99_city.sh
ESX: Creating: /opt/mln/projects/country/stop_99_city.sh
```

Fourthly, new VMX configure file and virtual disk are uploaded to new VMware ESX server.

```
ESX: Upload the city.vmx...
Uploaded file /opt/mln/projects/country/city.vmx to city.country/city.vmx successfully.
ESX: Upload the city.vmdk...
Uploaded file /opt/mln/projects/country/images/city.vmdk to city.country/city.vmdk successfully.
ESX: Upload the city-flat.vmdk...
```

Fifthly, new script to manipulate virtual switch is created.

```
ESX: ESX plugin activated to build switch: center.country
ESX: virtual switch: center
```
6.3. USER INTERFACE

Then migration is fulfilled. If system administrator checks Datastore Browser of new VMware ESX server, virtual machine city is able to be found here. It is showed in picture below.

![Figure 6.9: Virtual machine city displayed in new VMware ESX server](image)

6.3 User Interface

Masses of virtual machines, virtual switches or designed network project is able to be deployed by the user interface. User interface is comprised of three PHP web pages and one daemon file, which are enclosed in Appendix G, H, I and J. Their relation is expounded in figure below.

![Figure 6.10: User interface function graph](image)

Home page is face.php in which configure information is filled. After submitting this page, it will call process.php. Process.php will obtain all configure data, prepare MLN project file for network construction and create socket. Through the socket, MLN command is transmitted to daemon.pl. Daemon should be run definitely by root before process.php sends command. Next, process.php calls display.php and display.php will be waiting for output. When MLN receives command, it will begin to build the network. Meanwhile, output will be written into a file named TopologyName in tmp directory and printed by display.php.

Next picture is the face.php.
It is a simple but powerful user interface. First three text fields, username, password and IP are used to specify VMware ESX server, which is mandatory to be filled. Last two text fields, local user and password are prepared for Linux operating system, which is optional to be filled. Next a combo box is provided with two topology templates, which is also optional to be selected. Last component is a button. When the button is clicked, procedure stated in figure 6.10 will be performed. And long output will be printed in display.php to indicate the status of building virtual machines and switches.

The user interface is utilized to order and deploy virtual machines and virtual switches. It is tested in the case of three scenarios and two VMware ESX servers. One server is set up in Oslo. Another server is located in Nittedal. Three scenarios are a single virtual machine, simpleTopology and complexTopology respectively. When one virtual machine (2GB file system) is deployed to a VMware ESX server in Oslo, it takes approximately 5 minutes. When the same virtual machine is deployed to a VMware ESX server in Nittedal, it takes approximately 30 minutes. Furthermore, if more sorts of network project templates are provided, more various of networks are able to be deployed by the user interface.

### 6.4 Chapter Summary

Upgrade and migration are implemented in this chapter. Upgrade is an advanced feature. Attributes of a virtual machine is able to be altered. Users can be added or deleted. Virtual machines or virtual switches can be increased or removed. Migration is also an advanced feature. It is able to migrate a whole network from a VMware ESX server to another VMware ESX server modifying a project file and executing a single command. Moreover, a user interface is created to order virtual machines or networks.
Chapter 7

Result Four: Data Comparison and Analysis

In this chapter, Data and Analysis section in Approach is implemented. In the case of building and administrating same network, data, such as number of VMware commands, number of MLN commands, lines of MLN code and wizard clicks is collected by both VMware method and MLN method. Next, data is compared and analysed in group to achieve optimal solution.

7.1 Network Description Complexity and Coherence

Totally two different methods to describe same network are used. One method is to use VMware GUI to describe virtual machines and virtual switches. Another method is to use MLN description language and MLN commands to demonstrate virtual machines and virtual switches. For two networks (simpleTopology and complexTopology) constructed in section 5.1 and 5.2, number of wizard clicks and lines of MLN description code are gathered and listed below.

<table>
<thead>
<tr>
<th>Network</th>
<th>VMware</th>
<th>MLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>simpleTopology</td>
<td>65</td>
<td>51</td>
</tr>
<tr>
<td>complexTopology</td>
<td>139</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 7.1: Number of wizard clicks and code lines

For network simpleTopology, it takes 65 clicks to demonstrate one virtual switch and three virtual machines by VMware GUI. Moreover it is worth to mention that virtual switch should be described before virtual machines. However, it takes only 51 lines of code to describe network simpleTopology by MLN description language, which can be confirmed in project file in section 5.1.2. For network complexTopology, it takes 139 clicks to demonstrate three virtual switches and six virtual machines by VMware GUI. However, mere 90 lines of MLN code is necessary to describe this network, which can be counted from project file complexTopology.mln attached in Appendix B.
From the data displayed above, to describe same network, it is obvious that MLN method is the optimal solution. For both network, number of MLN code is fewer than VMware’s. As network scalability is increased, number of VMware wizard clicks is raised sharply. While, lines of MLN code is increased as well, but it is not increased as much as VMware’s. When network becomes more and more complex, using MLN to describe network is able to lead to much fewer lines of code, which means using MLN is able to deduce network complexity.

Though using VMware graphic wizard is able to describe network, it is unclear and difficult to have whole network view. However, since MLN uses description language to demonstrate network, it is clear to bring network view completely and coherence of attributes among virtual machines.

### 7.2 Network Deployment Complexity and Automation

Two methods are used to construct network stated in section 5.1 and 5.2. One method is to construct network by VMware commands. Another method is to use MLN administration command to build it. Number of VMware commands and MLN administration commands are gathered and listed below.

<table>
<thead>
<tr>
<th>Network</th>
<th>VMware</th>
<th>MLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>simpleTopology</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>complexTopology</td>
<td>48</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7.2: Number of commands to construct network

For network simpleTopology, it takes 24 commands to build one virtual switch and three virtual machines. These complicated commands are attached in Appendix E. However, it takes mere one MLN command (mln build -f simpleTopology.mln) to build whole network. For network complexTopology, it takes 48 commands to build three virtual switches and six virtual machines. And these commands are attached in Appendix F. If the same network is constructed by MLN command, only one command (mln build -f complexTopology.mln) is necessary to executed.

Thus from data, when scalability of network expands, it takes much more VMware commands to construct network than MLN command. However, for MLN, no matter what network scale it is, it always takes mere one command to construct network. In this case, using MLN method to construct same network is much simple and efficient than VMware method. So using MLN method is able to reduce network construction complexity. Moreover, network construction automation is increased sharply by MLN method, since it takes only one command to build network.
7.3 NETWORK MANIPULATION AUTOMATION

7.3 Network Manipulation Automation

Network manipulation includes operations to start, stop and remove network. For each operation, two methods are provided in this section. First method is to execute VMware commands to manipulate virtual switch or virtual machine. Second method is to execute MLN administration command to manipulate whole network.

Number of VMware and MLN commands to start network is displayed in first table below.

<table>
<thead>
<tr>
<th>Network</th>
<th>VMware</th>
<th>MLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>simpleTopology</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>complexTopology</td>
<td>18</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7.3: Number of commands to boot network

From data, for simpleTopology, it is necessary to execute 8 VMware commands to start whole network. However, only one MLN command (mln start -p simpleTopology) is needed to be executed to start whole network. For complexTopology, more VMware commands, 18 commands are executed to start whole network. While for MLN method, mere one command (mln start -p complexTopology) is needed to executed to start whole network. In two cases, it takes mere one MLN command to start whole network, much fewer than VMware commands. Thus using MLN method is able to increase automation degree of starting network.

Second table displays number of VMware and MLN commands to stop whole network.

<table>
<thead>
<tr>
<th>Network</th>
<th>VMware</th>
<th>MLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>simpleTopology</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>complexTopology</td>
<td>18</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7.4: Number of commands to stop network

From data, it takes exact same number of commands to stop whole network as starting whole network for both simpleTopology and complexTopology by VMware and MLN command. Furthermore, number of MLN command is fewer than VMware’s for both networks. So it takes much fewer commands, actually only one MLN command to stop whole network, which definitely increases degree of automation to stop network.

Number of commands to remove network is showed in the third table 7.5. For simpleTopology, it takes 3 VMware commands to remove whole network. However, only 1 MLN command (mln remove -p simpleTopology) is necessary to be executed to fulfil the same task. For complexTopology, 6 VMware commands are required to remove whole network. But for MLN method, mere
7.4. CHAPTER SUMMARY

<table>
<thead>
<tr>
<th>Network</th>
<th>VMware</th>
<th>MLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>simpleTopology</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>complexTopology</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7.5: Number of commands to remove network

1 command (mln remove -p complexTopology) is needed to be operated to fulfil the same task. Thus when network is removed, executing MLN command is much simpler and more efficient than VMware’s, which leads to high degree of automation to remove a network.

7.4 Chapter Summary

Consequently, from five tables displayed above, using MLN commands to administrate virtual machines and virtual switches is able to decrease network deployment complexity and increase degree of automatic administration and coherence of attributes among networks.
Chapter 8

Discussion

In this chapter, solution design principle, feasible and alternative approach is discussed. Solved and remained problems encountered in this project are listed. Further, solution achievement and influence are also stated. Moreover, future work is explored in the last section.

8.1 Why VMware ESX Platform is Chosen

VMware is the most popular and dominant virtualization technology nowadays. VMware ESX/ESXi is the core product that is an indispensable hypervisor in virtualization system. Both are commercial products. ESX needs a license which costs, while ESXi is free all the time and all features are unlocked during the 60 day trial period. Secondly, a powerful community is settled in the VMware website and elaborate documents are provided. Thus, it is more convenient to discuss problems. And they are swifter to be solved.

Generally there are two methods to manage virtual machines or network on the VMware ESX/ESXi. One is by GUI (Graphic User Interface). Another is through command line. It is hard to make automatic administration by GUI, but it is possible to administrate virtual machines and network automatically by command lines, which needs to read and write to virtualization system, especially writing. Since ESXi does not allow to write, while ESX supports to write, so ESX is chosen for this project.

8.2 Why Use Scripting to Administrate Virtualization System

Imagine a condition, one hundred virtual machines are running in a VMware ESX server and all of them are required to be shut down immediately. One solution is to click stop button 100 times from menu. Another solution is to run stop command 100 times to shut them down. Both methods are inefficient and unwise. So scripting is a good solution to manage them automatically. Although it is a special case, it illustrates that it is difficult to manage masses of
virtual machines by GUI or command. However it is possible to manage them by scripting in which VMware management commands are embedded.

It is worthy to develop script to administrate system. Abstractly, on one hand, it is a re-duplicative work to administrate hundreds of virtual machines or switches. On the other hand, computer is deigned to perform repetitive tasks. Therefore, developing scripting to administrate virtual machines links both of them together. Moreover, scripting brings some advantages. First, efficiency is enhanced. Second, risk of committing mistakes is reduced. However, due to scarce scripting knowledge, professional experience or time constraints, developing scripting is not eligible to everyone. While it is an intelligent choice for skilful system administrators or professionals.

8.3 Feasible Approach

It is feasible to extend MLN to administrate network on VMware ESX platform. Three points stated below to support it. First, MLN is able to describe attributes of virtual machine in a coherent method. Second, plug-in mechanism helps MLN extend virtualization platform. Third, groups of virtual machines or virtual switches are able to be administrated by MLN or its extended plug-in. On the other hand, VMware ESX platform is popular and no proper administration tool has been developed yet. Therefore, it is feasible to develop a new plug-in for MLN to administrate virtual machines and virtual switches in group.

8.4 Alternative Approach

Probably it is feasible to develop another new tool to administrate virtual machines and switches automatically by vSphere Perl SDK. Perl SDK provides programming interfaces to manipulate virtual machines. Virtual machines are expressed and stored in XML file. Perl SDK provides parser, for example `parse_file(‘vm.xml’)` to read it. Other functions, such as `CreateVM()` is provided to create virtual machine. Besides, some advanced features, like snapshot, virtual machine clone, performance monitor and upgrade are able to be implemented by Perl SDK as well. However, though Perl SDK is powerful, it is much complicated than vCLI to administrate virtual machines. Moreover, time might impose serious constraints on approach.

8.5 Comment to Deploy MLN

MLN is deployed on machine Manager. It is simple to install. However, several places need to be noticed. Firstly, MLN is set up for entire system. Secondly, User Mode Linux is not necessary to install. Thirdly, Xen is not necessary to install. And running xend requirement can be ignored. Lastly, since
default template is Debian4 and it is obsolete, so it is not necessary to be downloaded and installed. Rest of questions can be given default answers.

### 8.6 Solved Problems

Throughout this project, both system setup and plug-in development experienced some challenges. Problems are various from hardware and software. They are summarized in several fields below.

#### 8.6.1 Second Physical Adapter Installation

Only one physical adapter is deployed in machine Manager. It is used for VMware ESX Management Network and critical to be connected all the time. Otherwise console loses control to ESX server. In order to migrate virtual machine, it is necessary to deploy second physical adapter preserved for migrate merely. It is simple to install a compatible network adapter for VMware ESX. However, it is almost impossible to install an incompatible network adapter for VMware ESX. Thus selecting the proper network card is extremely important. First network card used for VMware ESX is D-Link Gigabit PCI Desktop Adapter DGE-528T. VMware ESX server is booted many times, but network card is not able to be detected and installed. Next combining machine model (Dell OptiPlex 780) and CPU (Intel Core2 Duo CPU E7500), VMware hardware compatible list is scanned to look for compatible network card. Result is that only Intel network adapter is acceptable. Therefore new network adapter Intel PRO/1000GT Desktop Adapter is ordered and installed successfully.

From this case, hardware compatibility is learnt. Since network adapter is produced by many brands, and they are not always compatible. So it is necessary to check compatibility before installing hardware.

#### 8.6.2 MAC Address Solution

MAC address can be assigned manually by system administrator to be a static MAC address. It ranges from 00:50:56:00:00:00 to 00:50:56:3F:FF:FF. It is noticeable that it has same prefix 00:50:56. In order to achieve it, network adapter address type is needed to assigned to “static”. Then MAC address can be assigned by system administrator. However, in order to administrate virtual machines and virtual switches automatically, MAC address of virtual machine is needed to be generated automatically. Thus network adapter address type is assigned to “generated” and it is generated automatically as soon as it is registered from Datastore to Inventory every time.

#### 8.6.3 Erase Error Message When Mounting VMDK File System

VDDK is required to install to mount VMDK file system. Although it is deployed fluently, library `libvixDiskLibVim.so` is scarce when VMDK file system is mounted. Then log file (`/tmp/vmware-root/fuseMount.log`) is investigated to
look for clue. A clue informing *libgmodule-2.0.so.0* is missing. Thus increasing this library is solution. The library is found from VMwareTools, so it is added into Linux linker configure file (*ld.so.conf*) and *ldconfig* is executed to update linkers. After that, no error message is appeared when command *vmware-mount* is executed.

This problem was unforeseen. However, the system administrator should be familiar with log that indicates system state. From log, system operating trial is able to be tracked and reported, which is helpful to system administrator to solve problem.

### 8.6.4 Hot-remove Project

Plug-in *esx.pl* is designed to be able to remove project even though it is running. For example, executing following command is to remove running project *simpleTopology*.

```bash
mln remove -p simpleTopology
```

Then MLN begins to remove *simpleTopology*. Firstly, plug-in detects whether virtual machine is up or down, if it is down, it will be removed directly. If it is up, it will be shut down and cancelled register from Inventory of VMware ESX. Secondly the plug-in will remove it as usual. Output of removing *simpleTopology* is expressed below.

```
Are you sure you want to remove simpleTopology? (y/N)
y
Removing /opt/mln/projects/simpleTopology
ESX: Paris.simpleTopology has been registered.
ESX: The host Paris.simpleTopology is up, stop it first...
stop() = 1
ESX: Paris.simpleTopology has been registered.
ESX: The host Paris.simpleTopology is down
ESX: Remove host Paris.simpleTopology.
ESX: The host Rome.simpleTopology is up, stop it first...
stop() = 1
ESX: Rome.simpleTopology has been registered.
ESX: The host Rome.simpleTopology is down
ESX: Remove host Rome.simpleTopology.
Deleted file ' [datastore1] Rome.simpleTopology ' successfully.
ESX: The host Oslo.simpleTopology is up, stop it first...
stop() = 1
ESX: Oslo.simpleTopology has been registered.
ESX: The host Oslo.simpleTopology is down
ESX: Remove host Oslo.simpleTopology.
Deleted file ' [datastore1] Oslo.simpleTopology ' successfully.
ESX: ESX plugin to remove virtual switch activated...
ESX: virtual switch centerSwitch.simpleTopology is on, stop it first...
Stopping switch centerSwitch.simpleTopology
Stopping the virtual switch centerSwitch.simpleTopology...
ESX: virtual switch centerSwitch.simpleTopology has been removed successfully
```

The first line is a prompt to confirm whether network *simpleTechnology* is removed or not. If input is "y", MLN will remove local network copy first. Next MLN begins to remove network stored remotely. Status of first virtual
8.7. FURTHER DEVELOPMENT

Machine Paris is checked. Line 5 indicates that Paris is still up, so stop command is executed. \texttt{stop() = 1} is printed in line 6 to indicate that Paris has been shut down. Next from line 9, plug-in esx.pl starts to remove virtual machine Paris remotely. Line 10 indicates Paris has been removed successfully. Other two virtual machines, Rome and Oslo are stopped and removed following the same process. Finally, virtual switch centerSwitch is stopped and removed from last three lines.

Since VMware commands are executed in background, it is not necessary to await other virtual machines operations to be done when a virtual machine is going to be shut down.

8.6.5 MLN Upgrade Command Modification

Normally, upgrade command of MLN is \texttt{mln upgrade -f project.mln}. However, it is interrupted in the beginning of upgrading network stored in VMware ESX server. It is abandoned because of the code showed below in MLN source code.

\begin{verbatim}
exit (1) unless $REMOVE;
\end{verbatim}

So solution is to set variable \texttt{REMOVE} to one to avoid to be aborted. Moreover MLN provides option \texttt{-r} to set it to one. Thus upgrade command executed in this project is \texttt{mln upgrade -r -f project.mln}. Therefore MLN is able to escape \texttt{exit (1)} to continue to execute code to finish upgrade task.

8.7 Further Development

Due to time constraints, several problems remain unsolved. However, it is worthy to develop them further in new projects.

8.7.1 Login Terminal Problem

Linux provides terminal from tty0 to tty7 to user to login into system. Normal user is able to login to Linux from tty0 to tty7. However, root is not able to login into Linux through tty0, which is defined in configure file \texttt{/etc/securetty}. If debian6 operating system encapsulated in VMDK file is mounted in Manager machines, all tty terminals are provided. However, if it is deployed into VMware ESX server by MLN and plug-in esx.pl, only tty0 is provided. Therefore, root is not enable to login into system. If tty0 is added into configure file \texttt{/etc/securetty}, then this problem is solved.

This problem is not solved in the plug-in. However, it is feasible to increase more code in the function \texttt{esx_configure} to solve the problem. The interface \texttt{esx_configure} has been provided in the plug-in. More code to modify configure file \texttt{/etc/securetty} is expected to add into the interface.
8.7. FURTHER DEVELOPMENT

8.7.2 Compromise Solution to Upgrade Virtual Switch

Considering a condition described in list below, virtual switch needs to be update to lan2. Two places are critical to be modified in MLN project file. However, since plug-in esx.pl is not intelligent enough, it does not work.

network eth0{
    switch lan1 => switch lan2
} switch lan1 {}
switch lan2 {}

But another compromising approach solves this problem, which is described below. Except updating switch name, first old switch definition lan1 is re-mained. Then one executes mln upgrade command. Next one updates mln project file to delete switch lan1 and executes mln upgrade command again. Then the switch lan1 is removed.

network eth0{
    switch lan1 => switch lan2
} switch lan1 {}
switch lan2 {}
switch lan2 {}
switch lan1 {}

Reason causing the bug is design defects from variable DIFF in MLN source code. DIFF is a global variable to tell difference between current project and former project. However, it provides virtual machines’ difference merely, no virtual switches’ difference at all. Thus, solution is to add flags and conditional statements to determine whether virtual switch exists or not in function esx_buildSwitch of plug-in esx.pl. It almost solves the problem of upgrading virtual switches, but it makes function esx_buildSwitch much complex.

In order to upgrade virtual switches described in the first condition, function esx_buildSwitch might need to be redesigned or DIFF might need to be updated.

8.7.3 Partition Problem in Linux Operating System

VDDK provides methods to expand disk, but no methods to expand partition. After expanding disk, a helpful message is printed to inform system administrator that third party partition tool is required to fulfil this task. From VMware official website, several tools are listed, such as gparted and BootIt NG. However, they are tools based on GUI. It is difficult to merge them into MLN plug-in. One possible solution is to find a tool to make partitions to VMDK file system only by commands. And these commands are able to be merged into MLN or plug-in. Then partition problem can be solved.

8.7.4 Resource Pool Management

Resource pool is a new mechanism to administrate virtual machines. It is treated as a container by a group of virtual machines. CPU and memory can be assigned to a resource pool occupied exclusively by group of virtual machines. Thus, managing resource pool is an advanced feature, if it is implemented into plug-in esx.pl. However, managing resource pool is not supported by vCLI,
which is used by plug-in in this project. Managing resource pool is supported by another library, called VMware vSphere Perl SDK. It provides Perl scripting interface to vSphere API to manage resource pool. Therefore, one solution is to develop a new plug-in with vSphere API to create and manage resource pool.

8.7.5 Separate Build and Upgrade Process

When either `mln build` or `mln upgrade` is executed, MLN will follow the same route to do the task. Due to follow the design pattern, build and upgrade features are implemented in a same function in plug-in `esx.pl`, which makes the function so complicated. Since build and upgrade are two different features, it is natural and logical to separate and implement them into two different functions from programming development principle, which makes it less complicated and easier to read.

Therefore, in further project, build and upgrade process are advised to be separated and implemented into two functions, which is beneficial to code reuse and further development.

8.8 Accomplishment

Lots of great achievements are obtained. Plug-in `esx.pl` is developed successfully. It is able to be deployed in productive environment or education institute to administrate virtual machines and virtual switches. Moreover, sound features, upgrade and migration are implemented as well.

8.8.1 Plug-in Mechanism

Plug-in `esx.pl` is developed successfully. Plug-in mechanism is confirmed to be feasible. It is able to associate MLN to deploy and administrate virtual machines and virtual switches to VMware ESX server. All development work is accomplished in plug-in and no modifications are occurred in MLN main source code. So no backward testing for Xen and UML platforms is required.

8.8.2 Network Deployment and Administration

Former MLN commands are kept in shape and still work. Two cases, simple-Topology and complexTopology are illustrated in chapter result two. They are real network deployed by MLN with plug-in `esx.pl`. From two networks, competence of MLN to deploy and administrate network is showed. Through the first network, how to use MLN and esx block keyword are demonstrated. An esx block is added into MLN project file to contain ESX server information. It is written into superclass only one time and inherited by other virtual machines. Besides, basic operations are also demonstrated. Building whole network is fulfilled by one command. Starting/Stopping whole network is operated by only one command as well. Other basic operations, like single virtual machine
8.9. PLUG-IN INFLUENCE

administration and network removal are also demonstrated.

The second case is to show how complicated network MLN is able to construct and administrate. No matter how complex network is, as long as it can be described by MLN language, it can be built by only one MLN command. Typical topology, such as mesh topology and star topology can be described in MLN project file and built efficiently.

8.8.3 Upgrade and Migration

MLN is able to upgrade virtual machines or networks on the Xen platform. However, Upgrading virtual machine or network on VMware ESX platform is a challenge. Since build feature and upgrade feature share same function in MLN main source code, they have to be implemented in same function in plug-in as well, which brings lots of conditional statements. Code of plug-in has been altered a lot to achieve the goal. Thus two networks, simpleTopology and complexTopology have to be tested again to guarantee that former achievement is still work. For upgrade testing, various of attributes are taken into consideration, which is stated in section 6.1 of chapter result three. From result, basic upgrade feature is achieved. Size of disk and memory can be upgraded. New user can be increased. Additional networks, including new virtual machines and virtual switches are able to be added or removed. Therefore, former network is able to be altered and upgraded, without reconstruction. Migration is another advanced feature. By changing vCenter IP address, VMFS label and UUID, network is able to be moved among ESX servers. It is a flexible administration method. For example, in order to achieve highest efficiency, networks might be moved to idletest ESX server to operate.

8.8.4 User Interface

VMware ESX server also has a user interface (VA Marketplace) to order and deploy a virtual machine. However, neither masses of virtual machines nor virtual switches are able to be deployed. But through my user interface, abundant virtual machines or switches can be ordered and deployed.

8.9 Plug-in Influence

Nowadays VMware ESX virtualization platform is popular and lots of ESX servers are deployed all over the world. Deployment and administration of virtual machines or switches are basic tasks to system administrators. Next, plug-in esx.pl has good quality to associate MLN to deploy and administrate virtual machines and switches to VMware ESX server. Therefore, it is a big market demand for the plug-in. The plug-in has brought a new but simple and automatic way to deploy and administrate virtual machines and switches.
8.10 Future Work

Although features of plug-in are almost implemented, some additional works are left to make the plug-in more excellent. First work is probably to do code optimization to make code clearer. Second work is to develop more beautiful and powerful user interfaces to manipulate and monitor virtual machines and switches constantly. Third, more specific file system modifications are able to be implemented in function \textit{esx\_configure}, which has been reserved in the plug-in.

More testing work remains. First an exciting test is from a bachelor group who are using MLN for their project in Gjøvik University College. Plug-in esx.pl is expected to test in their project. Second a testing is from a master student’s final project in University of Oslo. In his final project, in order to explore performance of distributed operating system, masses of virtual machines are deployed and administrated. Therefore, it is an ideal opportunity and environment to test plug-in esx.pl. Although plug-in is within his consideration, due to his time constraint, in the end Proxmox is selected to deploy and administrate virtual machines. However, I believe he would like to test plug-in esx.pl, if his time is ample.

Supporting more sorts of operating system is also one of the future work. First, multiple file system templates would be created. Debian 6 file system template has been supported. Other Linux and Windows template would be created and supported as well. Second, a keyword \textit{guestos} would be increased in esx block in MLN project file to refer which sort of file system it is.
Chapter 9

Conclusions

Nowadays virtual machines occupy big market and they are in big demand. VMware virtualization platform is popular and deployed by many companies. However, it is complicated to deploy and inefficient to administrate virtual machines and switches on VMware ESX platform.

Thus, a problem statement is posed to reduce deployment complexity and increase virtual machine coherence and administration automation. Around the problem statement, new plug-in esx.pl is developed to associate MLN to deploy and administrate virtual machines and switches on VMware ESX platform. Through this thesis, two vivid cases are demonstrated to exhibit how deployment complexity is decreased and how automatic administration is increased. Moreover, structure of virtual machines is also revealed to increase coherence.

Furthermore, two advanced features, upgrade and migration are implemented as well. Upgrade is a great functionality to alter attribute of virtual machine without rebuilding it. Migration is also powerful to move virtual machines and switches among VMware ESX servers. Two features above enhance the capability of plug-in to administrate virtual machines and switches. Besides, additional user interface is developed to make administration more convenient.

In conclusion, a new tool is implemented and novel manner is brought to deploy and administrate virtual machines and networks on VMware ESX platform.
Bibliography


[35] VMware Player Documentation


[38] Home page of VMware vCenter Server Heartbeat:

[39] VMware vCenter Orchestrator (vCO) Documentation


Appendix A

Full Output of Building the simpleTopology

---

BUILDING simpleTopology

VMware vSphere ESX plugin esx_postParse called

Switch centerSwitch

++ printing Superclasses ++

-> Superclass commonConfig

commonConfig: template = debian6.2GB.vmdk

++ printing Hosts ++

-> Host Paris

Paris: nameserver = ’128.39.89.8’

Paris: superclass = ’commonConfig’

Paris: memory = ’512M’

Paris: size = ’2GB’

Paris: template = debian6.2GB.vmdk

printing network

Paris: Network interfaces

Interface eth0

address: 10.0.0.3

netmask: 255.255.255.0

gateway: 10.0.0.1

### users:

karl KrmOkGR69Iqes

root 0zP13STp5yc0

-> Host London

London: nameserver = ’128.39.89.8’

London: superclass = ’commonConfig’

London: memory = ’512M’

London: size = ’2GB’

London: template = debian6.2GB.vmdk

printing network

London: Network interfaces

Interface eth0

address: 10.0.0.2

netmask: 255.255.255.0

gateway: 10.0.0.1

### users:

karl KrmOkGR69Iqes

root 0zP13STp5yc0

-> Host Oslo

Oslo: nameserver = ’128.39.89.8’

Oslo: superclass = ’commonConfig’

Oslo: memory = ’1024M’

Oslo: size = ’2GB’

Oslo: template = debian6.2GB.vmdk

printing network
Oslo: Network interfaces

Interface eth1
address: 128.39.73.253
gateway: 128.39.73.1

Interface eth0
address: 10.0.0.1
netmask: 255.255.255.0
gateway: 10.0.0.1

### users:
karl KrmoKGR69Iqes
root 0zPt3GSTp5yc

Saving Config file: /opt/mln/projects/simpleTopology/simpleTopology.mln

Building switch centerSwitch

vmware_buildSwitch called:

---

Paris

DATASTORE: datastore1
VC_SERVER: 128.39.73.231
ESX_HOST_USERNAME: root
ESX_HOST_PASSWORD: corenetwork
ESX plugin is enabled.
sizeDeclaredFromScript: 2
Image name: debian6.2GB
Image size of template: 2147483648
GBTemplateSize: 3

Create the machine folder first...

And the template is found, copying...


[2011-04-03 04:31:12.312 7f66752d1700 verbose 'ThreadPool'] Thread info: Min Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max fds: 2, 21, 2, 10, 31, 4, 600, -1

[2011-04-03 04:31:12.312 7f66752d1700 trivia 'ThreadPool'] Thread pool launched

Creating disk '/opt/mln/projects/simpleTopology/images/Paris.vmdk'

Convert: 100% done.

Virtual disk conversion successful.

esx_createFilesystem did filesystem creation

ESX plugin is mounting Paris on /root/.mln/mountdir

[2011-04-03 04:33:06.430 7f98e31b700 info 'DiskLibPlugin'] Current working directory: /opt/mln/projects/root

Filesystem   Size Used Avail Use% Mounted on
/dev/sda1  222G  59G  152G  29% /
tmpfs  1.9G  0  1.9G  0% /lib/init/rw
udev  1.9G  152K  1.9G  1% /dev
/tmpfs  1.9G  0  1.9G  0% /dev/shm
/dev/loop0  1.9G  1016M  768M  57% /root/.mln/mountdir

Adding users: karl, error: 0
useradd: warning: the home directory already exists.
Not copying any file from skel directory into it.

root,Adding interface eth0 (10.0.0.3)
Adjusting /etc/inittab for UML

ESX plugin is unmounting Paris on /root/.mln/mountdir

[2011-04-03 04:33:07.484 7f3a5be59700 info 'DiskLibPlugin'] Current working directory: /opt/mln/projects/root
100 [2011−04−03 04:33:07.484 7F3A5BE59700 verbose 'ThreadPool'] Thread info: Min
Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max
fds: 2, 21, 2, 10, 31, 4, 600,

101 [2011−04−03 04:33:07.484 7F3A5BE59700 trivia 'ThreadPool'] Thread pool
launched

102 Filesystem Size Used Avail Use% Mounted on
103 /dev/sda1 222G 59G 152G 29% /
104 tmpfs 1.9G 0 1.9G 0% /lib/init/rw
105 udev 1.9G 152K 1.9G 1% /dev
106 tmpfs 1.9G 0 1.9G 0% /dev/shm

107 VC_SERVER: 128.39.73.231
108 ESX_HOST_USERNAME: root
109 ESX_HOST_PASSWORD: corenetwork
110 VMFSLABEL: 4c74cc2d−f3d16a79−36d5−b8ac6f21f590
113 VC_UUID: 52 46 f5 2e 65 33 35 04−e1 0e bf f5 aa 6c 8f 13
114 DATASORE: datastore1
115 VC_SERVER: 128.39.73.231
116 ESX_HOST_USERNAME: root
117 ESX_HOST_PASSWORD: corenetwork
118 VMFSLABEL: 4c74cc2d−f3d16a79−36d5−b8ac6f21f590
119 Writing interface ethernet0
120 The interface ethernet0 is going to connect to virtual switch centerSwitch.
121 simpleTopology
122 Upload the Paris.vmx...
124 simpleTopology/Paris.vmx successfully.
125 Upload the Paris.vmdk...
127 simpleTopology/Paris.vmdk successfully.
128 Upload the Paris−flat.vmdk...
128 simpleTopology/Paris−flat.vmdk successfully.

129 ——> London
130 DATASORE: datastore1
131 VC_SERVER: 128.39.73.231
132 ESX_HOST_USERNAME: root
133 ESX_HOST_PASSWORD: corenetwork
134 ESX plugin is enabled.
135 sizeDeclaredFromScript: 2
136 Image name: debian6.2GB
137 Image size of template: 2147483648
138 GBTemplateSize: 3
139 Create the machine folder first...
140 Created directory '[datastore1] London.simpleTopology' successfully.
141 Disk size declared in the project is small, expanding to template size 3
142 automatically...
143 And the template is found, copying...
144 [2011−04−03 04:34:43.244 7F6CC92A2700 info 'DiskLibPlugin'] Current working
directory: /opt/mln/projects/root
145 [2011−04−03 04:34:43.244 7F6CC92A2700 verbose 'ThreadPool'] Thread info: Min
Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max
fds: 2, 21, 2, 10, 31, 4, 600, −1
146 [2011−04−03 04:34:43.245 7F6CC92A2700 trivia 'ThreadPool'] Thread pool
launched
147 Creating disk '/opt/mln/projects/simpleTopology/images/London.vmdk'
148 Convert: 100% done.
149 Virtual disk conversion successful.
150 esx_createFiesystem did filesystem creation
151 ESX plugin is mounting London on /root/.mln_mountdir
152 [2011−04−03 04:36:34.014 7F1CF1948700 info 'DiskLibPlugin'] Current working
directory: /opt/mln/projects/root
153 [2011−04−03 04:36:34.014 7F1CF1948700 verbose 'ThreadPool'] Thread info: Min
Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max
fds: 2, 21, 2, 10, 31, 4, 600, −1

113
Adding users: karl, error: 0
useradd: warning: the home directory already exists.
Not copying any file from skel directory into it.

Adding users: root, error: 0
Adding interface eth0 (10.0.0.2)
Adjusting /etc/init.d for UML
ESX plugin is unmounting London on /root/.mln_mountdir

[2011-04-03 04:36:37.447 7F3DAC0AB700 info 'DiskLibPlugin'] Current working directory: /opt/mln/projects/root

ESX HOST USERNAME: root
ESX HOST PASSWORD: corenetwork
VMFSLABEL: 4c744c2d-f3d16a79-36d5-b8ac6f21f590
creating: /opt/mln/projects/simpleTopology/start_99_London.sh
creating: /opt/mln/projects/simpleTopology/stop_99_London.sh

VC UUID: 52 46 f5 2e 65 33 35 04--e1 0e bf f5 aa 6c 8f 13
DATASTORE: datastore1
VC SERVER: 128.39.73.231
ESX HOST USERNAME: root
ESX HOST PASSWORD: corenetwork
VMFSLABEL: 4c744c2d-f3d16a79-36d5-b8ac6f21f590
Writing interface ethernet0
The interface ethernet0 is going to connect to virtual switch centerSwitch.
simpleTopology
Upload the London.vmx...
simpleTopology/London.vmx successfully.
Upload the London.vmdk...
simpleTopology/London.vmdk successfully.
Upload the London--flat vmdk...

-> Oslo
DATASTORE: datastore1
VC SERVER: 128.39.73.231
ESX HOST USERNAME: root
ESX HOST PASSWORD: corenetwork
ESX plugin is enabled.
sizeDeclaredFromScript: 2
Image name: debian6_2GB
Image size of template: 2147483648
GBTemplateSize: 3
Create the machine folder first...
Created directory '[datastore1] Oslo.simpleTopology' successfully.
Disk size declared in the project is small, expanding to template size 3 automatically...
And the template is found, copying...

207 [2011−04−03 04:38:19.883 7F3D93874700 verbose 'ThreadPool'] Thread info: Min Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max fds: 2, 21, 2, 10, 31, 4, 600, −1
208 [2011−04−03 04:38:19.883 7F3D93874700 trivias 'ThreadPool'] Thread pool launched
210 Virtual disk conversion successful.
211 ESX plugin is mounting Oslo on /root/.mln/mountdir
212 [2011−04−03 04:40:10.420 7F1A47238700 info 'ThreadPool'] Thread pool launched
213 [2011−04−03 04:40:10.420 7F1A47238700 verbose 'ThreadPool'] Thread info: Min Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max fds: 2, 21, 2, 10, 31, 4, 600, −1
214 [2011−04−03 04:40:10.420 7F1A47238700 trivias 'ThreadPool'] Thread pool launched
215 Filesystem Size Used Avail Use% Mounted on
216 /dev/sda1 222G 63G 148G 30% /
217 tmpfs 1.9G 0 1.9G 0% /lib/init/rw
218 udev 1.9G 152K 1.9G 1% /dev
219 tmpfs 1.9G 0 1.9G 0% /dev/shm
220 /dev/loop0 1.9G 1016M 769M 57% /root/.mln/mountdir
221 Adding users: karl, error: 0
222 useradd: warning: the home directory already exists.
223 Not copying any file from skel directory into it.
224 root, Adding interface eth1 (128.39.73.253)
225 Adding interface eth0 (10.0.0.1)
226 Adjusting /etc/inittab for UML
227 ESX plugin is unmounting Oslo on /root/.mln/mountdir
228 [2011−04−03 04:40:13.320 7FBC1A1D6700 info 'DiskLibPlugin'] Current working directory: /opt/mln/projects/root
229 [2011−04−03 04:40:13.320 7FBC1A1D6700 verbose 'ThreadPool'] Thread info: Min Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max fds: 2, 21, 2, 10, 31, 4, 600, −1
230 [2011−04−03 04:40:13.320 7FBC1A1D6700 trivias 'ThreadPool'] Thread pool launched
231 VCF: 128.39.73.231
232 ESX_HOST_USERNAME: root
233 ESX_HOST_PASSWORD: corenetwork
234 VMFSLABEL: 4c74cc2d-f3d16a79-36d5-b8ac6f21f590
237 VC_UUID: 52 46 f9 2e 65 33 35 04−e1 0e bf f5 aa 6c 8f 13
238 DATASSTORE: datastore1
239 VCF: 128.39.73.231
240 ESX_HOST_USERNAME: root
241 ESX_HOST_PASSWORD: corenetwork
242 VMFSLABEL: 4c74cc2d-f3d16a79-36d5-b8ac6f21f590
243 Writing interface ethernet1
244 The interface ethernet1 is going to connect to virtual switch VM Network
245 Writing interface ethernet0
246 The interface ethernet0 is going to connect to virtual switch centerSwitch.
247 simpleTopology
248 Upload the Oslo.vmx...
250 Upload the Oslo.vmdk...
Upload the Oslo-flat vmdk...

Uploaded file /opt/mln/projects/simpleTopology/images/Oslo-flat.vmdk to Oslo.

simpleTopology/Oslo-flat.vmdk successfully.

PROJECT simpleTopology FINISHED
Appendix B

Project File of Complex Topology
(complexTopology.mln)

```mln
global {
  project firewallTopology
}

superclass commonConfig {
  esx {
    datastore datastore1
    vc_server 128.39.73.231
    username root
    password corenetwork
    vmfslabel 4c74cc2d-f3d16a79-36d5-b8ac6f21f590
    vc_uuid 52 46 f5 2e 65 33 35 04-e1 0e bf f5 aa 6c 8f 13
  }
  template debian6.2GB.vmdk
  size 2GB
  memory 128M
  network eth0 {
    netmask 255.255.255.0
    gateway 10.0.0.1
  }
  users {
    admin ZjCdn5kgD6vvY
    root VJZYCcpp8xHUYZ
  }
}

host gateway {
  superclass commonConfig
  size 3GB
  memory 256M
  nameserver 128.39.89.8
  network eth0 {
    address 128.39.73.251
    gateway 128.39.73.1
  }
  network eth1 {
    switch lan1
    address 10.0.0.1
    netmask 255.255.255.0
    gateway 10.0.0.1
  }
}
```
host choke {
    superclass commonConfig
    network eth0 {
        switch lan1
        address 10.0.0.2
    }
    network eth1 {
        switch lan2
        address 10.0.1.1
        netmask 255.255.255.0
        gateway 10.0.0.1
    }
    network eth2 {
        switch lan3
        address 10.0.2.1
        netmask 255.255.255.0
        gateway 10.0.0.1
    }
}

host server1 {
    superclass commonConfig
    network eth0 {
        switch lan1
        address 10.0.0.3
    }
}

host server2 {
    superclass commonConfig
    network eth0 {
        switch lan1
        address 10.0.0.4
    }
}

host laptop {
    superclass commonConfig
    network eth0 {
        switch lan2
        address 10.0.1.2
    }
}

host desktop {
    superclass commonConfig
    network eth0 {
        switch lan3
        address 10.0.2.2
    }
}

switch lan1 { }
switch lan2 { }
switch lan3 { }
Appendix C

Plug-in esx.pl

```perl
# VMware ESX MLN plug-in written by Dong Yan
# Tested on VMware vSphere ESX 4.1.0

my $ESX_PLUGIN_VERSION = 0.1;
my $DEFAULT_DATASTORE = "datastore1";
my $DEFAULT_VC_SERVER = "128.39.73.231";
my $DEFAULT_ESX_HOST_USERNAME = "root";
my $DEFAULT_ESX_HOST_PASSWORD = "corenetwork";
my $DEFAULT_VMFSLABEL = "4c74cc2d-f3d16a79-36d5-b8ac6f21f590";
my $DEFAULT_VC_UUID = "52 46 f5 2e 65 35 04-e1 0e bf f5 aa 6c 8f 13";
my $DEFAULT_PORTGROUP_NAME = "VM Network";
# Since vCLI is 64 bits package, 32 bits is not supported
# Ubuntu: guestOS = "ubuntu-64"
my $DEFAULT_GUESTOS = "debian5-64";

sub esx_version {
  out("ESX: VMware vSphere ESX plugin version $ESX_PLUGIN_VERSION\n", "red");
}

sub esx_incomingLiveVM {
  out("ESX: VMware vSphere ESX plugin esx_incomingLiveVM called\n", "red");
}

sub esx_configure {
  out("ESX: VMware vSphere ESX plugin esx_configure called\n", "red");
}

sub esx_createFilesystem {
  my $hostname = $[0];
  my $TemplateSize = 0;
  my $GBTemplateSize = 0;
  my $sizeDeclaredFromScript = 0;
  my $freeSpace = 0;
  my $diskSizeUnit = "GB";
  my $freeSpaceSizeUnit = "MB";
  my $old_memory;
  my $new_memory;
  my $old_users;
  my $new_users;
  my $diff_users;
  my $old_disk;
  my $new_disk;
  my $old_nameserver;
  my $new_nameserver;
  my $old_network;
  my $new_network;
  my $diff_network;
```
my $old_datastore;
my $new_datastore;
my $old_vc_server;
my $new_vc_server;
my $old_username;
my $new_username;
my $old_password;
my $new_password;
my $old_vmfslabel;
my $new_vmfslabel;
my $old_vc_uuid;
my $new_vc_uuid;

if (getScalar("/host/\$hostname/esx")){
  my $DATASTORE = getScalar("/host/\$hostname/esx/datastore");
  $DATASTORE = $DEFAULT_DATASTORE unless $DATASTORE;
  verbose("ESX: DATASTORE: $DATASTORE\n");
  my $VC_SERVER = getScalar("/host/\$hostname/esx/vc_server");
  $VC_SERVER = $DEFAULT_VC_SERVER unless $VC_SERVER;
  verbose("ESX: VC_SERVER: $VC_SERVER\n");
  my $ESX_HOST_USERNAME = getScalar("/host/\$hostname/esx/username");
  $ESX_HOST_USERNAME = $DEFAULT_ESX_HOST_USERNAME unless $ESX_HOST_USERNAME;
  verbose("ESX: ESX_HOST_USERNAME: $ESX_HOST_USERNAME\n");
  my $ESX_HOST_PASSWORD = getScalar("/host/\$hostname/esx/password");
  $ESX_HOST_PASSWORD = $DEFAULT_ESX_HOST_PASSWORD unless $ESX_HOST_PASSWORD;
  verbose("ESX: ESX_HOST_PASSWORD: $ESX_HOST_PASSWORD\n");
}

# Open file to write all VMware commands
my $filename = "$PROJECT_PATH/$PROJECT/VMWARE_COMMANDS";
open(VMWARECOMMANDS, ">>$filename") or die "Error: fail to open $filename\n";
print VMWARECOMMANDS "START\n";

out("ESX: ESX plugin is enabled.\n");
my $stem = getScalar("/host/\$hostname/template");
my $sizeDeclaredFromScript = getScalar("/host/\$hostname/size");
my $diskSizeUnit = "GB";

if ( $sizeDeclaredFromScript =~ /\(d+)\[(G|M)B\]$/ ){
  $sizeDeclaredFromScript = $1;
  $diskSizeUnit = $2;
} else {
  out("ESX: The scalar sizeDeclaredFromScript or diskSizeUnit is wrong, abort.\n");
  exit 1;
}

verbose("ESX: sizeDeclaredFromScript: $sizeDeclaredFromScript\n");
my $freeSpace = getScalar("/host/\$hostname/free_space");
verbose("ESX: freeSpace: $freeSpace\n");
my $freeSpaceSizeUnit = "MB";

if ( $freeSpace =~ /\(d+)\[(G|M)B\]$/ ){
  $freeSpace = $1;
  $freeSpaceSizeUnit = $2;
} else {
  out("ESX: The scalar freeSpace or freeSpaceSizeUnit is wrong, abort.\n");
  exit 1;
}

verbose("ESX: freeSpace3: $freeSpace\n");

if (stat("TEMPLATEDIR/$stem")){
  if ( stat("TEMPLATEDIR/$stem"){

  # If the template image is here, print its real disk size
my $imageName;
if ( $tem =~ /\(.*,\)\.vmdk/g ){
    verbose("ESX: Image name: $1\\n");
} else{
    verbose("ESX: The image does not find, abort\\n");
    exit 1;
}
$imageName = "$1" . " * " ;
open(TEMPLATESIZE,"ls -al $TEMPLATEDIR/$imageName |") or die "Error: "$ !
while ( my $templateSizeInfo = <TEMPLATESIZE> ){
    chomp $templateSizeInfo;
    if ( $templateSizeInfo =~ / ^ ( [\$+] ) \s* \([\w+]+\) \s* \([\d+]+\) \s* \([.]+\) \s* \([.]+\) \s* \([.]+\) - flat \. vmdk$/ ){
        verbose("ESX: $templateSizeInfo\\n");
        verbose("ESX: Image size of template: $4\\n");
        $TemplateSize = $4;
        $GBTemplateSize = substr $TemplateSize , 0 , 1 ;
        $GBTemplateSize = $GBTemplateSize +1;
        verbose("ESX: GBTemplateSize: $GBTemplateSize\\n");
    }
}
close(TEMPLATESIZE);
$old_datastore = getScalar("/host/$hostname/esx/datasstore", $OLDDATA_ROOT);
$new_datastore = getScalar("/host/$hostname/esx/datasstore", $DATA_ROOT);
out("ESX: Old datastore = $old_datastore\\n");
out("ESX: New datastore = $new_datastore\\n");
$old_vc_server = getScalar("/host/$hostname/esx/vc_server", $OLDDATA_ROOT);
$new_vc_server = getScalar("/host/$hostname/esx/vc_server", $DATA_ROOT);
out("ESX: Old vc_server = $old_vc_server\\n");
out("ESX: New vc_server = $new_vc_server\\n");
$old_username = getScalar("/host/$hostname/esx/username", $OLDDATA_ROOT);
$new_username = getScalar("/host/$hostname/esx/username", $DATA_ROOT);
out("ESX: Old username = $old_username\\n");
out("ESX: New username = $new_username\\n");
$old_password = getScalar("/host/$hostname/esx/password", $OLDDATA_ROOT);
$new_password = getScalar("/host/$hostname/esx/password", $DATA_ROOT);
out("ESX: Old password = $old_password\\n");
out("ESX: New password = $new_password\\n");
$old_vmfslabel = getScalar("/host/$hostname/esx/vmfslabel", $OLDDATA_ROOT);
$new_vmfslabel = getScalar("/host/$hostname/esx/vmfslabel", $DATA_ROOT);
out("ESX: Old vmfslabel = $old_vmfslabel\\n");
out("ESX: New vmfslabel = $new_vmfslabel\\n");
$old_vcuuid = getScalar("/host/$hostname/esx/vc_uuid", $OLDDATA_ROOT);
$new_vcuuid = getScalar("/host/$hostname/esx/vc_uuid", $DATA_ROOT);
out("ESX: Old vcuuid = $old_vcuuid\\n");
out("ESX: New vcuuid = $new_vcuuid\\n");
# Migrate
if ( ($old_datastore ne $new_datastore) or ($old_vc_server ne $new_vc_server)) {
    out("ESX: Migrate...\\n");
```bash
$VC_SERVER = $old_vc_server;
$ESX_HOST_USERNAME = $old_username;
$ESX_HOST_PASSWORD = $old_password;
$DATASTORE = $old_datastore;

out("ESX: Create the virtual machine directory...\n");

system("(vifs -server $new_vc_server -username $new_username
    -password $new_password -mkdir '$new_datastore'] $hostname.$PROJECT'");
print VMWARECOMMANDS "vifs -server $new_vc_server -username $new_username
    -password $new_password -mkdir '[' $new_datastore'] $hostname.$PROJECT'");

out("ESX: Removing old remote disk...\n");

system("(rm -rf $PROJECT/PATH/$hostname . vmx)");

out("ESX: Remove host $hostname.$PROJECT...\n");

system("# Delete former virtual machine
system("(vifs -force -server $VC_SERVER -username $ESX_HOST_USERNAME
    -password $ESX_HOST_PASSWORD -rm '[' $DATASTORE'] $hostname.$PROJECT'");

print VMWARECOMMANDS "vifs -force -server $VC_SERVER -username $ESX_HOST_USERNAME
    -password $ESX_HOST_PASSWORD -rm '[' $DATASTORE'] $hostname.$PROJECT'");

return 1;

} out("ESX: Create the virtual machine directory...\n");
```
```perl
my $newDiskSize = 0;
my $newFreeSpace = 0;
if ($sizeDeclaredFromScript > $GBTemplateSize) {
    out("ESX: Template found, copying...
    mkdir [\$DATASTORE] $hostname . $PROJECT
    my $old_disk = getScalar("/host/$hostname/size", $OLD_DATA_ROOT);
    my $new_disk = getScalar("/host/$hostname/size", $DATA_ROOT);
    out("ESX: Old disk = $old_disk\n    out("ESX: New disk = $new_disk\n    $old_nameserver = getScalar("/host/$hostname/nameserver", $OLD_DATA_ROOT);
    $new_nameserver = getScalar("/host/$hostname/nameserver", $DATA_ROOT);
    out("ESX: Old nameserver = $old_nameserver\n    out("ESX: New nameserver = $new_nameserver\n    $old_network = getScalar("/host/$hostname/network", $OLD_DATA_ROOT);
    $new_network = getScalar("/host/$hostname/network", $DATA_ROOT);
    $diff_network = getScalar("/host/$hostname/network", $DIFF)
    out("ESX: Old network = $old_network\n    out("ESX: New network = $new_network\n    out("ESX: Diff network = $diff_network\n
    if (stat("$PROJECT_PATH/$PROJECT/images/$hostname-flat.vmdk") ){
        $old_memory = getScalar("/host/$hostname/memory", $OLD_DATA_ROOT);
        $new_memory = getScalar("/host/$hostname/memory", $DATA_ROOT);
        out("ESX: Old memory = $old_memory\n        out("ESX: New memory = $new_memory\n        $old_users = getScalar("/host/$hostname/users", $OLD_DATA_ROOT);
        $new_users = getScalar("/host/$hostname/users", $DATA_ROOT);
        $diff_users = getScalar("/host/$hostname/users", $DIFF)
        out("ESX: Old users = $old_users\n        out("ESX: New users = $new_users\n        out("ESX: Diff users = $diff_users\n
        if (stat("$PROJECT_PATH/$PROJECT/$hostname.vmx") ){
            if ($old_memory != $new_memory) or ($diff_network)
                system("rm -rf $PROJECT_PATH/$PROJECT/$hostname.vmx");
            out("ESX: Removing old vmx file...
            system("vifs --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --mkdir [\$DATASTORE] $hostname.$PROJECT"");
            print VMWARECOMMANDS "vifs --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --mkdir [\$DATASTORE] $hostname.$PROJECT/\n            my $newDiskSize = 0;
            my $newFreeSpace = 0;
            if ($sizeDeclaredFromScript > $GBTemplateSize) {
                out("ESX: Template found, copying...
                mkdir [\$DATASTORE] $hostname . $PROJECT
                my $old_disk = getScalar("/host/$hostname/size", $OLD_DATA_ROOT);
                my $new_disk = getScalar("/host/$hostname/size", $DATA_ROOT);
                out("ESX: Old disk = $old_disk\n                out("ESX: New disk = $new_disk\n                $old_nameserver = getScalar("/host/$hostname/nameserver", $OLD_DATA_ROOT);
                $new_nameserver = getScalar("/host/$hostname/nameserver", $DATA_ROOT);
                out("ESX: Old nameserver = $old_nameserver\n                out("ESX: New nameserver = $new_nameserver\n                $old_network = getScalar("/host/$hostname/network", $OLD_DATA_ROOT);
                $new_network = getScalar("/host/$hostname/network", $DATA_ROOT);
                $diff_network = getScalar("/host/$hostname/network", $DIFF)
                out("ESX: Old network = $old_network\n                out("ESX: New network = $new_network\n                out("ESX: Diff network = $diff_network\n
                if (stat("$PROJECT_PATH/$PROJECT/images/$hostname-flat.vmdk") ){
                    $old_memory = getScalar("/host/$hostname/memory", $OLD_DATA_ROOT);
                    $new_memory = getScalar("/host/$hostname/memory", $DATA_ROOT);
                    out("ESX: Old memory = $old_memory\n                    out("ESX: New memory = $new_memory\n                    $old_users = getScalar("/host/$hostname/users", $OLD_DATA_ROOT);
                    $new_users = getScalar("/host/$hostname/users", $DATA_ROOT);
                    $diff_users = getScalar("/host/$hostname/users", $DIFF)
                    out("ESX: Old users = $old_users\n                    out("ESX: New users = $new_users\n                    out("ESX: Diff users = $diff_users\n
                    if (stat("$PROJECT_PATH/$PROJECT/$hostname.vmx") ){
                        if ($old_memory != $new_memory) or ($diff_network)
                            system("rm -rf $PROJECT_PATH/$PROJECT/$hostname.vmx");
                        out("ESX: Removing old vmx file...
                        system("vifs --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --mkdir [\$DATASTORE] $hostname.$PROJECT"");
                        print VMWARECOMMANDS "vifs --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --mkdir [\$DATASTORE] $hostname.$PROJECT/\n                        my $newDiskSize = 0;
                        my $newFreeSpace = 0;
                        if ($sizeDeclaredFromScript > $GBTemplateSize) {
                            out("ESX: Template found, copying...
                            mkdir [\$DATASTORE] $hostname . $PROJECT
                            my $old_disk = getScalar("/host/$hostname/size", $OLD_DATA_ROOT);
                            my $new_disk = getScalar("/host/$hostname/size", $DATA_ROOT);
                            out("ESX: Old disk = $old_disk\n                            out("ESX: New disk = $new_disk\n                            $old_nameserver = getScalar("/host/$hostname/nameserver", $OLD_DATA_ROOT);
                            $new_nameserver = getScalar("/host/$hostname/nameserver", $DATA_ROOT);
                            out("ESX: Old nameserver = $old_nameserver\n                            out("ESX: New nameserver = $new_nameserver\n                            $old_network = getScalar("/host/$hostname/network", $OLD_DATA_ROOT);
                            $new_network = getScalar("/host/$hostname/network", $DATA_ROOT);
                            $diff_network = getScalar("/host/$hostname/network", $DIFF)
                            out("ESX: Old network = $old_network\n                            out("ESX: New network = $new_network\n                            out("ESX: Diff network = $diff_network\n
                            if (stat("$PROJECT_PATH/$PROJECT/images/$hostname-flat.vmdk") ){
                                $old_memory = getScalar("/host/$hostname/memory", $OLD_DATA_ROOT);
                                $new_memory = getScalar("/host/$hostname/memory", $DATA_ROOT);
                                out("ESX: Old memory = $old_memory\n                                out("ESX: New memory = $new_memory\n                                $old_users = getScalar("/host/$hostname/users", $OLD_DATA_ROOT);
                                $new_users = getScalar("/host/$hostname/users", $DATA_ROOT);
                                $diff_users = getScalar("/host/$hostname/users", $DIFF)
                                out("ESX: Old users = $old_users\n                                out("ESX: New users = $new_users\n                                out("ESX: Diff users = $diff_users\n
                                if (stat("$PROJECT_PATH/$PROJECT/$hostname.vmx") ){
                                    if ($old_memory != $new_memory) or ($diff_network)
                                        system("rm -rf $PROJECT_PATH/$PROJECT/$hostname.vmx");
                                    out("ESX: Removing old vmx file...
                                    system("vifs --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --mkdir [\$DATASTORE] $hostname.$PROJECT"");
                                    print VMWARECOMMANDS "vifs --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --mkdir [\$DATASTORE] $hostname.$PROJECT/\n                                    my $newDiskSize = 0;
                                    my $newFreeSpace = 0;
                                    if ($sizeDeclaredFromScript > $GBTemplateSize) {
                                        out("ESX: Template found, copying...
                                        mkdir [\$DATASTORE] $hostname . $PROJECT
                                        my $old_disk = getScalar("/host/$hostname/size", $OLD_DATA_ROOT);
                                        my $new_disk = getScalar("/host/$hostname/size", $DATA_ROOT);
                                        out("ESX: Old disk = $old_disk\n                                        out("ESX: New disk = $new_disk\n                                        $old_nameserver = getScalar("/host/$hostname/nameserver", $OLD_DATA_ROOT);
                                        $new_nameserver = getScalar("/host/$hostname/nameserver", $DATA_ROOT);
                                        out("ESX: Old nameserver = $old_nameserver\n                                        out("ESX: New nameserver = $new_nameserver\n                                        $old_network = getScalar("/host/$hostname/network", $OLD_DATA_ROOT);
                                        $new_network = getScalar("/host/$hostname/network", $DATA_ROOT);
                                        $diff_network = getScalar("/host/$hostname/network", $DIFF)
                                        out("ESX: Old network = $old_network\n                                        out("ESX: New network = $new_network\n                                        out("ESX: Diff network = $diff_network\n```

123
} else{
    # No change for disk
    return 1;
}

} # Deal with disk and IP

DISKF:
if( stat("{PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk") ){
    if( $diff_users or ($old_disk != $new_disk) or ($old_nameserver != $new_nameserver) or $diff_network ){
        verbose("ESX: The disk is {hostname}−flat.vmdk...\n" );
        out("ESX: Removing old local disk...\n" );
        system("rm -rf {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk");
        system("rm -rf {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk");
    out("ESX: Removing old remote disk...\n" );
    system("vifs --server {VC_SERVER} --username
      {ESX_HOST_USERNAME} --password {ESX_HOST_PASSWORD} --get '{[DATASTORE] $hostname.}{PROJECT/{hostname}−flat.vmdk}' {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk");
    print VMWARECOMMANDS "vifs --server {VC_SERVER} --username
      {ESX_HOST_USERNAME} --password {ESX_HOST_PASSWORD} --get '{[DATASTORE] $hostname.}{PROJECT/{hostname}−flat.vmdk}' {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk";
    system("vifs --server {VC_SERVER} --username
      {ESX_HOST_USERNAME} --password {ESX_HOST_PASSWORD} --get '{[DATASTORE] $hostname.}{PROJECT/{hostname}−flat.vmdk}' {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk");
    out("ESX: Removing old remote disk...\n" );
    system("vifs --server {VC_SERVER} --username
      {ESX_HOST_USERNAME} --password {ESX_HOST_PASSWORD} --get '{[DATASTORE] $hostname.}{PROJECT/{hostname}−flat.vmdk}' {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk" --force\n" );
    print VMWARECOMMANDS "vifs --server {VC_SERVER} --username
      {ESX_HOST_USERNAME} --password {ESX_HOST_PASSWORD} --get '{[DATASTORE] $hostname.}{PROJECT/{hostname}−flat.vmdk}' {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk" --force\n" );
    system("vifs --server {VC_SERVER} --username
      {ESX_HOST_USERNAME} --password {ESX_HOST_PASSWORD} --get '{[DATASTORE] $hostname.}{PROJECT/{hostname}−flat.vmdk}' {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk" --force\n" );
    print VMWARECOMMANDS "vifs --server {VC_SERVER} --username
      {ESX_HOST_USERNAME} --password {ESX_HOST_PASSWORD} --get '{[DATASTORE] $hostname.}{PROJECT/{hostname}−flat.vmdk}' {PROJECT_PATH}/{PROJECT/images}/{hostname}−flat.vmdk" --force\n" );
    if( $old_disk == $new_disk ){
        return 1;
    } else{
        out("ESX: Change disk size...\n" );
    }
    }
    # Build virtual machine
}
if (not stat("$PROJECT_PATH/$PROJECT/images/${hostname}-flat.vmdk")) {
    system("vmware-vdiskmanager -r $TEMPLATEDIR/Stem -t 4 $PROJECT_PATH/$PROJECT/images/${hostname}.vmdk");
    print VMWARECOMMANDS "vmware-vdiskmanager -r $TEMPLATEDIR/Stem -t 4 $PROJECT_PATH/$PROJECT/images/${hostname}.vmdk"
};

# Change disk size
# Add the unit to the newDiskSize, use 'MB' unitive
if ($diskSizeUnit eq "GB") {
    $newDiskSize = $sizeDeclaredFromScript * 1024;
} elsif ($diskSizeUnit eq "MB") {
    $newDiskSize = $sizeDeclaredFromScript;
}

if ($freeSpace != 0) {
    # Add the unit to the newDiskSize, use 'MB' unitive
    if ($freeSpaceSizeUnit eq "GB") {
        $newFreeSpace = $freeSpace * 1024;
    } elsif ($freeSpaceSizeUnit eq "MB") {
        $newFreeSpace = $freeSpace;
    }
    $newDiskSize = $newDiskSize + $newFreeSpace;
} else {
    # freeSpace is zero, but this section is used to upgrade
    and normal construction
    verbose("ESX: freeSpace: $freeSpace\n");
}

# Add the unit to the newDiskSize, use 'MB' unitive
$newDiskSize = $newDiskSize . "MB";

system("vmware-vdiskmanager -x $newDiskSize $PROJECT_PATH/$PROJECT/images/${hostname}.vmdk");
print VMWARECOMMANDS "vmware-vdiskmanager -x $newDiskSize $PROJECT_PATH/$PROJECT/images/${hostname}.vmdk"

elseif ($sizeDeclaredFromScript < $GBTemplateSize ){
    out("ESX: Disk size declared in the project is small, expanding to template size $GBTemplateSize automatically...\n")
    out("ESX: The template is found, copying...\n")
}

# For upgrade
if (stat("$PROJECT_PATH/$PROJECT/images/${hostname}-flat.vmdk")) {
    $old_memory = getScalar("/host/${hostname}/memory", SOLDATA_ROOT);
    $new_memory = getScalar("/host/${hostname}/memory", SDATA_ROOT);
    out("ESX: Old memory = $old_memory\n");
    out("ESX: New memory = $new_memory\n");
    $old_users = getScalar("/host/${hostname}/users", SOLDATA_ROOT);
    $new_users = getScalar("/host/${hostname}/users", SDATA_ROOT);
    $old_users = getScalar("/host/${hostname}/users", SDIFF);
    out("ESX: Old users = $old_users\n");
    out("ESX: New users = $new_users\n");
    $old_disk = getScalar("/host/${hostname}/size", SOLDATA_ROOT);
    $new_disk = getScalar("/host/${hostname}/size", SDATA_ROOT);
    out("ESX: Old disk = $old_disk\n");
    out("ESX: New disk = $new_disk\n");
    $old_nameserver = getScalar("/host/${hostname}/nameserver", SOLDATA_ROOT);
    $new_nameserver = getScalar("/host/${hostname}/nameserver", SDATA_ROOT);

    out("ESX: Old nameserver = $old_nameserver\n");
    out("ESX: New nameserver = $new_nameserver\n");
}
out("ESX: Old nameserver = \$old_nameserver\n")
out("ESX: New nameserver = \$new_nameserver\n")
\$old_network = getScalar("/host/\$hostname/network", 
\$OLD_DATA_ROOT)
\$new_network = getScalar("/host/\$hostname/network", 
\$DATA_ROOT)
\$diff_network = getScalar("/host/\$hostname/network",\$DIFF)
out("ESX: Old network = \$old_network\n")
out("ESX: New network = \$new_network\n")
out("ESX: Diff network = \$diff_network\n")
}

# If machine is created, a copy should be in local
# Deal with memory
if( stat("\$PROJECT PATH/\$PROJECT/\$hostname . vmx") ){
  if( \$old_memory != \$new_memory or \$diff_network ){
    out("ESX: Removing old vmx file . . .\n")
    system("\rm -rf \$PROJECT PATH/\$PROJECT/\$hostname . vmx")
    out("ESX: Removing old remote vmx file . . .\n")
    system("vifs --server \$VC_SERVER --username
             \$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD
             --rm ['\$$DATASTORE\$'] \$hostname.\$PROJECT/\$hostname .
             vmx' --force");
  }
}

# Deal with disk as well
if( \$diff_users or (\$old_disk != \$new_disk) or (\n     \$old_nameserver != \$new_nameserver) or \n     \$diff_network ){
goto DISKM;
} else{
  # No change for disk
  return 1;
}
}

# Deal with disk
DISKM:
if( stat("\$PROJECT PATH/\$PROJECT/images/\$\{hostname\}-flat.vmdk") ){
  if( \$diff_users or (\$old_disk != \$new_disk) or (\n     \$old_nameserver != \$new_nameserver) or \$diff_network ){
    verbose("ESX: The disk is \$\{hostname\}-flat.vmdk . . .\n")
    out("ESX: Removing old local disk . . .\n")
    system("\rm -rf \$PROJECT PATH/\$PROJECT/images/\$\{hostname\}-flat.vmdk")
    out("ESX: Downloading old remote disk . . .\n")
    system("vifs --server \$VC_SERVER --username
             \$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD
             --get ['\$$DATASTORE\$'] \$hostname.\$PROJECT/\$\{hostname\}-flat.vmdk'
             \$PROJECT/\$\{hostname\}-flat.vmdk")
  }
  print \nVMWARE_COMMANDS "vifs --server \$VC_SERVER --username
             \$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD
             --get ['\$$DATASTORE\$'] \$hostname.\$PROJECT/\$\{hostname\}-flat.vmdk'
             \$PROJECT/\$\{hostname\}-flat.vmdk")
  out("ESX: Downloading old remote disk . . .\n")
  system("vifs --server \$VC_SERVER --username
             \$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD
             --get ['\$$DATASTORE\$'] \$hostname.\$PROJECT/\$\{hostname\}-flat.vmdk'
             \$PROJECT/\$\{hostname\}-flat.vmdk")
}

vmdk' '$PROJECT
PATH/$PROJECT/images/$hostname . vmdk
');

print VMWARECOMMANDS "vifs —server SVC_SERVER —
username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —get '[SDATASTORE] $hostname.
$PROJECT/$hostname.vmdk' '$PROJECT/SPROJECT/images/$hostname.vmdk";

out("ESX: Removing old remote disk...\n");

system("vifs —server SVC_SERVER —
username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm [SDATASTORE] $hostname. $PROJECT/$hostname —flat.vmdk —force\n");

print VMWARECOMMANDS "vifs —server SVC_SERVER —
username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm [SDATASTORE] $hostname.
$PROJECT/$hostname —flat.vmdk —force\n";

print VMWARECOMMANDS "vifs —server SVC_SERVER —
username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm [SDATASTORE] $hostname.
$PROJECT/$hostname.vmdk —force\n";

print VMWARECOMMANDS "vifs —server SVC_SERVER —
username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm [SDATASTORE] $hostname.
$PROJECT/$hostname —flat.vmdk —force\n";

print VMWARECOMMANDS "vifs —server SVC_SERVER —
username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm [SDATASTORE] $hostname.
$PROJECT/$hostname —flat.vmdk —force\n";

if ( old_disk == new_disk ){
    return 1;
} else {
    out("ESX: Change disk size...\n");
}

}

if ( not stat("$PROJECT/SPROJECT/images/${hostname}—flat.vmdk") ){
    system("vmware—vdiskmanager —r $TEMPLATEDIR/ —t 4
$PROJECT/SPROJECT/images/$hostname.vmdk" );

print VMWARECOMMANDS "vmware—vdiskmanager —r $TEMPLATEDIR/ —t 4
$PROJECT/SPROJECT/images/$hostname.vmdk" ;

# Change disk size
# Use 'MB' unit
$GBTemplateSize = $GBTemplateSize * 1024;

if ( $freeSpace != 0 ){
    # Add the unit to the newDiskSize, use 'MB' unitive
    if ( $freeSpaceSizeUnit eq "GB" ){
        $newFreeSpace = $freeSpace * 1024;
    } elsif ( $freeSpaceSizeUnit eq "MB" ){
        $newFreeSpace = $freeSpace;
    }
    $GBTemplateSize = $GBTemplateSize + $newFreeSpace;
} else{
    verbose("ESX: freeSpace: $freeSpace\n");
}

# Add the unit to the GBTemplateSize, use 'MB' unitive
$GBTemplateSize = $GBTemplateSize + $MB;

verbose("ESX: GBTemplateSize: $GBTemplateSize\n");

system("vmware—vdiskmanager —x $GBTemplateSize $PROJECT/SPROJECT/images/$hostname.vmdk" );

print VMWARECOMMANDS "vmware—vdiskmanager —x $GBTemplateSize
$PROJECT/SPROJECT/images/$hostname.vmdk" ;

} else{
    out("ESX: Template found, copying...Equally disk size...\n");
    # For upgrade
    if ( stat("$PROJECT/SPROJECT/SPROJECT/images/$hostname—flat.vmdk"") ){
        }
$old_memory = getScalar("/host/$hostname/memory", SOLD_DATA_ROOT);
$new_memory = getScalar("/host/$hostname/memory", $DATA_ROOT);
out("ESX: Old memory = $old_memory\n");
$out("ESX: New memory = $new_memory\n");
$old_users = getScalar("/host/$hostname/users", SOLD_DATA_ROOT);
$new_users = getScalar("/host/$hostname/users", $DATA_ROOT);
$diff_users = getScalar("/host/$hostname/users", $DIFF);
out("ESX: Old users = $old_users\n");
$out("ESX: New users = $new_users\n");
$out("ESX: Diff users = $diff_users\n");
$old_disk = getScalar("/host/$hostname/size", SOLD_DATA_ROOT);
$new_disk = getScalar("/host/$hostname/size", $DATA_ROOT);
$out("ESX: Old disk = $old_disk\n");
$out("ESX: New disk = $new_disk\n");
$old_nameserver = getScalar("/host/$hostname/nameserver", SOLD_DATA_ROOT);
$new_nameserver = getScalar("/host/$hostname/nameserver", $DATA_ROOT);
$out("ESX: Old nameserver = $old_nameserver\n");
$out("ESX: New nameserver = $new_nameserver\n");
$old_network = getScalar("/host/$hostname/network", SOLD_DATA_ROOT);
$new_network = getScalar("/host/$hostname/network", $DATA_ROOT);
$diff_network = getScalar("/host/$hostname/network", $DIFF);
$out("ESX: Old network = $old_network\n");
$out("ESX: New network = $new_network\n");
$out("ESX: Diff network = $diff_network\n");
}

# If machine is created, a copy should be in local
# Deal with memory
if (stat("$PROJECT_PATH/$PROJECT/$hostname.vmx")){
  if($old_memory != $new_memory or $diff_network){
    out("ESX: Removing old vmx file ...\n");
    system("rm -rf $PROJECT_PATH/$PROJECT/$hostname.vmx");
    out("ESX: Removing old remote vmx file ...\n");
    system("vifs --server SVC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --rm [DATASTORE] $hostname.$PROJECT/$hostname.vmx' --force\n");
    print VMwareCOMMANDS "vifs --server SVC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --rm [DATASTORE] $hostname.$PROJECT/$hostname.vmx' --force\n";
  # Deal with disk as well
  if ( $diff_users or ($old_disk != $new_disk) ) or ($old_nameserver != $new_nameserver) or ($diff_network ){
    goto DISKL
  } else{
    # No change for disk
    return 1;
  }
}
# Deal with disk
DISKL:
if (stat("$PROJECT_PATH/$PROJECT/images/$hostname-flat.vmdk" )){


if ( ($old_memory != $new_memory) or ($diff_users) or ($old_disk != $new_disk) or ($old_nameserver != $new_nameserver) or $diff_network ) {

    verbose("ESX: The disk is $hostname−flat.vmdk...\n")
    
    out("ESX: Removing old local disk...\n")
    system("rm −rf $PROJECT PATH/$PROJECT/images/$hostname−flat.vmdk")
    
    out("ESX: Downloading old remote disk...\n")
    system("vifs −server SVC_SERVER —username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —get '[$DATASTORE] $hostname.$PROJECT/$hostname−flat.vmdk' $PROJECT PATH/$PROJECT/images/$hostname−flat.vmdk")
    

print "VMWARECOMMANDS "vifs —server SVC_SERVER —username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —get '[$DATASTORE] $hostname.$PROJECT/$hostname−flat.vmdk' $PROJECT PATH/$PROJECT/images/$hostname−flat.vmdk")

print "VMWARECOMMANDS "vifs —server SVC_SERVER —username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —get '[$DATASTORE] $hostname.$PROJECT/$hostname−flat.vmdk' $PROJECT PATH/$PROJECT/images/$hostname−flat.vmdk")

out("ESX: Removing old remote disk...\n")

system("vifs —server SVC_SERVER —username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm '[$DATASTORE] $hostname.$PROJECT/$hostname−flat.vmdk' —force")

print "VMWARECOMMANDS "vifs —server SVC_SERVER —username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —get '[$DATASTORE] $hostname.$PROJECT/$hostname−flat.vmdk' $PROJECT PATH/$PROJECT/images/$hostname−flat.vmdk")

system("vifs —server SVC_SERVER —username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm '[$DATASTORE] $hostname.$PROJECT/$hostname−flat.vmdk' —force")

print "VMWARECOMMANDS "vifs —server SVC_SERVER —username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm '[$DATASTORE] $hostname.$PROJECT/$hostname−flat.vmdk' —force")

print "VMWARECOMMANDS "vifs —server SVC_SERVER —username $ESX_HOST_USERNAME —password $ESX_HOST_PASSWORD —rm '[$DATASTORE] $hostname.$PROJECT/$hostname−flat.vmdk' —force")

if ( $old_disk == $new_disk ) {

    return 1;

} else{

    out("ESX: Change disk size...\n")
}
}

} else{

    out("ESX: Change disk size...\n")
}

if ( not stat("$PROJECT PATH/$PROJECT/images/$hostname−flat.vmdk") ) {

    system("vmware−vdiskmanager −r $TEMPLATEDIR/Stem −t 4 $PROJECT PATH/$PROJECT/images/$hostname.vmdk")

    print "VMWARECOMMANDS "vmware−vdiskmanager −r $TEMPLATEDIR/Stem −t 4 $PROJECT PATH/$PROJECT/images/$hostname.vmdk")

} # Change disk size

# Use "MB" unit

$GBTemplateSize = $GBTemplateSize * 1024;
if ($freeSpace != 0) {
  # Add the unit to the newDiskSize, use 'MB' unitive
  if ($freeSpaceSizeUnit eq "GB") {
    $newFreeSpace = $freeSpace * 1024;
  } elsif ($freeSpaceSizeUnit eq "MB") {
    $newFreeSpace = $freeSpace;
  }
  $GBTemplateSize = $GBTemplateSize + $newFreeSpace;
} else {
  verbose("ESX: freeSpace: $freeSpace\n");
  $GBTemplateSize = $GBTemplateSize . "MB";
  verbose("ESX: GBTemplateSize: $GBTemplateSize\n");
  system("vmware-vdiskmanager -x $GBTemplateSize $PROJECT_PATH/PROJECT/images/$hostname.vmdk");
}

sub esx_mountFilesystem {
  my $hostname = s\[0\];
  if (getScalar("/host/$hostname/esx")) {
    out("ESX: ESX plugin is mounting $hostname on $MOUNTDIR\n");
    system("vmware-mount $PROJECT_PATH/PROJECT/images/$hostname.vmdk $MOUNTDIR");
    print VMWARECOMMANDS "vmware-vdiskmanager -x $GBTemplateSize
  $PROJECT_PATH/PROJECT/images/$hostname.vmdk\n";
  }
  return 1;
}

sub esx_createStartStopScripts {
  my $hostname = s\[0\];
  if (getScalar("/host/$hostname/esx")) {
    $bo = getScalar("/host/$hostname/boot_order");
    $bo = 99 unless $bo;
    my $VC_SERVER = getScalar("/host/$hostname/esx/vc_server");
    $VC_SERVER = $DEFAULT_VC_SERVER unless $VC_SERVER;
    my $ESX_HOSTUSERNAME = getScalar("/host/$hostname/esx/username");
    $ESX_HOSTUSERNAME = $DEFAULT_ESX_HOSTUSERNAME unless $ESX_HOSTUSERNAME;
    $ESX_HOSTUSERNAME = getScalar("/host/$hostname/esx/password");
    $ESX_HOSTPASSWORD = $DEFAULT_ESX_HOSTPASSWORD unless $ESX_HOSTPASSWORD;
    my $VMFSLABEL = getScalar("/host/$hostname/esx/vmfslabel");
    $VMFSLABEL = $DEFAULT_VMFSLABEL unless $VMFSLABEL;
    $ESX_HOSTUSERNAME = $ESX_HOSTUSERNAME . "\n";
    $ESX_HOSTPASSWORD = $ESX_HOSTPASSWORD . "\n";
    my $vmfslabel = getScalar("/host/$hostname/esx/vmfslabel");
    $vmfslabel = $DEFAULT_VMFSLABEL unless $vmfslabel;
    my $vc_server = getScalar("/host/$hostname/esx/vc_server");
    $vc_server = $DEFAULT_VC_SERVER unless $vc_server;
    my $boot_order = getScalar("/host/$hostname/esx/boot_order");
    $boot_order = 99 unless $boot_order;
    out("ESX: Creating: $PROJECT_PATH/PROJECT/start_${bo}.$hostname.sh\n");
    open(START,">$PROJECT_PATH/PROJECT/start_${bo}.$hostname.sh"){
      print START "#!/bin/bash\n";
      print START "echo ""Starting $hostname...\n";
      print START "vmware-cmd --server $vc_server --username $ESX_HOSTUSERNAME --password $ESX_HOSTPASSWORD --s register /vmfs/volumes/$vmfslabel/$hostname.$PROJECT/PROJECT/PROJECT/vm/Register.vm REGISTER\n";
      print START "vmware-cmd --server $vc_server --username $ESX_HOSTUSERNAME --password $ESX_HOSTPASSWORD --s register /vmfs/volumes/$vmfslabel/$hostname.$PROJECT/PROJECT/vm/Register.vm REGISTER\n"
    }
  }
  return 1;
}
#!/bin/bash

# Check the virtual machine has been registered or not
print START "if [ ""-eq ""$PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMREGISTERSTATUS' then
print START ""Virtual machine $hostname.$PROJECT has been started, abort...
print START "exit 1"
print START "fi"
# Make sure it is registered
print START "while [..."; do
print START "cat $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMREGISTERSTATUS' = ""register() =1" ]; do

print START "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --s register /vmfs/volumes/$VMFSLABEL/$hostname.$PROJECT/$hostname.vmx\\ $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMREGISTERSTATUS'"
print START "echo "Register the virtual machine $hostname.$PROJECT again"
print START "done"

# Start the virtual machine
print START "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/$hostname.$PROJECT/$hostname.vmx start soft &"

close (START);
system ("chmod +x $PROJECT_PATH/$PROJECT/start.$bo.$hostname.sh")

out ("ESX : Creating : $PROJECT_PATH/$PROJECT/stop.$bo.$hostname.sh")

open (STOP) "=$PROJECT_PATH/$PROJECT/stop.$bo.$hostname.sh";
print STOP "#!/bin/bash"
print STOP "echo \"Stopping $hostname...\""
print STOP "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/$hostname.$PROJECT/$hostname.vmx stop soft &"
# It is better to sleep a while after soft stop
print STOP "sleep 10s"
# Get status of virtual machine
print STOP "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/$hostname.$PROJECT/$hostname.$PROJECT.VMLIVE"
print STOP "if [..."; do
print STOP "cat $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMLIVE' = ""getstate() = off" ]; do

print STOP "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/$hostname.$PROJECT/$hostname.$PROJECT.VMLIVE $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMLIVE'"
print STOP "echo \"Virtual machine $hostname.$PROJECT has been stopped, abort...
print STOP "exit 1"
print STOP "fi"
# Make sure it is stopped
print STOP "while [..."; do
print STOP "cat $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMLIVE' = ""getstate() = off" ]; do

print STOP "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/$hostname.$PROJECT/$hostname.$PROJECT.VMLIVE $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMLIVE'"
print STOP "echo \"Stopping the virtual machine $hostname.$PROJECT again\"
print STOP "done"

# Make sure it is stopped
print STOP "while [..."; do
print STOP "cat $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMLIVE' = ""getstate() = on" ]; do

print STOP "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/$hostname.$PROJECT/$hostname.$PROJECT.VMLIVE $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMLIVE'"
print STOP "echo \"The stop takes some time\"
print STOP "sleep 10s"
print STOP "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/$hostname.$PROJECT/$hostname.$PROJECT.VMLIVE $PROJECT_PATH/$PROJECT/$hostname.$PROJECT.VMLIVE'"
print STOP "done"

# Unregister the virtual machine
print STOP "vmware-cmd --server $VC --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --s unregister $PROJECT_PATH/$PROJECT/$hostname.$PROJECT/$hostname.$PROJECT.VMREGISTERSTATUS'"
close (STOP);
system("chmod +x $PROJECT_PATH/stop_${bo}_${hostname}.sh");
esx_writeVMXandRegister($hostname);
return 1;
}
}

sub esx_importFilesystem {
    my $host = $_[0];
    my $project = ($_[1];
    my $directory = $_[2];
    if ( getScalar("/host/$host/esx") ){
        verbose("ESX: Triggled the esx_importFilesystem()...
";
        out("ESX: Import the file system from $directory to $PROJECT_PATH/";
        system("cp -v $directory/$host.vmdk $PROJECT_PATH/$project/images/$host.vmdk");
        system("cp -v $directory/\{host\}−flat.vmdk $PROJECT_PATH/$project/images/\{host\}−flat.vmdk");
        return 1;
    }
}

sub esx_exportFilesystem {
    my $host = $_[0];
    my $project = ($_[1];
    my $directory = $_[2];
    if ( getScalar("/host/$host/esx") ){
        verbose("ESX: Triggled the esx_exportFilesystem()...
";
        out("ESX: Export the file system from $PROJECT_PATH/$project to $directory...
";
        system("cp -v $PROJECT_PATH/$project/images/$host.vmdk $directory/$host.vmdk");
        system("cp -v $PROJECT_PATH/$project/images/\{host\}−flat.vmdk $directory/\{host\}−flat.vmdk");
        return 1;
    }
}

sub esx_getImportExportFiles {
    my $host = $_[0];
    my $project = ($_[1];
    my %targets;
    if ( getScalar("/host/$host/esx") ){
        %targets["$host.vmdk"] = "$PROJECT_PATH/$project/images/$host.vmdk";
        %targets["$host−flat.vmdk"] = "$PROJECT_PATH/$project/images/\{host\}−flat.vmdk";
        return %targets;
    }
}

sub esx_writeVMXandRegister {
    my $hostname = $_[0];
    if ( getScalar("/host/$hostname/esx") ){
        my $memory = getScalar("/host/$hostname/memory");
        $memory = $DEFAULS{MEMORY} unless $memory;
        $memory =~ s/\d+M/$1/;
        my $VC_UUID = getScalar("/host/$hostname/esx/vc_uuid");
        $VC_UUID = $DEFAULS{VC_UUID} unless $VC_UUID;
        verbose("ESX: VC_UUID: $VC_UUID\n");
        my $DATASTORE = getScalar("/host/$hostname/esx/datastore");
        $DATASTORE = $DEFAULS{DATASTORE} unless $DATASTORE;
        verbose("ESX: DATASTORE: $DATASTORE\n");
        my $VC_SERVER = getScalar("/host/$hostname/esx/vc_server");
        $VC_SERVER = $DEFAULS{VC_SERVER} unless $VC_SERVER;
        verbose("ESX: VC_SERVER: $VC_SERVER\n");
        my $ESX_HOST_USERNAME = getScalar("/host/$hostname/esx/username");

$ESX_HOST_USERNAME = $DEFAULT_ESX_HOST_USERNAME unless $ESX_HOST_USERNAME;
verbose("ESX : ESX_HOST_USERNAME: $ESX_HOST_USERNAME\n");
my $ESX_HOST_PASSWORD = getScalar("/host/$hostname/esx/password");
$ESX_HOST_PASSWORD = $DEFAULT_ESX_HOST_PASSWORD unless $ESX_HOST_PASSWORD;
verbose("ESX : ESX_HOST_PASSWORD: $ESX_HOST_PASSWORD\n");
$VMFSLABEL = getScalar("/host/$hostname/esx/vmfslabel");
$VMFSLABEL = $DEFAULT_VMFSLABEL unless $VMFSLABEL;
my $GUESTOS = getScalar("/host/$hostname/esx/guestos");
$GUESTOS = $DEFAULT_GUESTOS unless $GUESTOS;
verbose("ESX : GUESTOS: $GUESTOS\n");
$HOST = $DEFAULT_GUESTOS unless $GUESTOS;

open(VMX, ">$PROJECT/Path/PROJECT/$hostname.vmx");
print VMX "# VM file generated by MKN VMware plugin (\nVMWARE_PLUGIN_VERSION)\n";
print VMX " .encoding = '\"UTF-8\"\n";
print VMX " .config.version = '\"8\"\n";
# Hardware Virtualization
print VMX "virtualHW. version = '\"7\"\n";
# PCI Bridge
print VMX "pciBridge0.present = \"TRUE\"\n";
print VMX "pciBridge4.present = \"TRUE\"\n";
print VMX "pciBridge4.virtualDev = \"pciRootPort\"\n";
print VMX "pciBridge4.functions = \"81\"\n";
print VMX "pciBridge5.present = \"TRUE\"\n";
print VMX "pciBridge5.virtualDev = \"pciRootPort\"\n";
print VMX "pciBridge5.functions = \"81\"\n";
print VMX "pciBridge6.present = \"TRUE\"\n";
print VMX "pciBridge6.virtualDev = \"pciRootPort\"\n";
print VMX "pciBridge6.functions = \"81\"\n";
print VMX "pciBridge7.present = \"TRUE\"\n";
print VMX "pciBridge7.virtualDev = \"pciRootPort\"\n";
print VMX "pciBridge7.functions = \"81\"\n";
# Virtual Machine Communication Interface (VMCI)
print VMX "vmci0.present = \"TRUE\"\n";
print VMX "vnram = \"$hostname.nvram\n";
print VMX "virtualHW.productCompatibility = \"hosted\"\n";
print VMX "powerType.powerOff = \"soft\"\n";
print VMX "powerType.powerOn = \"hard\"\n";
print VMX "powerType.suspend = \"hard\"\n";
print VMX "powerType.reset = \"soft\"\n";
print VMX "displayName = \"$hostname.SPROJECT\"\n";
print VMX "extendedConfigFile = \"$hostname.vmxf\n";
print VMX "floppy0.present = \"TRUE\"\n";
print VMX "scsi0.present = \"TRUE\"\n";
print VMX "scsi0.sharedBus = \"none\"\n";
print VMX "scsi0.virtualDev = \"lsilogic\"\n";
print VMX "memsize = "$memory\n";
# Storage Devices
print VMX "scsi0:1.present = \"TRUE\"\n";
print VMX "scsi0:1.fileName = \"$hostname.vmdk\"\n";
print VMX "scsi0:1.deviceType = \"scsi\"\n";
print VMX "id1:0.present = \"TRUE\"\n";
print VMX "id1:0.clientDevice = \"TRUE\"\n";
print VMX "id1:0.deviceType = \"cdrom\"\n";
print VMX "id1:0.startConnected = \"FALSE\"\n";
print VMX "floppy0.startConnected = \"FALSE\"\n";
print VMX "floppy0.fileName = \"\n";
print VMX "ESX_HOST_PASSWORD: $ESX_HOST_PASSWORD\n";
# OS Type
print VMX "guestOS = \"$GUESTOS\"\n";
# print VMX "guestOS = \"debian5−64\"\n";
print VMX "uuid.location = \""\n";
print VMX "uuid.bios = \""\n";
print VMX "vc.uuid = \""\n";
print VMX "vmci0.id = \""\n";
print VMX "cleanShutdown = \""\n";
print VMX "replay.supported = \""\n";
print VMX "sched.swap.derivedName = \""\n";
print VMX "replay.filename = \""\n";
print VMX "sci0.1.redo = \""\n";
print VMX "pciBridge0.pciSlotNumber = \""\n";
print VMX "pciBridge4.pciSlotNumber = \""\n";
print VMX "pciBridge5.pciSlotNumber = \""\n";
print VMX "pciBridge6.pciSlotNumber = \""\n";
print VMX "pciBridge7.pciSlotNumber = \""\n";
print VMX "sci0.pciSlotNumber = \""\n";
print VMX "vmci0.pciSlotNumber = \""\n";
print VMX "vmotion.checkpointFBSize = \""\n";

# CPU Information
print VMX "hostCPUID.0 = \""\n";
print VMX "hostCPUID.1 = \""\n";
print VMX "hostCPUID.80000001 = \""\n";
print VMX "guestCPUID.0 = \""\n";
print VMX "guestCPUID.1 = \""\n";
print VMX "guestCPUID.80000001 = \""\n";
print VMX "userCPUID.0 = \""\n";
print VMX "userCPUID.1 = \""\n";
print VMX "userCPUID.80000001 = \""\n";
print VMX "evcCompatibilityMode = \""\n";
print VMX "tools.syncTime = \""\n";

# Network interfaces and which virtual switch a host should connect
my %interfaces = getHash("/host/$hostname/network");
if ( %interfaces ) {
  my $interface;
  my $device;
  my $PORTGROUP_NAME;
  foreach $interface ( keys %interfaces ){
    $device = $interface;
    verbose("ESX: Device: $device...
    $interface = "$eth/ethernet/"
    out("ESX: Writing interface $interface\n"");
    print VMX "Sinterface.present = \""\n"
    print VMX "Sinterface.virtualDev = \"e1000\"
    if ( not $interfaces{$device} ){"switch"} {
      $PORTGROUP_NAME = "$DEFAULT_PORTGROUP_NAME"
    } else{
      $PORTGROUP_NAME = "$interfaces{$device}('switch').$PROJECT"
    }
    out("ESX: Interface $interface is going to connect to virtual switch $PORTGROUP_NAME\n"");
    print VMX "Sinterface.networkName = \"$PORTGROUP_NAME\n"
    print VMX "Sinterface.addressType = \"generated\n"
    print VMX "Sinterface.generatedAddress = \"\n"
    print VMX "Sinterface.pciSlotNumber = \"32\n"
    print VMX "Sinterface.generatedAddressOffset = \"0\n"
  }
}

}
my $old_users;
my $new_users;
my $diff_users;
my $old_disk;
my $new_disk;
my $old_nameserver;
my $new_nameserver;
my $old_network;
my $new_network;
my $diff_network;
my $old_datastore;
my $new_datastore;
my $old_vcs_server;
my $new_vcs_server;
my $old_username;
my $new_username;
my $old_password;
my $new_password;
my $old_vmfs_label;
my $new_vmfs_label;
my $old_vc_uuid;
my $new_vc_uuid;

if ($OLD_DATA_ROOT) {
  $old_memory = getScalar("/host/$hostname/memory",$OLD_DATA_ROOT);
  $old_disk = getScalar("/host/$hostname:size",$OLD_DATA_ROOT);
  $old_users = getScalar("/host/$hostname/users",$OLD_DATA_ROOT);
  $old_nameserver = getScalar("/host/$hostname/nameserver",$OLD_DATA_ROOT);
  $old_network = getScalar("/host/$hostname/network",$OLD_DATA_ROOT);
  $old_datastore = getScalar("/host/$hostname/esx/datastore",$OLD_DATA_ROOT);
  $old_vcs_server = getScalar("/host/$hostname/esx/vcs_server",$OLD_DATA_ROOT);
  $old_username = getScalar("/host/$hostname/esx/username",$OLD_DATA_ROOT);
  $old_password = getScalar("/host/$hostname/esx/password",$OLD_DATA_ROOT);
  $old_vmfs_label = getScalar("/host/$hostname/esx/vmfslabel",$OLD_DATA_ROOT);
  $old_vc_uuid = getScalar("/host/$hostname/esx/vc_uuid",$OLD_DATA_ROOT);
}

if ($DATA_ROOT) {
  $new_memory = getScalar("/host/$hostname/memory",$DATA_ROOT);
  $new_disk = getScalar("/host/$hostname:size",$DATA_ROOT);
  $new_users = getScalar("/host/$hostname/users",$DATA_ROOT);
  $new_nameserver = getScalar("/host/$hostname/nameserver",$DATA_ROOT);
  $new_network = getScalar("/host/$hostname/network",$DATA_ROOT);
  $new_datastore = getScalar("/host/$hostname/esx/datastore",$DATA_ROOT);
  $new_vcs_server = getScalar("/host/$hostname/esx/vcs_server",$DATA_ROOT);
  $new_username = getScalar("/host/$hostname/esx/username",$DATA_ROOT);
  $new_password = getScalar("/host/$hostname/esx/password",$DATA_ROOT);
  $new_vmfs_label = getScalar("/host/$hostname/esx/vmfslabel",$DATA_ROOT);
  $new_vc_uuid = getScalar("/host/$hostname/esx/vc_uuid",$DATA_ROOT);
}

if ($DIFF) {
  $diff_users = getScalar("/host/$hostname/users",$DIFF);
  $diff_network = getScalar("/host/$hostname/network",$DIFF);
if ( $old_memory != $new_memory or $diff_network ) {

    # Upload the main configuration file of virtual machine to VMware
    esx_output("ESX: Upload the $hostname . vmx ... \n"");
    system("vifs --server $VC_SERVER --username $ESX_HOST_USERNAME --
           password $ESX_HOST_PASSWORD --put $PROJECT_PATH/$PROJECT/
           $hostname . vmx $[DATASTORE] $hostname . $PROJECT/$hostname . vmx";
    print VMWARECOMMANDS "vifs --server $VC_SERVER --username
           $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --put
           $PROJECT_PATH/$PROJECT/$hostname . vmx $[DATASTORE] $hostname.
           $PROJECT/$hostname . vmx\n"");

} }
# Upload the disk file of virtual machine to VMware ESX server

```bash
out("ESX: Upload the $hostname-flat.vmdk...

system("vifs --server SVC_SERVER --username $ESX_HOST/USERNAME --password $ESX_HOST/PASSWORD --put $PROJECT/path/$PROJECT/images/$hostname-flat.vmdk [SDATASTORE] $hostname.$PROJECT/hostname.$PROJECT/flat.vmdk")

print VMWARECOMMANDS "vifs --server SVC_SERVER --username $ESX_HOST/USERNAME --password $ESX_HOST/PASSWORD --put $PROJECT/path/$PROJECT/images/$hostname-flat.vmdk [SDATASTORE] $hostname.$PROJECT/hostname.$PROJECT/flat.vmdk"

close(VMWARECOMMANDS);
```

```bash
if ( ( $old_datastore and ($old_datastore ne $new_datastore) ) or ( $old_vc_server and ($old_vc_server ne $new_vc_server) )) {
    my $key;
    foreach $key ( getBlockKeys("/switch") ) {
        esx_buildSwitch($key);
    }
    return 1;
}
```

```bash
sub esx_unmountFilesystem {
    my $hostname = $[0];
    if ( getScalar("/host/$hostname/esx") ) {
        out("ESX: ESX plugin is unmounting $hostname on $MOUNTDIR\n
        system("vmware-mount -d $MOUNTDIR")

        print VMWARECOMMANDS "vmware-mount -d $MOUNTDIR"

        system("df -h")
        return 1;
    }
}
```

```bash
sub esx_removeHost {
    my $hostname = $[0];
    my $project = $[1];
    # Check the host is up or down, if it is up, stop and then delete it
    my $bo = getScalar("/host/$hostname/boot_order")
    $bo = 99 unless $bo;
    my $DATASTORE = getScalar("/host/$hostname/esx/datostore")
    $DATASTORE = $DEFAULT_DATASTORE unless $DATASTORE;
    verbose("ESX: DATASTORE: $DATASTORE\n
    my $VC_SERVER = getScalar("/host/$hostname/esx/vc_server")
    $VC_SERVER = $DEFAULT_VC_SERVER unless $VC_SERVER;
    verbose("ESX: VC_SERVER: $VC_SERVER\n
    my $ESX_HOST/USERNAME = getScalar("/host/$hostname/esx/username")
    $ESX_HOST/USERNAME = $DEFAULT_ESX_HOST/USERNAME unless $ESX_HOST/USERNAME;
    verbose("ESX: ESX_HOST/USERNAME: $ESX_HOST/USERNAME\n
    my $ESX_HOST/PASSWORD = getScalar("/host/$hostname/esx/password")
    $ESX_HOST/PASSWORD = $DEFAULT_ESX_HOST/PASSWORD unless $ESX_HOST/PASSWORD;
    verbose("ESX: ESX_HOST/PASSWORD: $ESX_HOST/PASSWORD\n
    # Check virtual machine status
    my $value = esx_checkIfUp($hostname,$project);
    while ( $value == 1 ) {
        out("ESX: The host $hostname.$Project is up, stop it first...

        system("vmware-cmd --server SVC_SERVER --username $ESX_HOST/USERNAME --password $ESX_HOST/PASSWORD /vmfs/volumes/SVMFLAG/hostname.$Project/hostname.$Project/vmware_vmx stop soft &\n
        print VMWARECOMMANDS "vmware-cmd --server SVC_SERVER --username $ESX_HOST/USERNAME --password $ESX_HOST/PASSWORD /vmfs/volumes/SVMFLAG/hostname.$Project/hostname.$Project/vmware_vmx stop soft\n
        system("sleep 10s")
        # MN delete stop bash scripting in upgrade, so do not use stop bash scripting here
```

137
if ($value == -1) {
  out("ESX: The host $hostname.$project is down\n");
}

if ($value == -1) {
  out("ESX: Remove host $hostname.$project.
  ");
  system("vifs --force --server SVC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --rm \"[DATASTORE] $hostname.$project\"\n");
  print VMWARECOMMANDS "vifs --force --server SVC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --rm \"[DATASTORE] $hostname.$project\\n";
  system("rm -rf $PROJECT_PATH/ $project/$hostname.$project.VMLIVE\") ;
  system("rm -rf $PROJECT_PATH/ $project/$hostname.$project.VMREGISTERSTATUS\") ;
  system("rm -rf $PROJECT_PATH/ $project/$hostname.$vmx\") ;
  system("rm -rf $PROJECT_PATH/ $project/images/$hostname.vmdk\") ;
  system("rm -rf $PROJECT_PATH/ $project/images/$hostname-flat.vmdk\") ;
  return 1;
} else {
  out("ESX: Failed to remove host $hostname.$project, abort...\n");
  return -1;
}
}

sub esx_checkIfUp {
  my $hostname = $_.[0];
  my $project = $_.[1];
  my $root = $_.[2];
  my $command;
  # The register Flag , 0 means a VM down and 1 means a VM up.
  my $registerFlag = 0;
  my $line;
  my $virtualMachine;
  my $vmxFile;
  my $VC_SERVER = getScalar("/host/$hostname/esx/vc_server");
  $VC_SERVER = $DEFAULT_VC_SERVER unless $VC_SERVER;
  my $ESX_HOST_USERNAME = getScalar("/host/$hostname/esx/username");
  $ESX_HOST_USERNAME = $DEFAULT_ESX_HOST_USERNAME unless $ESX_HOST_USERNAME;
  my $ESX_HOST_PASSWORD = getScalar("/host/$hostname/esx/password");
  $ESX_HOST_PASSWORD = $DEFAULT_ESX_HOST_PASSWORD unless $ESX_HOST_PASSWORD;
  my $VMFSLABEL = getScalar("/host/$hostname/esx/vmfslabel");
  $VMFSLABEL = $DEFAULT_VMFSLABEL unless $VMFSLABEL;
  verbose("ESX: VMFSLABEL: \$VMFSLABEL\n");
  # Get register list
  my $PROJECTSTATUS = "PROJECTSTATUS";
  $command = "vmware-cmd --server SVC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --l > $PROJECT_PATH/$project/$PROJECTSTATUS 2>&1";
  system("$command");
  print VMWARECOMMANDS "$command\n";
  verbose("ESX: $command: Command\n");
  verbose("ESX: hostname, PROJECT: $hostname.$project $hostname\n");
  open(REGISTER, "$PROJECT_PATH/$project/$PROJECTSTATUS") or die "Error: failed to open register data...$!\n";
  # The first line is a newline. Ignore it.
  $line = <REGISTER>;
  while ( $line = <REGISTER> ){
    chomp $line;
    if ( $line =~ "\^(vmfs/\ volumes/\((.+)-\(\w+\)-(\w+)-(\w+)\/(.*))/(.*)\)/(.*).(vmx)\n") {
      ...
    }
  }
}
$virtualMachine=$5;
$vmxFile=$6;
verbose("ESX: virtualMachine,vmxFile: $virtualMachine $vmxFile\n") ;
if ( ($virtualMachine eq "Hostname.$project") and ($vmxFile eq "$hostname") ){
  verbose("ESX: virtualMachine,vmxFile: $virtualMachine $vmxFile \n") ;
  verbose("ESX: hostname, PROJECT: Hostname.$project $hostname\n") ;
  $registerFlag=1;
  if ( ( $virtualMachine eq "$hostname.$project") and ( $vmxFile eq "$hostname") ) {
    verbose("ESX: virtualMachine, vmxFile: $virtualMachine $vmxFile \n") ;
    verbose("ESX: hostname, PROJECT: $hostname . $project $hostname\n") ;
    $registerFlag=1;
    verbose("ESX: registerFlag: Hostname.$project: $registerFlag\n") ;
    verbose("ESX: $hostname.$project has been registered.\n") ;
    # Check the VM is start or not
    $command = "vmware−cmd−server $VC_SERVER−username $ESX_HOST_USERNAME−password $ESX_HOST_PASSWORD /vmfs/volumes/$VMFSLABEL/Hostname.$project/hostname.vmx getstate" ;
    print VMWARE_COMMANDS "$command\n" ;
    open(RESULT, "$command |") or die "Error in esx host check up. $!
" ;
    while ( $line = <RESULT> ) {
      chomp $line ;
      if ( $line =~ /\(w+\)\(l\)\s=$on\)/ {
        # A host is up
        verbose("ESX: host is up\n") ;
        return 1 ;
      } elsif ( $line =~ /\(w+\)\(l\)\s=$off\)/ {
        # A host is down
        verbose("ESX: host is down\n") ;
        return −1 ;
      } elsif ( $line =~ /No\s\svirtual\smachine\sfound\.$/) {
        # No virtual machine found.
        verbose("ESX: host is non−existed\n") ;
        return 256 ;
      }
    }
    close(RESULT) ;
  } else {
    $registerFlag=0;
    verbose("ESX: registerFlag2: $virtualMachine: $registerFlag\n") ;
    verbose("ESX: $virtualMachine has been registered, but not the VM wanted.\n") ;
  } } else {
    $registerFlag=0;
    verbose("ESX: registerFlag3: Hostname.$project: $registerFlag\n") ;
    verbose("ESX: Hostname.$project has not been registered. Run \"mln start −p project\" to start it.\n") ;
  }
} close(REGISTER) ;
# Tell MN the status
if ( $registerFlag == 0 ) {
  verbose("ESX: registerFlag4: Hostname.$project: $registerFlag\n") ;
  verbose("ESX: Finally, after a search, Hostname.$project has not been registered. Run \"mln start −p project\" to start it.\n") ;
  return −1 ;
} }
sub esx_checkIfSwitchIsUp {
  my $switch = $_.[0] ;
}
my $project = $_.[1];
my $root = $_.[2];
my $VC_SERVER = getScalar("/host/Hostname/esx/vc_server");
$VC_SERVER = $DEFAULT VC_SERVER unless $VC_SERVER;
verbose("ESX: VC_SERVER: $VC_SERVER
");
my $ESX_SERVER = getScalar("/host/Hostname/esx/server");
$ESX_SERVER = $DEFAULT ESX_SERVER unless $ESX_SERVER;
verbose("ESX: ESX_SERVER: $ESX_SERVER
");
my $ESX_HOST_USERNAME = getScalar("/host/Hostname/esx/username");
$ESX_HOST_USERNAME = $DEFAULT ESX_HOST_USERNAME unless $ESX_HOST_USERNAME;
verbose("ESX: ESX_HOST_USERNAME: $ESX_HOST_USERNAME
");
my $ESX_HOST_PASSWORD = getScalar("/host/Hostname/esx/password");
$ESX_HOST_PASSWORD = $DEFAULT ESX_HOST_PASSWORD unless $ESX_HOST_PASSWORD;
verbose("ESX: ESX_HOST_PASSWORD: $ESX_HOST_PASSWORD
");
my $command = "vicfg-vswitch --server $VC_SERVER --username 
$ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --c $project
";
print VMWARE_COMMANDS $command

while( my $status = <STATUS> ){
  chomp $status;
  verbose("ESX: virtual switch $name.$project status: $status\n");
  if($status =~ /1$/){
    # Switch is up
    return 1;
  } elsif( $status =~ /0$/){
    # Switch is down
    return -1;
  }
}
close(STATUS);

sub esx_removeSwitch {
  my $name = $_.[1];
  my $project = $_.[2];
  my $root = $_.[2];
  out("ESX: ESX plugin to remove virtual switch activated...
");
  # Check virtual switch status
  my $value = esx_checkIfSwitchIsUp($name,$project);
  while ( $value == 1 ) {
    out("ESX: virtual switch $name.$project is on, stop it first...
");
    system("$PROJECT_PATH/$project/stop$Name.sh");
    $value = esx_checkIfSwitchIsUp($name,$project);
  }
  if ( $value == -1 ){
    out("ESX: virtual switch $name.$project has been removed successfully
");
    return 1;
  } else{
    return -1;
  }
}

sub esx_buildSwitch {
  my $name = $_.[1];
  my $SwitchList = "
  # vSwitch flag: 0 means a vSwitch is not built, 1 means a vSwitch has been built
  # Used in the first time, since no OLD_DATA_ROOT in the first time
  my $built="$PROJECT_PATH/$PROJECT/$name.$PROJECT_VSWITCH_BUILT";
  if( ! stat $built ){
    system("echo '0' > $built");
  }
  # Get older virtual switch number
  my $oldSwitch = getBlockKeys("/switch",SOLD_DATA_ROOT);
  out("ESX: oldSwitch = $oldSwitch\n");
# esx block does not exist here
my $VC_SERVER = getScalar("/host/$hostname/esx/vc_server");
verbose("ESX: VC_SERVER: $VC_SERVER
");
my $VC_SERVER = $DEFAULT VC_SERVER unless $VC_SERVER;
my $ESX_HOST_USERNAME = getScalar("/host/$hostname/esx/username");
$ESX_HOST_USERNAME = $DEFAULT ESX_HOST_USERNAME unless $ESX_HOST_USERNAME;
my $ESX_HOST_PASSWORD = getScalar("/host/$hostname/esx/password");
$ESX_HOST_PASSWORD = $DEFAULT ESX_HOST_PASSWORD unless $ESX_HOST_PASSWORD;
my $old_vc_server;
my $new_vc_server;
my $old_username;
my $new_username;
my $old_password;
my $new_password;
$old_vc_server = getScalar("/host/$hostname/esx/vc_server", $OLD_DATA_ROOT);
$new_vc_server = getScalar("/host/$hostname/esx/vc_server", $DATA_ROOT);
if ($old_vc_server and ($old_vc_server ne $new_vc_server )){
$VC_SERVER = $new_vc_server;
$ESX_HOST_USERNAME = $new_username;
$ESX_HOST_PASSWORD = $new_password;
}
open(START, ">$PROJECT_PATH/$PROJECT/start$name.$PROJECT"") or die "Failed to open $PROJECT_PATH/$PROJECT/start$name.$PROJECT"
print START " #!/ bin/bash"
print START " echo 'Starting switch $name.$PROJECT'"
if ( $old_vc_server and ( $old_vc_server ne $new_vc_server )){
$VC_SERVER = $new_vc_server;
$ESX_HOST_USERNAME = $new_username;
$ESX_HOST_PASSWORD = $new_password;
}
print START "vicfg --vswitch --server $VC_SERVER --username $ESX_HOST_USERNAME --password $ESX_HOST_PASSWORD --c $name.$PROJECT">
141
```perl
print START "\"if [ -eq 0 ]; then \"n\";"
print START "\"Create virtual switch \$name:\$PROJECT...\"\"n\";"
# Create a virtual switch
print START "\"vicfg-vswitch --server SVC_SERVER --username
\$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD --add \$name:\$PROJECT\"\n\" ;"
# Create the subnet
print START "\"vicfg-vswitch --server SVC_SERVER --username
\$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD --add-pg \$name.
\$PROJECT\"\n\" ;"
# Set MIC to 1500 bytes
print START "\"vicfg-vswitch --server SVC_SERVER --username
\$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD --m 1500 \$name.
\$PROJECT\"\n\" ;"
print START "\"fi\"n\";"
print START "\"if [ -eq 1 ]; then \"n\";"
print START "\"Virtual switch \$name:\$PROJECT is already existed, abort...\"\"n\";"
print START "\"exit \"n\";"
print START "\"fi\"n\";"
close (START);

system ('\"chmod +x $PROJECT_PATH/$PROJECT/start_$name.sh\"');

open( STOP, \"$PROJECT_PATH/$PROJECT/stop_$name.sh\" ) or die "Failed to open $PROJECT_PATH/$PROJECT/stop_$name.sh\"n\";
print STOP \"#!/bin/bash\"n\";
print STOP \"echo \"Stopping switch \$name:\$PROJECT\"n\";"
# Check a virtual switch existence
print STOP "\"vicfg-vswitch --server SVC_SERVER --username
\$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD --c \$name:\$PROJECT >
\$PROJECT_PATH/$PROJECT/\$name:\$PROJECT:\$PROJECT\"n\";"
print STOP "\"if [ -eq 1 ]; then \"n\";"
print STOP "\"Cannot stop non-existed virtual switch: \$name:\$PROJECT abort...\"\"n\";"
print STOP "\"exit \"n\";"
print STOP "\"fi\"n\";"
print STOP "\"if [ -eq 1 ]; then \"n\";"
print STOP "\"Stopping the virtual switch \$name:\$PROJECT...\"\"n\";"
# Remove its subnet
print STOP "\"vicfg-vswitch --server SVC_SERVER --username
\$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD --del-pg \$name.
\$PROJECT \$name:\$PROJECT\"n\";"
# Remove the virtual switch
print STOP "\"vicfg-vswitch --server SVC_SERVER --username
\$ESX_HOST_USERNAME --password \$ESX_HOST_PASSWORD --delete \$name.
\$PROJECT\"n\";"
print STOP "\"fi\"n\";"
close (STOP);

system ('\"chmod +x $PROJECT_PATH/$PROJECT/stop_$name.sh\"');

if (\$SwitchList[\$name]=1;}
system (\"echo '1' > \$built\";
)

# Remove vSwitches removed from the project file
open(BUILT, \"$built\") or die "Error: can't open \$built.\"n\";
while($line = <BUILT>){
chomp $line;
# It built last time
if($line == 1){

# Get all the virtual switch name from the list
```
foreach $vs (keys %vSwitchList){
    # It does not exist this time
    if(%vSwitchList{$vs} == 0){
        out("ESX: Remove $vs $PROJECT ...
        
        system("rm -rf $PROJECT_PATH/$PROJECT/start_$vs.sh");
        system("rm -rf $PROJECT_PATH/$PROJECT/stop_$vs.sh");
        system("rm -rf $PROJECT_PATH/$PROJECT/$vs.$PROJECT.
        VSWITCHBUILT");
        system("rm -rf $PROJECT_PATH/$PROJECT/$vs.$PROJECT.
        VSWITCHSTATUS");
    }
}
return 1;
}

sub esx_configureSwitch {
    out("ESX: VMware vSphere ESX plugin esx_configureSwitch called
    ","red");
}
1;
Appendix D

Full Output of Migrating country.mln

```
building global config (country.mln)
fetching old config: /opt/mln/projects/country/country.mln

Upgrade Info:

The Following Diff has been calculated

host {
    city {
        esx {
            vmfslabel 4db83218-04c4d991-28f4-000c2908bab8
            password corenetwork2
            vc_uuid 52 e2 33 a9 0b 6a e9 e1-01 3f 78 70 d1 c0 f3
class "db"
            vc_server 128.39.73.235
        }
    }
}

No hosts will be removed

UPGRADING country
Collecting Status information for country

FATAL: One or more hosts in this project could not be verified.
This might be that the MLN daemon is not running on the service host,
or that the project does not exist on it.
Use the -r option to override if you really want to continue.

--- Building switch center
vmware_buildSwitch called:
  ESX: oldvSwitch = 1
  ESX: Old vc_server =
  ESX: New vc_server =
  ESX: Old username =
  ESX: New username =
  ESX: Old password =
  ESX: New password =
  ESX: ESX plugin activated to build switch: center.country
  ESX: virtual switch: center
  --- city
  ESX: ESX plugin is enabled.
  ESX: Old datastore = datastore1
  ESX: New datastore = datastore1
  ESX: Old vc_server = 128.39.73.231
  ESX: New vc_server = 128.39.73.235
  ESX: Old username = root
```
ESX: New username = root
ESX: Old password = corenetwork
ESX: New password = corenetwork2
ESX: Old vmfslabel = 4c74cc2d-f3d16a79-36d5-b8ac6f21f590
ESX: New vmfslabel = 4db83218-04c4d991-28f4-000c2908bab8
ESX: Old vc_uuid = 52 46 f5 2e 65 33 35 04-e1 0e bf f5 aa 6c 8f 13
ESX: New vc_uuid = 52 e2 33 a9 0b 6a e9 e1-01 3f 78 70 d1 c0 f3 db
ESX: Create the virtual machine directory...
Created directory '[datastore1] city.country' successfully.
ESX: Removing old remote vmx file...
Deleted file '[datastore1] city.country/city.vmx' successfully.
ESX: Removing old local disk...
ESX: Downloading old remote disk...
Downloaded file to /opt/mln/projects/country/images/city-flat.vmdk successfully.
ESX: Removing old remote disk...
Deleted file '[datastore1] city.country-flat.vmdk' successfully.
Deleted file '[datastore1] city.country/vmdk' successfully.
ESX: Remove host city.country
ESX: esx_createFilesystem did filesystem creation
ESX: ESX plugin is mounting city on /root/.mln_mountdir
[2011-04-28 15:04:16.293 7FF72D741700 verbose 'ThreadPool'] Thread info: Min Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max fds: 2, 21, 2, 10, 31, 4, 600, -1
[2011-04-28 15:04:16.293 7FF72D741700 trivias ' ThreadPool'] Thread pool launched
Filesystem Size Used Avail Use% Mounted on
/dev/sda1 222G 60G 152G 29% /
tmpfs 1.9G 0 1.9G 0% /lib/init/rw
udev 1.9G 152K 1.9G 1% /dev
tmpfs 1.9G 0 1.9G 0% /dev/shm
[dev/loop0 1.9G 1016M 769M 57% /root/.mln_mountdir
Adding users: karl,root,Adding interface eth0 (128.39.73.236)
ESX: ESX plugin is unmounting city on /root/.mln_mountdir
[2011-04-28 15:04:17.277 7FCEDA418700 verbose 'ThreadPool'] Thread info: Min Io, Max Io, Min Task, Max Task, Max Thread, Keepalive, thread kill, max fds: 2, 21, 2, 10, 31, 4, 600, -1
[2011-04-28 15:04:17.277 7FCEDA418700 trivias ' ThreadPool'] Thread pool launched
Filesystem Size Used Avail Use% Mounted on
/dev/sda1 222G 60G 152G 29% /
tmpfs 1.9G 0 1.9G 0% /lib/init/rw
udev 1.9G 152K 1.9G 1% /dev
tmpfs 1.9G 0 1.9G 0% /dev/shm
We are in upgrade mode
ESX: Creating: /opt/mln/projects/country/start_99_city.sh
ESX: Creating: /opt/mln/projects/country/stop_99_city.sh
ESX: Writing interface ethernet0
ESX: Interface ethernet0 is going to connect to virtual switch center.country
ESX: Old memory = 512M
ESX: New memory = 512M
ESX: Old disk = 2GB
ESX: New disk = 2GB
ESX: Old users = 1
ESX: New users = 1
ESX: Diff users =
ESX: Old nameserver = 128.39.89.8
ESX: New nameserver = 128.39.89.8
ESX: Old network = 1
ESX: New network = 1
ESX: Diff network =
ESX: Old datastore = datastore1
ESX: New datastore = datastore1
ESX: Old vs_server = 128.39.73.231
ESX: New vs_server = 128.39.73.235
ESX: Old username = root
ESX: New username = root
ESX: Old password = corenetwork
ESX: New password = corenetwork2
ESX: Old vmfslabel = 4c74cc2d-f3d16a79-36d5-b8ac6f21f590
ESX: New vmfslabel = 4db83218-04c4d991-28f4-000c2908bab8
ESX: Old vs_uuid = 52 46 f5 2e 65 33 35 04--el 0e bf f5 aa 6c 8f 13
ESX: New vs_uuid = 52 e2 33 a9 0b 6a e9 e1--01 3f 78 70 d1 c0 f3 db
ESX: Upload the city.vmx...
ESX: Upload the city.vmdk...
ESX: Upload the city-flat.vmdk...
ESX: oldvSwitch = 1
ESX: Old vs_server = 128.39.73.231
ESX: New vs_server = 128.39.73.235
ESX: Old username = root
ESX: New username = root
ESX: Old password = corenetwork
ESX: New password = corenetwork2
ESX: ESX plugin activated to build switch: center.country
ESX: virtual switch: center
Done
Saving Config file: /opt/mln/projects/country/country.mln
Appendix E

VMware Commands to Build Network simpleTopoology

```bash
1. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --mkdir '/datastore1/Par is .simpleTopo logy'
2. vmware-vdiskmanager -r /opt/mln/templates/debian6.2GB.vmdk -t 4 /opt/mln/projects/simpleTopo logy/images/Paris.vmdk
3. vmware-vdiskmanager -X 3072MB /opt/mln/projects/simpleTopo logy/images/Paris.vmdk
4. vmware-mount /opt/mln/projects/simpleTopo logy/images/Paris.vmdk /root/.mln_mountdir
5. vmware-mount -d /root/.mln_mountdir
6. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --put /opt/mln/projects/simpleTopo logy/images/Paris.vmdk '/datastore1/Par is .simpleTopo logy/Paris.vmx'
7. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --put /opt/mln/projects/simpleTopo logy/images/Paris.vmdk '/datastore1/Par is .simpleTopo logy/Paris.vmdk'
8. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --put /opt/mln/projects/simpleTopo logy/images/Paris-flat.vmdk '/datastore1/Par is .simpleTopo logy/Paris-flat.vmdk'
9. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --mkdir '/datastore1/London.simpleTopo logy'
10. vmware-vdiskmanager -r /opt/mln/templates/debian6.2GB.vmdk -t 4 /opt/mln/projects/simpleTopo logy/images/London.vmdk
11. vmware-vdiskmanager -X 3072MB /opt/mln/projects/simpleTopo logy/images/London.vmdk
12. vmware-mount /opt/mln/projects/simpleTopo logy/images/London.vmdk /root/.mln_mountdir
13. vmware-mount -d /root/.mln_mountdir
14. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --put /opt/mln/projects/simpleTopo logy/images/London.vmdk '/datastore1/London.simpleTopo logy/London.vmx'
15. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --put /opt/mln/projects/simpleTopo logy/images/London.vmdk '/datastore1/London.simpleTopo logy/London.vmdk'
16. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --put /opt/mln/projects/simpleTopo logy/images/London-flat.vmdk '/datastore1/London.simpleTopo logy/London-flat.vmdk'
17. vi f s  --server 128.39.73.231  --username root  --password corenetwork  --mkdir '/datastore1/ Oslo.simpleTopo logy'
18. vmware-vdiskmanager -r /opt/mln/templates/debian6.2GB.vmdk -t 4 /opt/mln/projects/simpleTopo logy/images/Oslo.vmdk
19. vmware-vdiskmanager -X 5120MB /opt/mln/projects/simpleTopo logy/images/Oslo.vmdk
20. vmware-mount /opt/mln/projects/simpleTopo logy/images/Oslo.vmdk /root/.mln_mountdir
```
vmware–mount -d /root/.mln_mountdir
Appendix F

VMware Commands to Build Network complexTopology

```bash
1 vi fs − server 128.39.73.231 — username root — password corenetwork — mkdir '['
datastore1] laptop.firewallTopology'
2 vmware−vdiskmanager − r ./opt/mln/templates/debian6.2GB.vmdk − t 4 /opt/mln/
 projects/firewallTopology/images/laptop.vmdk
3 vmware−vdiskmanager − x 3072MB /opt/mln/projects/firewallTopology/images/laptop
 .vmdk
 mln_mountdir
5 vmware−mount − d /root/.mln_mountdir
6 vi fs − server 128.39.73.231 — username root — password corenetwork — put /opt/
 firewallTopology/laptop.vmx'
7 vi fs − server 128.39.73.231 — username root — password corenetwork — put /opt/
 firewallTopology/laptop.vmdk'
8 vi fs − server 128.39.73.231 — username root — password corenetwork — put /opt/
 .firewallTopology/laptop−flat.vmdk'
9 vi fs − server 128.39.73.231 — username root — password corenetwork — mkdir '['
datastore1 ] server2.firewallTopology'
10 vmware−vdiskmanager − r ./opt/mln/templates/debian6.2GB.vmdk − t 4 /opt/mln/
 projects/firewallTopology/images/server2.vmdk
11 vmware−vdiskmanager − x 3072MB /opt/mln/projects/firewallTopology/images/
server2.vmdk
 mln_mountdir
13 vmware−mount − d /root/.mln_mountdir
14 vi fs − server 128.39.73.231 — username root — password corenetwork — put /opt/
 firewallTopology/server2.vmx'
15 vi fs − server 128.39.73.231 — username root — password corenetwork — put /opt/
 firewallTopology/server2.vmdk'
16 vi fs − server 128.39.73.231 — username root — password corenetwork — put /opt/
 mln/projects/firewallTopology/images/server2−flat.vmdk '[' datastore1 ]
server2.firewallTopology/server2−flat.vmdk'
17 vi fs − server 128.39.73.231 — username root — password corenetwork — mkdir '['
datastore1 ] server1.firewallTopology'
18 vmware−vdiskmanager − r ./opt/mln/templates/debian6.2GB.vmdk − t 4 /opt/mln/
 projects/firewallTopology/images/server1.vmdk
19 vmware−vdiskmanager − x 3072MB /opt/mln/projects/firewallTopology/images/
server1.vmdk
 mln_mountdir
```
vmware-mount /mntdir /root/.mln_montdir
vifs -s server 128.39.73.231 —username root —password corenetwork —put /opt/ mln/projects/firewallTopology/images/server1.vmdk 
firewallTopology/images/server1.vmdk'


vifs -s server 128.39.73.231 —username root —password corenetwork —mkdir ' [ datastore1] gateway. firewallTopology'

vmware-vdiskmanager -r /opt/mln/templates/debian6.2GB.vmdk -t 4 /opt/mln/ projects/firewallTopology/images/gateway.vmdk

vmware-vdiskmanager -x 3072MB /opt/mln/projects/firewallTopology/images/gateway.vmdk

vmware-mount /opt/mln/projects/firewallTopology/images/gateway.vmdk /root/.mln_montdir

vmware-mount /mntdir /root/.mln_montdir
vifs -s server 128.39.73.231 —username root —password corenetwork —put /opt/ mln/projects/firewallTopology/gateway.vmx ' [datastore1] gateway. firewallTopology/gateway.vmx'


vifs -s server 128.39.73.231 —username root —password corenetwork —mkdir ' [ datastore1] desktop. firewallTopology'

vmware-vdiskmanager -r /opt/mln/templates/debian6.2GB.vmdk -t 4 /opt/mln/ projects/firewallTopology/images/desktop.vmdk

vmware-vdiskmanager -x 3072MB /opt/mln/projects/firewallTopology/images/desktop.vmdk

vmware-mount /opt/mln/projects/firewallTopology/images/desktop.vmdk /root/.mln_montdir

vmware-mount /mntdir /root/.mln_montdir


vifs -s server 128.39.73.231 —username root —password corenetwork —mkdir ' [ datastore1] choke. firewallTopology'

vmware-vdiskmanager -r /opt/mln/templates/debian6.2GB.vmdk -t 4 /opt/mln/ projects/firewallTopology/images/choke.vmdk

vmware-vdiskmanager -x 3072MB /opt/mln/projects/firewallTopology/images/choke.vmdk

vmware-mount /opt/mln/projects/firewallTopology/images/choke.vmdk /root/.mln_montdir

vmware-mount /mntdir /root/.mln_montdir


Appendix G

User Interface: face.php

```html
<html><head>
title>VMware ESX Server</title>
<style type="text/css">
body {background-color: #E6E6E6;}
p {style="margin-top: 1; margin-bottom: 1"; align="center";}
</style>
</head>
<body>
<form name="main" method="post" action="process.php">
<table align="center" border="0" cellpadding="1" cellspacing="0">
<tr><td align="center" colspan="2">
<p><b><font face="Arial" align="center" size="4" color="#000000">VMware ESX Server</font></b></p>
</td></tr>
<tr><td align="center" colspan="2">
<input type="submit" name="submit1" value="Submit" />
</td></tr>
</table>
</form>
</body>
</html>
```
Appendix H

User Interface: process.php

```php
<?php
$topologySelect = $_POST['topologySelect'];
$username = $_POST['username'];
$password = $_POST['password'];
$ip = $_POST['ip'];
$localUser = $_POST['localUser'];
$localPassword = $_POST['localPassword'];
print($topologySelect . "\n");
print($username . "\n");
print($password . "\n");
print($ip . "\n");
print($localUser . "\n");
print($localPassword . "\n");

// Copy project file

$number = md5(uniqid(rand(), true));
# Since ESX only support 31 chars for name, so number is one digit
$number = substr($number, 0, 1);
if($topologySelect){
    if(strcmp($topologySelect, 'simpleTopology') == 0 ){
        $project = 'simpleTopology';
        $source = 'simpleTopology.mln';
        $destination = "$number.simpleTopology.mln";
        print("$number.simpleTopology.mln" . "\n");
    } elseif(strcmp($topologySelect, 'complexTopology') == 0 ){
        $project = 'firewallTopology';
        $source = 'firewallTopology.mln';
        $destination = "$number.firewallTopology.mln";
        print("$number.firewallTopology.mln" . "\n");
    } else{
        $project = 'esxtemplate';
        $source = 'esxtemplate.mln';
        $destination = "$number.esxtemplate.mln";
        print("$number.esxtemplate.mln" . "\n");
    }
}$data = @file_get_contents($source);
$handle = fopen($destination, 'w') or die("can't open file");
fwrite($handle, $data);close($handle);
setcookie("NUMBER", $number, time() + 3600);
```
setcookie("PROJECT", $project, time()+3600);

// Edit project file
$modification1="sed -i \"s/esxroot/$username/\" ${number}.${source};
$modification2="sed -i \"s/esxpassword/$password/\" ${number}.${source};
$modification3="sed -i \"s/esxip/$ip/\" ${number}.${source};
$modification4="sed -i \"s/localuser/$localuser/\" ${number}.${source};
$modification5="sed -i \"s/localpassword/$localpassword/\" ${number}.${source};
$modification6="sed -i \"s/esxtemplate/$esxtemplate/${number}.${project}/\" ${number}.${source};

system("$modification1");

system("$modification2");

system("$modification3");

system("$modification4");

system("$modification5");

system("$modification6");

// Since permission problem, a deamon is advised, MNL needs root permission
$construction="mln build -f ${number}.${source} > /tmp/${number}.${project} &";
$host="127.0.0.1";
$port=12345;
$timeout=30;
$sk = fsockopen($host, $port, $errnum, $errstr, $timeout);
if ( ! is_resource($sk)){
    print("Connection fail: ", $errnum, ", ", $errstr, 
);
    exit("Connection fail: ", $errnum, ", ", $errstr);
} else{
    fputs($sk, "$construction");
    print("Connection is OK\n");
}
fclose($sk);
require('display.php');
?>
</body>
</html>
Appendix I

User Interface: display.php

```html
<html>
<head>
<title>Display</title>
<META HTTP-EQUIV=Refresh CONTENT="15; URL=http://client1.vlab.iu.hio.no/display.php"/>
</head>
<body>
<?php
$number = $_COOKIE['NUMBER'];
$project = $_COOKIE['PROJECT'];
print(system("cat /tmp/${number}.${project}"));
?>
</body>
</html>
```
Appendix J

daemon.pl

```perl
#!/usr/bin/perl
use IO::Socket;
use POSIX;
my $SOLDSIG = $SIG{"CHILD"};
$SIG{"CHILD"} = "IGNORE";
POSIX::setsid() or die "Can't start a new session: $!";

my $sock = new IO::Socket::INET(
    LocalHost => "127.0.0.1",
    LocalPort => "12345",
    Proto => 'tcp',
    Listen => SOMAXCONN,
    Reuse => 1
);

$sock or ( warn "no socket: $!
" and return);

my $time = localtime time;
print "$time
";
my(%$new_sock, $buf);
while( $new_sock = $sock->accept() ){
    my $pid = fork();
    if( $pid != 0 ){
        # Still on the parent daemon
        close($new_sock);
        print "Parent closing socket
";
        next;
    }else{
        my $client = $new_sock->peerhost();
        print "Connection from $client\n";
        my $command = <$new_sock>;
        if( $command =~ /mln build -f .*$/){
            print $new_sock "OK\n";
            system("$command");
        }else{
            print "Command wrong\n";
            print $new_sock "Not OK\n";
        }
        close($new_sock);
        print "child exiting\n";
        exit 0;
    }
}
```