The MLN Manual

Version 1.0

by Kyrre Begnum and John Sechrest

Project homepage: mln.sourceforge.net

Latest manual version: mln.sourceforge.net/doc

Email help: mln-help@lists.sourceforge.net

The MLN Manual
mln version 1.0.1

November 4, 2009

Contents

Introduction

MLN is a powerful tool that can configure and administer virtual networks for you. Key features of MLN include:

- Support for Xen, VMware Server or User-Mode Linux virtual machines,
- Root permissions not required, and
- Easy installation with pre-existing virtual machine templates.

For a quick start, take a look at Section . However, this document will also provide in-depth explanations of the motivation for MLN as well as its features and advantages.

Quick Guide for the Impatient

1. MLN depends on the following software:
   - Perl
   - uml-utilities
   - bridge-utils
   - screen
   - sudo
2. Download MLN:
   - wget http://mln.sourceforge.net/files/mln-latest.tar.gz
   - tar zxf mln-latest.tar.gz
3. Run the interactive setup:
   - cd mln-latest
   - ./mln setup
   - During the setup process, accept all defaults by simply hitting Return at each prompt.
4. Build an example project:
   - ./mln build -f examples/simple-network.mln
5. Start your new virtual network:
   - ./mln start -p simple-network
6. **Now you should have 3 xterms, one for each virtual host in the simple-network** project. Login as root to each one (no password) and play!
7. Stop your new virtual network:
   - ./mln stop -p simple-network

Chapter 1

Overview

The goal of this chapter is to explain the key concepts related to the inner workings of MLN.

1.1 Main Concepts

1.1.1 Virtual Host

A *virtual host* consists of its own filesystem and runs in software on top of your OS. MLN uses either User-Mode Linux (UML) or Xen and can use different filesystems based on different Linux distributions. MLN will customize the filesystem for the virtual host you wish to build based on your high-level specification.

1.1.2 Filesystem Template

A *template* is a basic pre-customized filesystem for a virtual host. You can download templates from the project homepage and choose which hosts should be built from those templates. Templates differ in what distribution they are based on and how much software they contain. For example, one template might be a user desktop environment with graphical login and numerous productivity applications while another might be a small firewall environment using busybox to replace applications and conserve space.

1.1.3 Virtual Switch

MLN supports the virtual switch capability provided by UML. **uml_switch** is simply a process that opens up a Unix-socket and listens to it. It accepts network packets on that socket and behaves just like a typical home-network switch. Virtual hosts can connect to these switches. For Xen, the switch is a so-called ethernet bridge-device that can connect several network interfaces together like a switch.

1.1.4 Virtual Networks

Virtual hosts connected to virtual switches constitute a *virtual network*, which mln is mainly all about. Many virtual hosts and switches make pretty large networks and it is MLN's job to configure and build these networks for you. You can choose to build automatically functional networks or you can build lots of virtual machines that are connected to switches and configure networking on them by hand too if you like. It's up to you.

Virtual networks can also be part of your real networks. Meaning that neither your physical hosts nor the virtual hosts can tell the difference, they are simply on the same LAN.

1.1.5 Projects

One virtual network is one *project*. MLN can build and run several projects at the same time. Sometimes it is sensible to keep them apart, other times you might wish to connect them together. Projects are identified by their name.

1.1.6 The MLN Language

You might wonder how you should tell mln what the virtual network should look like? The answer is the *mln configuration language*. This language looks much like a declarative programming language. The goal is that easy networks should be easy to write while complex networks should be possible to write (and in some cases hopefully easy as well). Based on your needs, you can omit or add parts to your project to make it do exactly what you want. Sometimes the point is to build simple networks without much configuration of the virtual hosts. This is a typical setting for student assignments. So if you don't want much, you shouldn't have to write much. Here is an example of a small network consisting of two machines and a switch. If you understand this configuration without too much hassle, then the rest of mln should be straight forward for you.

```
global {
      project simple_network
}

switch lan {
}

host starfish {

    network eth0 {
          switch lan
          address 10.0.0.1
          netmask 255.255.255.0
    }
}
```

```
host catfish {

    network eth0 {
        switch lan
        address 10.0.0.2
        netmask 255.255.255.0
    }
}
```

Installation and Use

Chapter 2

Templates

Every virtual machine's filesystem is built from a template. There are currently five different pre-made templates available:

- **Debian-3.0 (aka woody)**
  This is the smallest Debian-based template. It basically contains the base-system. Nice for regular dummy machines and it is possible to install new software on them using apt. It contains a dhcp client for quick access to local networks.

  This template is necessary for MLN build process even though the resulting virtual machine is not based on this template.
  Minimum build size: 75MB

- **sarge-thick**
  built from Debian sarge (3.1), this template contains the sarge base-system and some additional apps: tcpdump, bonnie++, vtun, hping2.
  Minimum build size: 220MB

- **ubuntu-server**
  The template is buildt from a base install of ubuntu breezy. It does not contain much software other than a dhcp client. It is a good starting-point for minimal servers.

- **ubuntu-desktop**
  This is by far the largest template as it is 1.4GB large when extracted. On the other hand, it contains all the software installed by a regular Ubuntu Breezy install, including office tools. The special thing about this template is that it is modified to start the Ubuntu Login screen in a VNC session, enabling users to connect to the running virtual machine using a VNC client and to use the graphical Ubuntu desktop.

- **busybox**
  Busybox is a small linux distribution usually meant for floppies and the like. It makes a nice router in virtual networks, because it takes very little space. Complex things, like adding users and groups, are not supported in this image.
  Minimum build size: 25MB

- **blimp**
  This is a typical LAMP fileystem with apache, mysql and PhP. Pre-installed software is Drupal, Mediawiki and request-tracker.

The default host filesize is 250MB. You can set a smaller size, but MLN will refuse building hosts where the assigned size is smaller then the actual template.

2.1 Downloading Templates and Template Versions

MLN has it's own download manager. It is launched by typing:

```
mln download_templates
```

The first thing the download manager does, is to fetch the latest list of available templates. It then prompts you for every available template and asks if you want to download it. The default answer to that question is "No", so by pressing enter, you'll skip to the next template. The presented template will show the word "NEW!" if you have an older version or you don't have any version of it.

The templates are compressed and will be unpacked automatically when downloaded.

The good thing with versions, is that you don't have to specify what version of the template you want. You actually don't have to know anything about the versions. When you say: sarge-thick.ext2, then mln will use the newest version that you have automatically.

You can also specify exactly what template you want to use. If mln does not find a version for your template, it will assume it is one of your own templates and try to use it. So if you say template foobar.ext2, then mln will assume you have a template called exactly that in your templates directory.

2.1.1 Version numbering of Templates

The syntax of the version syntax is:
template-name\textbf{-V\emph{m}.\emph{n}}.ext2

Where *m* and *n* are the major and minor version numbers respectively.

2.1.2 Dealing with a slow download

The MLN download manager fetches its templates form one particular sourceforge mirror. This does not suit everyone, off course. If you feel brave, then you are invited to edit the mln script and change the URL to a mirror closer to you. You should find the variable in the beginning of the script. But, we cannot guarantee, that you will find all templates on all mirrors. We will update mln as soon as we figure out how we can let sourceforge choose the mirror itself. Any pointers are welcome. As of version 0.71 this is how it is done, however.

2.1.3 Downloading and Registering templates manually

If you have downloaded templates manually from a faster sourceforge.net mirror or modified or even made one yourself, you can add it to MLN's template registry with this command:

```
mln register_template [ -m "message" ] -t template
```

If the template-name contains a valid version tag, then MLN will take notice of it, and you can use the template name in your configurations without the version name in order to get the latest version of the template.

2.2 Managing Templates

MLN keeps track of its templates by storing them in your templates directory. It is possible to share the templates directory between several users, since one only takes copies of the template. Just make sure every one has read access to them. A list of all downloaded templates is stored in a file called templates.list, also stored in your templates folder. If you want to have a list of the templates mln knows of locally, you can write:

```
mln list_templates
```

There is currently no support for removing templates, so you will have to remove them by hand and delete the corresponding line from the templates.list file.

Chapter 3

Building Projects

As of version 0.73, MLN assumes that a non-root user build the projects. In order for the build process to work, you will need to at least have the default template, which is obtained during the setup process.

3.1 mln build

To build a project, specify the name of the project file you would have created or would like to use.

```
mln build -f project-file.mln
```

The build command is rather simple, but a few extra steps can prevent some frustrations later on. First, you need a project file that describes the project you want to build. For rather complex networks it's a good idea to run a simulation first. The simulation just reads the project file and outputs the corresponding data structure. That way you can double-check if something is misspelled or just simply wrong. To run a simulation, add the option -s} after the \texttt{build command:

```
mln build -s -f project-file.mln
```

Here you can see how mln understands the project file. If you like what you see, you can start the build process. The name of the project might correspond to an already existing project and that will be overwritten when you build this one. You will, however, be asked for permission to do so. If you are sure that you want to overwrite any existing project with the same name and don't want to be bothered about it, add the -r option after the build command.

```
mln build -r -f project-file.mln
```

3.2 Non-Root Building

Normally, you need to mount a filesystem in order to modify its contents. *root* is the only user allowed to do that. MLN has a way to circumvent this.

The trick is to do everything we can as regular users, like copying and resizing the templates. Before the filesystem images are mounted, MLN boots into a user-mode-linux system ourselves and, as root, mounts the images from there and configure them.

Note, that one effect that not being root, is that you cannot build on behalf of someone else. So the owner, sudo} and \texttt{group (see Syntax chapter) keywords won't work.

3.3 Upgrading Running Projects

MLN has, as of 0.71, the possibility to upgrade running virtual networks. This is done the following way: When you build a new project, mln stores a copy of the project file along with the project. You can then update your own copy, by changing variables, adding/removing hosts and switches (as long as you don't change the name of the project). Then, you can run the mln command for upgrading, and it will compare its own copy and your new copy to figure out which virtual hosts need to be rebuilt. This comparison is quite complex, i.e. if you change a variable in a superclass, all machines that inherit from it will have to be rebuilt, but not the ones that inherit, but override the variable themselves. So a change in the syntax, might not give a change in the semantics.
Why is it necessary to upgrade a running project? Why can't you just rebuild? Important question. The answer is, that you can get far by just rebuilding the whole project. But sometimes it is not what you want. You don't have to rebuild (and thereby delete the old filesystems) a project just because you want to add a machine. If your system is running while you want to upgrade, add the "-S" option, mln will boot the machines which have been rebuilt or added. This is handy when users are active on your virtual network while you upgrade.

```
mln upgrade -S -f new-project-file.mln
```

Chapter 4

Starting and Stopping

Every host and switch has their own start and stop scripts, similar to system init scripts. When a project is started, all start-scripts are run in alphabetical order. There is support for setting a boot order on each host. The default position is 99 (last). Any number smaller then 99 will have precedence. The stopping happens the same way, except that the stop scripts have the reversed order, meaning 99 will be taken down first. So machines that boot first will be taken down last.

```
mln <start | stop> -p project-name
```

Note: you don't have to specify the path of the project, only its name. MLN will look for that project in it's project directory.

Hosts can also be started and stopped individually within a project like this:

```
mln <start | stop> -p project-name -h hostname
```

4.0.1 Setting splay-time to slow down booting and shutting down

Starting a project with many hosts can tax a system and is often the most resource consuming part of the virtual network. To ease the process, you can issue a pause between every host to ease the pressure:

```
mln <start | stop> -s seconds -p project-name -h hostname
```

4.0.2 Choosing between xterm, screen, and none

Even though you decided on one way the vm should start, you can also set this at boot-time using the -t type option. Currently, "screen", "xterm", and "none" are supported. Example:

```
mln start -s -p project-name -t screen
```

4.0.3 What projects are running?

You can view the status of your projects with this command:

```
mln status
```

Your output will then look something like this:

```
################ MLN - Status #################
dmz-lan host choke-firewall down
dmz-lan host gateway down
dmz-lan host server down
dmz-lan host workstation down
dmz-lan switch dmz-switch down
dmz-lan switch lan-switch down
external_switch switch ext down
flab host choke1 down
flab host choke2 down
rh host dummy up
rh host redhat up
rh switch lan up
```

Chapter 5

MLN Syntax

The philosophy behind the syntax is that is should be easy to create simple networks and possible to create complex ones. The more features you want to put into your project, the longer the project file gets. But it should always be easy to read the project file and understand the functionality of the network.
Every project needs to have a global block where the name of the project is stated. This block looks like the following:

```
global {
    project project_name
}
```

What follows can be one or more hosts and a set of switches if desired. A project could simply be a group of machines not connected together but all of them conencted to the lan. Let us have a look at the main language features.

5.1 Language Features (superclasses and variables)

Writing simple networks does not require much work and you should be able to have a good result after only a few lines. You might want to tweak the network a bit, and start to add users and different root passwords to the virtual hosts. One machine might need an extra network interface so that it can function as a gateway for the rest of the virtual network, and so you add a few more lines. Steadily your project file grows. To ease the task of maintaining larger projects, we added support for inheritance and variables. Through inheritance, you can specify a superclass for a group of hosts. Every host that is set to inherit from that superclass will inherit that configuration. Locally specified attributes will override the inherited value. Variables can be used to make sure the same value is placed correctly several places, like the ip address of your nameserver or the the template filesystem you want to use.

You do not have to use these features in the project file, but when you are writing large network projects, you fill find it much easier to correct errors, typos and to add new features this way. Here is an exxample that uses both inheritance and variables:

```
global {
     project syntax-example
     $standard_memory = 64M
}

superclass common {
     free_space
     memory $standard_memory
}

host one {
    superclass common
}
```

Here, we define a variable "$standard_memory" already in the global block and we use it in the superclass. Host "one" will inherit the settings from the superclass. You can have hierarchies of superclasses, but a host can only inherit from one superclass. In the next example, we override the global variable and we also insert the variable into a text string:

```
global {
     project syntax-example
     $standard_memory = 64
}

superclass common {
     free_space
     $standard_memory = 128

}

host one {
    superclass common
    memory $[standard_memory]M
}
```

The resulting string is now "128M" for the host "one". Notice how the variable name is enclosed in brackets when inserted into text.

5.1.1 Keywords and values

The configuration is generally contructed from either blocks or keyword-value pairs. A keyword-value pair is not written with any assignment operator like =} or \texttt{:=, but straight forward:

```
memory 64M
```

Usually we put one keyword-value pairs separate lines for elegance, but this is also possible:

```
memory 64M; term screen
```

5.1.2 Blocks

Blocks are enclosed by curly brackets. They are usually on the form of:

```
block {
    line1
    line2
}
```

Exceptions to this rule are hosts, switches and network interfaces, which all have an extra parameter to them:

```
host one {
    network eth0 {
        address dhcp
        switch lan
    }
}

switch lan {
}
```

The reason for this is to keep compatability with earlier versions of MLN. The reader of the plugin chapter later in this manual, will discover that MLN creates sub-blocks out of these parameters when it builds its internal data structure.

5.1.3 Including other files

It is possible to spread the configuration into separate files and to include them into other configurations. In order to do so, you use the \#include keyword. It can be used anywhere in the configuration, and the MLN parser will simply continue on the next file as if it was the same file:

```
#include /my/other/config.mln
```

5.2 Syntax in depth

5.2.1 The global block

This block contains all the gloabal information for the project and is also the place where you define variables and assign values to them. Possible keywords and blocks are:

project <name>
The name of your project. If not specified, the build tool will prompt you for a name.

beforeProjectStart \{ \}
Run a list off commands before the project starts. Example:

```
global {
   project xen_on_lan
   afterHostsStart {
       echo "You can connect to the virtuam machine using 'screen -r xeno'
    }
}
```

beforeHostsStart \{ \}
Run a list of commands after the switches have started, but before the hosts are started.

afterHostsStart \{ \}
Run a list of commands after the Hosts have started.

afterProjectStart \{ \}
Run a list of commands after the entire project has started.

5.2.2 Switch blocks

Each switch block defines one instance of a switch. Usually, only the name of the switch is enough, but some extra features are available. The range of features for a switch depends on wether it has User-Mode Linux or Xen virtual machines connected to it. Mixing of the two on the same switch is currently not supported, although it is not impossible to achieve. Possible features are:

For User-Mode Linux

type <type>
This is the type of network component you want. The uml_switch has the opportunity to act as a hub. This will be enabled if you supply type hub in the switch block. Default is a regular switch.

socket <path>
Every network component opens up a unix socket and listens on it. The virtual machines will connect to that socket if they want to send through that switch. You can specify that the socket should be placed somewhere else, e.g. outside the projects directory. This is useful when you want to connect different projects together.

tap <tap-device>
With this option, you can connect the switch to a tap device. If the tap device is connected to a ethernet bridge on you computer, then every virtual host connected to that switch will be on your LAN. See the command enable_bridge for more information.

owner <user>
The owner of the socket for a switch.

group <group>
The group that owns the socket for a switch.

sudo <user>
The owner of the socket and process of a switch.

Xen

bridge <bridge_interface>
Usually, the switch will define a name for the bridge interface, but you can override ig with this option.

5.3 The host block

The host block is the most complex part of the mln syntax. It is not necessary to assign a value to each keyword, so you can get away with pretty small blocks of code for simple hosts.

5.3.1 Scalar Keywords

filepath <path>
Use a different path to store the filesystem images of the virtual machines. This is used if you wish to place the filesystems on a SAN or distributed filesystem. he name of the image will be "hostname.project"

swap <size>
This keyword adds a swapfile to the vm. Example:

```
swap 128M
```

owner <user>
The owner of this host's filesystem image. Currently only for User-Mode Linux.

group <group>
The group that owns this host's filesystem image. Currently only for User-Mode Linux.

cow_filesystem basename

Assign this host to use a copy-on-write filesystem with basename as filesystem base for reading. *Note: Copy-on-write filesystems are not currently supported in Xen.*

sudo <user>
Implies owner and assumes root is the one that runs the host's start script. The effect is that although it is started by root, the other user owns the filesystem image and owns the process. Currently only for User-Mode Linux.

size <size>
The size of the filesystem for this host expressed in megabytes and with a trailing "M". I.e 250M. Note, that this size needs to be larger than the size of the template in order to make it fit in. Default value: 250M.

free_space
With this keyword you can set how much space should be added to the template, giving you at least that amount of free space on the host. This keyword will override size. There will always be residual free space on the template to begin with, so the actual amount of free space will be this much or more.

A special case is this: "free_space 0M". The host will then end up with the size of the template, giving you the smallest possible size of that host.

term [xterm|screen|none]
This keyword describes how the virtual machine should start. It usually needs a terminal to which to connect its console. There are three options here:

1. You start the machine in an xterm. The xterm will open when you start the given machine but will terminate when you log out.

2. You start the machine in a backgrounded terminal using screen. You can then connect to the machine's console at your leisure using the command screen -r hostname. This is the recommended option if you want the project to run for a while and/or have a lot of machines.

3. For Xen-based machines, you can choose to have *no* terminal manager by specifying term none. For non-Xen machines, mln reverts to the default. This is possible since the Xen management command, xm, incorporates the

features provided by screen. To list virtual machines, use xm list, and to access a virtual machine use, e.g., xm console myhost.myproject.

Default value: xterm. As of version 0.73, this value can be set at boot-time too, using the "-t term" option with the start command. Example:
mln start -t screen -p myproject

color <color-name> This keyword makes only sense if xterm is the terminal used. It sets the color of that particular xterm when the virtual machine starts. This helps distinguish the xterms. The background color is always black. Default front color is lightgrey.

root_password <encrypted password>
Specify the root password. Supply the encrypted variant of the password. No default.

template <template>
Specify what template you wics to build this host from. The template needs to be downloaded AND registered beforehand.

nameserver <ip>
IP address of nameserver.

memory <amount of ram>
The amount of RAM memory for this host when it is started. This amount is not fully used until necessary, meaning that the whole amount is not locked at startup. Default 32M

boot_order <priority>
If you want any machines to start before someone else, assign them a lower boot order. A value of 1 means highest priority. Several hosts can have the same priority. Default value is 99, which is also the lowest priority. Note that the shut down order is automatically determined by the boot order. Machines that are booted first, are shut down last.

superclass <name>
The superclass of this host. Superclasses are a way to gather information about a class of machines. If the machine has a superclass, it will inherit all variables from that class. A host can also overwrite the keywords locally. Note that also superclasses can have superclasses of their own, creating a hiearrchy where only the leaf nodes ar actual hosts. Only Hosts are build.

kernel <path-to-kernel>
You use this if you want to specify a special home-grown UML kernel for this virtual machine. Write the absolute path of the kernel to avoid errors. Remember to add the modules_dir keyword too, if you need to copy any modules. Example:

```
kernel /opt/uml/linux2.6.4
```

modules_dir <dir>
Copy the modules from this directory. Usually used together with the kernel keyword. Example:

```
modules_dir /opt/uml/modules/2.6.4-1um
```

lvm [lvcreate options]
Please read the LVM chapter for an introduction on how to use LVM.

xen
Use the Xen virtual machine instead of the default User-Mode Linux.

hvm
Use the Xen hardware virtualization.

vncpassword <cleartextpassword>
(HVM) set a password for the Xen HVM VNC session.

vncdisplay <displaynumber>
(HVM) Use a specific VNC display number. (The port number will be 5900 + displaynumber)

5.3.2 Host blocks

modules
What modules are to be loaded ad boot time. The presence of this block will copy all the available modules for the kernel into the filesystem. The ones listed in the block will be written into /etc/modules. So if you want to have modules, you at least need this empty block. Example:

```
modules {
nat
tun
}
```

users
Add users to the virtual machine. You have to supply and encrypted version of the password. The syntax is like this:

```
users {
name password [homedir] [uid]
.
```

```
.
}
```

The home directory and UID are optional. Adding users is not supported on the busybox filesystems.

startup

Commands that are to be run at each boot. They will be placed in a bash script /etc/init.d/startup and this file is linked to from /etc/rc2.d/S99startup. Example:

```
startup {
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
route add -net 10.1.1.0 netmask 255.255.255.0 gw 10.0.0.3
}
```

mount

This block contains all the other filesystems you wish to mount in addition to the root filesystem. These can be both images and folders and can reside anywhere on the host machine. You can also choose any mountpoint you like except for the root. Note, that these filesystems will not be mounted during the building phase, meaning that you cannot as of now copy any files into the filesystem or use it as /home while you add users. We will address this issue in the next versions, however. Example:

```
mount {
/disks/backup.ext2 /mnt/backup ext2 defaults
/folders/www-data /var/www hostfs ro
10.0.0.1:/my/mnt /mnt nfs defaults
}
```

The filesystems supported are decided by the virtual machine kernel. “hostfs” for direct access to folders on the host are only supported in User-Mode Linux. NFS is also possible, but it assumes that the filesystem and the uml kernel has the appropriate software. The last field, containing the options, can be omittet. In that case “defaults” will be written in /etc/fstab.

network iface

Configure a network device. Example for DHCP:

```
network eth0 {
        switch < switch-name | socket >
        address dhcp
}
```

Static IP example:

```
network eth0 {
       switch < switch-name | socket >
       address x.x.x.x
       netmask x.x.x.x
     [ broadcast x.x.x.x ]
     [ gateway x.x.x.x ]
     [ mac x:x:x:x:x:x ]
     [ slirp [ slirp path] ] (for UML)
     [ bridge <bridge-name> ] (for Xen)
}
```

If the MAC address of the interface is not specified, it will be generated randomly.

The simplest way to setup networking for a User-Mode Linux is using the slirp software package. It provides NAT-like acces to the internet and supports UDP and TCP traffic (no ICMP). You need to have slirp installed on your system. You can also download compiled slirp binaries from the MLN site. Here is an example:

```
network eth0 {
      slirp
}
```

An interface connected to a TUN/TAP device: (UML)

```
network eth0 {

       tun_iface <iface_name>
       tun_address <tun_address>
       address x.x.x.x
       netmask 255.255.255.252
     [ broadcast x.x.x.x ]
       gateway <tun_address>
}
```

This last example makes a point about the nature of the TUN/TAP connections. They involve only two addresses, and you get away with a much smaller netmask. It also a good idea if the gateway for the virtual hosts interface points the physical host side of the connection. The TUN/TAP can be called whatever you want, and it is enabled and destroyed automatically by mln if started and stopped by root. In order to utilize this functionality but still run the host as a different user, see the owner} and \texttt{sudo keyword.

Connecting a Xen host to the network is particlary easy, because youa re most likely already root and the xend daemon has set up the proper requirements for you. If you omit both the switch and the bridge from a hosts network interface, then MLN will assume that it should be connected to the "xenbr0" bridge. This bridge device is set up by xend 3.0.1 and later and is connected to your lan already. Here is one such host:

```
host one {
    network eth0 {
        address dhcp
    }
}
```

The interface settings can be changed to a static address if you do not have a dhcp server on your lan.

files

Use this block to specify what files are to be copied into a virtual host's filesystem at boot time. The files you want to copy have to be in the directory configured as your files-directory. If you are unsure where that is, run mln write_config. Example:

```
files {
foobar /root/foobar 644
scripts/scpecial_script.sh /usr/local/rum-me.sh 755
}
```

Note that the first field is the path to the file you want to copy relative to your files folder. The second field is where in the virtual machine's filesystem you want to put the file. The last field is the permissions the file shall have.

groups

You can add groups and assign users to groups. The following example will create a group called admin and assign the user jack to it:

```
users {
    jack lkjlkadlfasd
}

groups {
     admin {
         jack
     }
}
```

Block Type	function
global	Contains name of project and variable declarations. Suitable place to put information regarding the entire project (i.e. for plugins to use
switch	An ethernet switch shared amongst one or more virtual machines
host	Each host block corresponds to one virtual machine in the project.
superclass	A special class for grouping configurations which other virtual machines can inherit from. Usefull when you have lots of similar virtual machines in the same project. Hierarchies of superclasses can be built.

Table 5.1: The different 1. level blocks.

5.3.3 Summary

5.3.4 Inheritance

The most organized way to keep a consistent configuration is through superclasses. A superclass is configured simply as a host, but it will not be built into a VM. Other hosts can inherit the configuration from a superclass by using the superclass keyword. Here is an example:

```
superclass common {
        term screen
        memory 64M

        network eth0 {
           netmask 255.255.255.0
           gatway 10.0.0.1
        }
}

host one {
    superclass common
    network eth0 {
       address 10.0.0.2
    }
}

host two {
    superclass common
    network eth0 {
        address 10.0.0.3
    }

}
```

In this example the hosts one and two inherit from the superclass common.

Administration

Chapter 6

The MLN daemon, Distributed virtual networks and Migration

If you have more than one server for virtual machine hosting, then there is a chance you want to spread projects accross those servers but still manage them from single commands. The MLN daemon is a way to achieve this.

The daemon runs as a process on each server and recieves instruction regarding MLN projects from one or more authorized sources. From the user perspective, one writes the MLN project on one host, and builds it just like before. MLN will then detect if the project is distributed and attempt to contact the other servers and send them the project as well. The same goes for starting/stopping/upgrgading and removal of projects. Another aspect is the collection of status information in order to monitor serveral servers and make decisions based on their free resources. The MLN daemon provides a specialized status command that lets the user see the ammount of projects and virtual machines running on each server. For the servers that use Xen, the status command collects output from the xm list command and displays that as well. We will show examples of this later in this chapter.

In this chapter we walk through the few steps involved in setting up the MLN daemon and writing distributed projects.

6.1 Base Setup

6.1.1 MLN Daemon setup

Consider the following example: We have three servers, master, backend1 and backend2. The master is our main MLN server and has no need to run the daemon, as all the MLN commands will be issued there. The servers backup1 and backup2 are dedicated MLN servers which are controlled mainly from the master and therefore need to run the MLN daemon. The easiest way to transform an unin-

stalled machine into an MLN dedicated server is through the specialized install CD, which you can find a link to here: LINK MISSING. But any machine where MLN is installed can run the mln daemon.

In MLN terms, a server that runs a part of a project is called a service_host. It provides a service to the virtual machines, i.e keeping the filesystem and letting it run.

Lets look at the necessary configuration. The MLN daemon does not allow any connections by default, so we need to define the IP addresses of the hosts we want to allow. This is done in the /etc/mln/mln.conf file on each of the backend servers:

```
daemon_allow 128.39.73.10
daemon_allow 128.39.74.*
```

Here, we set that the host with IP address 128.39.73.10 and all host on subnet 128.39.74.* are allowed to connect to the daemons. Further, we need to give the backends a necessary ID so that they understand which part of the project is to be buildt on them. The ID is their service host tag, and will be used when writing projects later. It has to be either an IP address or a relsolve-able name. The most natural is to use their hostnames. Here is how it would look on backend1:

```
service_host backend1.vlab.iu.hio.no
```

Lastly, we define the ammount of memory reseverd for the server itself. This is not acted upon by the MLN daemon, but helps with the status output to quickly see where there is resources to add more virtual machines. For Xen users, the default reserved ammount is 192 MB. If the backend is installed thorugh our specialized installer CD, the reserved ammount is 128MB.

```
daemon_max_memory 128M
```

Once this is added to the /etc/mln/mln.conf file, we can start the server the following way (as root if you run Xen):

```
master:~# mln daemon
```

If you wish to start the daemon in he background, add the **-D pidfile** option like this:

```
master:~# mln daemon -D /var/run/mln.pid
```

6.1.2 The Master server

There is little configuration needed on the machines that will send projects to the service hosts. The first thing needed is a service_host tag here as well because the projects will be spread out accross all three servers. In the /etc/mln/mln.conf at master we set the following:

```
service_host master.vlab.iu.hio.no
```

Next we need to define that master should collect daemon status information from the two backend servers when we issue the mln daemon_status command. So we add the following two lines to the mln.conf file:

```
daemon_status_query backend1.vlab.iu.hio.no
daemon_status_query backend2.vlab.iu.hio.no
```

Here also, we set the ammount we would like to reserve for the server itself:

```
daemon_max_memory 192M
```

We are now ready to write a distributed project and build it.

6.2 Writing a distributed project

A distributed project is not much different from a regular one, except that one uses the service_host tag on the hosts and switches to decide where they shall be placed. Here follows a distributed project, where we place one virtual machine on each service host:

```
global {
      project dtest
}

superclass common {
      xen
      nameserver 128.39.89.10
      network eth0 {
            netmask 255.255.255.0
            gateway 128.39.73.1
      }
}

host one {
    superclass common
    network eth0 {
          address 128.39.73.11
    }
```

```
    service_host master.vlab.iu.hio.no
}

host two {
    superclass common
    network eth0 {
        address 128.39.73.12
    }
    service_host backend1.vlab.iu.hio.no
}

host three {
    superclass common
    network eth0 {
        address 128.39.73.13
    }
    service_host backend2.vlab.iu.hio.no
}
```

Make sure the MLN daemons are running on all your servers before you start the build. The project can be buildt with the usual command:

```
master:~# mln build -f dtest.mln
```

MLN will send the project to all other service hosts before doing the build itself. That way, all the servers can do their share in paralell. Once the build is done at the main server it will start query the other servers for their output until everyone is done.

The project is started with the usual: `mln start -p dtest`.

6.3 Collecting status information

You can get the status information from either special projects or the servers itself.

6.3.1 Project status

```
mln status -p projctname
```

6.3.2 Server status

```
mln daemon_status [-s]
```

The daemon status command will query all the daemons and gather statistics from them. With the -s option, only a summary will be printed. Without it, you will be presented with detailed data from each daemon, especially if you run Xen. In

that case you will get the output from xm list with some additional information from each server.

6.4 Migration

MLN supports migration of virtual machines from one service host to another through the upgrade command. Lets say we have a third backend server, backend3, and want to move one of the virtual machines over to it. The way we do this is by making a copy of our original project file and edit the service_host line for the particular host we wish to move. This is an excerpt of that file:

```
host three {
    superclass common
    network eth0 {
        address 128.39.73.13
    }
    # notice how the next line has changed:
    service_host backend3.vlab.iu.hio.no
}
```

Now, we issue the upgrade command from our main server. Note, that all the involved servers need to have their MLN daemon running at this point. Especially the two servers involed in the migration process:

```
master:~# mln upgrade -f dtest2.mln
```

The server, backend2, which is where the vm three is located prior to the upgrade will shut down the vm and await contact from the new service host. The server being the new service_host for the vm three will contact the other server and fetch the compressed filesystem image. Once it is transferred, it will do the other changes which migh be on the upgrade list.

6.4.1 Live Vs Cold migration

Xen supports live migration, meaning the ability to move a running virtual machine from one location to another without shutting it down. For this feature to work, one needs to have a shared network storage of the filesystem so that both involved servers can access the filesystem simultaneously. Further both servers need to be of the same CPU architecture and on the same subnet.

MLN does at this version not support live migration. The method currently used, cold migration, means shutting the vm down and moving the filesystem to the other location. This method might sound inferior to live migrations promise of seamless migration and uptime, but there are som benefits to MLN's approach as well:

- The migration can be to any other location. No same subnet is required.
- One can change platform of the server, i.e go from a Intel-based server to an AMD-based one.
- One can change virtualization platform and and system variables in the same process. You could start out with a light-weight User-Mode Linux VM and migrate it to a Xen virtual machine with more memory.
- It does not require shared network storage of the filesystem images.

Unless uptime is of the absolute importance, cold migration is a suitable option for most.

6.5 SANs and live migration

MLN supports live migration of entire networks using Xen, as long as the virtual machines filesystem is placed on a concurrent network storage. SANs using AoE (ATA over Ethernet) or NFS or other distributed filesystems are supported. MLN needs to get a hint, that a filesystem placed in a certain location actually is on a san. Use the san_path option in the mln.conf file to point to distributed filesystems, like this:

```
san_path /mounts/mlnsan
```

Two servers will attempt to live migrate IF:

- Both servers are connected to the SAN and aware of it
- The virtual machines run Xen
- The virtual machines are up
- The respective xend daemons accepts migrations from the other servers

The live migration is triggered the same way as a normal upgrade whith different service_host}. \textbf{IMPORTANT: If you plan to live migrate, you should ONLY change the service host and nothing else of the virtual machines. Remember also, to change the service host of the switch.

Chapter 7

Backups (Export/Import)

Backing up virtual machines is convenient as you don't have to take care of backup software inside each of the VMs. MLN does not offer a backup management system, it does not take backups regularly nor does it support rotation. MLN offers a way to extract a project into a convenient form so that you can build whatever backup scheme you want around it.

7.1 Taking backups with MLN export

MLN can save projects into directories and compress them. This can be utilized to create "snapshots" of an entire project. The snapshot will contain the following:

- The projects mln configuration file
- A folder "images" which contains all filesystems from each virtual machine

The way to create such a snapshot, is to invoke the **mln export** command:

```
mln export -p myproject -d myproject_july_2008 -z -s
```

This command will now stop all the virtual machines belonging to the project "myproject" (-s option). It will then create a folder called "myproject_july_2008" and store all filesystems there. Next, it will create a tarball from the directory (-z option).

7.2 Restoring projects with MLN import

MLN can read a folder or tarball created with the export command and use it to restore a project to a previous state:

```
mln import -p myproject -d myproject_july_2008.tar.gz
```

Note, that this is a crude way of doing backups and that some finer-grained backup scheme, perhaps running as software inside the VM could complement this. When restoring a project, the filesystem is restored to a previous state, which will have an effecto on logfiles and timestamps and the like. You should test it in a scenario best describing a real emergency on your site before relying on it.

7.3 Distributed backups/restores

These commands also work if the project is spread accross many servers. In that case, all filesystems will be extracted from the remote servers and stored in the same folder for convenience. Likewise, when importing, all filesystems are transported back to the server they belong to. Make sure that the MLN daemons run on all servers and that they allow access from the particular machine performing the backup.

Chapter 8

Setting Ownerships

MLN is able to set the ownership of virtual machines and switches, making it possible to run your projects as someone else then root, even if you are root when you build. This is recommended if you plan to have some security on your projects. This is also handy if parts of the network are to be owned by different users, typically in class.

There are three keywords you may use for this purpose: *sudo, owner* and *group*.

- **sudo user|uid**
  This keyword is used if you plan to run the host as a user account that normally does not correspond to a human, or a special user. The project is still started and stopped by root. The sudo command is incorporated into the start-script of the host. The application sudo has to be installed on your system for this to work.

  Also, this may cause problems when the term for the host is set to "xterm". Users can't normally open windows in others' X sessions.

- **owner user|uid**
  Here, the purpose is to build the host for somebody else. Building as root is faster then as a regular user. With this keyword, you can build a project where ownership is spread among several users. These user can then start and stop those hosts themselves as long as their project points to the same folder (this can be set with the -P dir option at command time too).

- **group user|uid**
  sets the group ownership on the filesystem image. This one is most useful for switches that are started as root but you want write access for other users that are in a special group too.

Switches that have external sockets but run as a specialized user need to have write permission in the folder where the socket is stored. Further, MLN does not create those users, they have to exist beforehand.

8.0.1 Example: Starting as root, but running as someone else

Part of the network setup is done in the actual start-script for a host, so running the script itself as root can prove convenient.

```
global {
project own-test
}

switch lan {
      group uml-net
}

superclass host {

    sudo mln-user
    group uml-net

    term screen

    network eth0 {
              switch lan
          netmask 255.255.255.0
          broadcast 10.0.0.255
     }
}

host te1 {
superclass host

network eth0 {
       address 10.0.0.1
       }

network eth1 {
       tun_iface owtest
       tun_address 192.168.0.1
       address 192.168.0.2
       netmask 255.255.255.252
       gateway 192.168.0.1
}

}
host te2 {
    superclass host

    network eth0 {
      address 10.0.0.2
      }
}
```

In this example, the entire project is built and started as root, but the running instances will belong to the user called mln-user. One of the hosts, te1, has an extra network interface connected to a tunnel device. This is set up properly by MLN as long as the project is started as root.

The screen and the host processes will belong to mln-user and he can connect to its console.

Storage

Chapter 9

Using the iSCSI plugin for Xen enabled MLN servers

ISCSI is an exiting storage area network (SAN) technology, which enables you to share block devices over a network. A block device, as opposed to a filesystem, will be viewed by the client as a new device. All IO will be passed on directly to that device and no local filesystem caching will be done. It is connection based, which means you set up one connection per shared device. Using iSCSI has become popular with regard to virtualization because of its stability and performance.
The MLN iSCSI plugin enables you to utilize an iSCSI SAN backend with MLN. It can use iSCSI targets which are set up users, or it talks to a daemon backend which will dynamically deploy new iSCSI shares using LVM to partitions on one or more backend servers. This allows a very flexible way to deploy new virtual machines with centralized storage, a prerequisite for live migration. Once everything is set up, you can easily deploy new projects and all the iSCSI parts will be taken care of transparently.
This guide will explain the following topics:

- How the MLN plugin works
- Setting up a backend iSCSI server for use with MLN
- Installing iSCSI software and the plugin on MLN servers
- How to use the MLN plugin with iSCSI hardware
- Additional notes on administering iSCSI-based virtual machines

9.1 The iSCSI backend server

The backend server is any Linux machine running the iscsitarget software, which is available for many different recent Linux distributions, such as Ubuntu 8.04. It also resides over a LVM volume group, which will be the repository for the VMs

disks. In addition, the backend server may have a set of MLN filesystem templates stored locally for ease of deployment. This is not a requirement, however. Finally, the backend server also runs a simple daemon which will accept requests from each MLN server when new VMs are built. You can set up any machine as a server and also have several servers function this way.

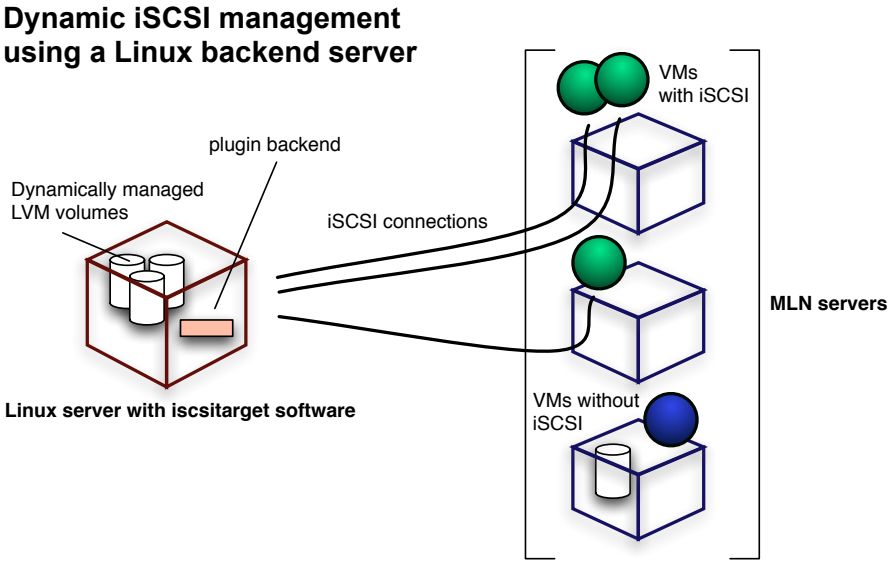

Figure 9.1: The iSCSI backend daemon and iscsitarget software will transform a Linux server into a dynamic backend for filesystem storage.

9.2 How the build process works with the iSCSI plugin

The goal of the plugin is to handle automatic creation and connections to iSCSI targets. This should happen in a transparent way to the user. Here follows a description of what happens under the hood:

1. A new project is built on one of the servers, containing a host with the iscsi [server] line. For example:

```
global {
    project iscsitest
}

host example {
    xen
    iscsi sanserver
}
```

2. During the build process, the iscsi.pl plugin will be run and it will look for that line. If found it will send a message to the iscsi-backend daemon running on the iSCSI backend server. The message sent to the server will contain the following information:

   - The name of the project
   - The name of the VM
   - The size of the filesystem
   - The template used

3. The iscsi-backend.pl daemon does not care about projects, only disks. Upon receiving the message It will go ahead and create a LVM volume called "example.iscsitest" with the required size. Next it will check if it has a local copy of the template used. If so, it will copy it on the volume and resize it. If not, it will leave the volume as is and expect the MLN server to copy the template over at a later time. This will certainly be an impact on the network, so for larger templates which are used often, it is strongly recommended to keep a copy on the iSCSI backend server. After the disk is ready, a new volume is dynamically added to the iscsitarget software using the ietadm command and the corresponding information written to /etc/ietd.conf in case of service restarts and reboots. A message is sent back to the MLN server saying if the disk was created and also if a local template was found. In addition, a iSCSI URI is returned, which the MLN server can use to connect to it.

4. Once the message is received on the MLN server that the process is complete it will connect the iSCSI volume using the open-iscsi initiator software. The

disk is then mounted up and MLN can continue to configure the filesystem according to its specification. When the configuration is done, the disk is unmounted and the iSCSI connection is closed. The start and stop scripts will also be edited by the plugin so that every time the VM is started or stopped, the connection will be established or closed respectively.

9.3 Using the MLN iSCSI plugin with iSCSI hardware

Some users have iSCSI hardware running at their site instead of a Linux server. Naturally, they can't use the approach where they install software and have volumes created dynamically. That changes the the way in which the plugin functions. Instead, the user will have to create the disks manually and share them using the interface of the iSCSI hardware. Once these disks are created, the user needs to decide which virtual machines use which disks.

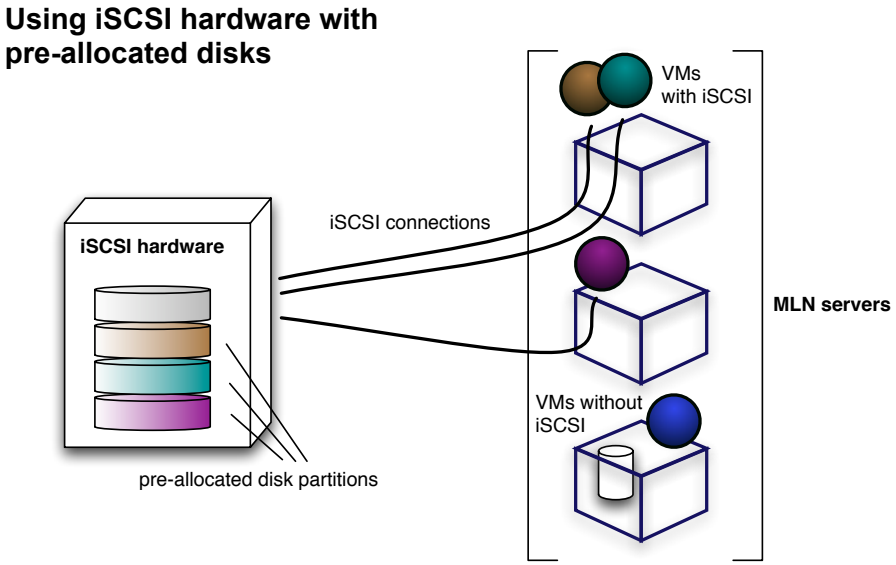

Figure 9.2: The iSCSI plugin can also be used together with iSCSI hardware were there are pre-configured targets.

The MLN syntax will be slightly different in this case. Say the user has set up some shares on the iSCSI hardware box called iqn.example.org:mln.vmdisk.1, iqn.example.org:mln.vmdisk.2 and so on. For the host "example", vmdisk.1 is chosen as the drive. The syntax now looks the following:

```
host example {
    iscsi sanserver
    iscsi_target iqn.example.com:mln.vmdisk.1
}
```

Note: The user needs to be sure that the disk is large enough to accommodate the template.
MLN will simply use this resource and not attempt to talk to the iSCSI hardware and simply connect to the server. This may have some implications: The iSCSI plugin will now copy the template onto the disk resource and that may introduce performance hits while this is going on.

9.4 Setting up the iSCSI backend server

These are the necessary steps involved in setting up the backend server. This has been tested using a standard Ubuntu 8.04 install. If you are using iSCSI hardware, you can skip this section.

9.4.1 Create a LVM volume group

What disks you want to use is up to you. Let's say you have only a single block device, /dev/md0, which is a software RAID of some sort. But this could also be a single hard-drive or partition.

```
pvcreate /dev/md0
vgcreate mlnsan /dev/md0
```

9.4.2 Install iscsitarget

From Ubuntu 8.04 and up you can install iscsitarget from the package repositories: apt-get install iscsitarget
You can also download and install manually from http://iscsitarget.sourceforge.net. Installing iscsitarget will also install the ietadm tool which allows dynamic additions of the iSCSI service.

Test the iSCSI server before moving on

Lets create a disk image and test the iSCSI software. Do the following steps and go over the corresponding section on "Preparing the MLN servers".

1. Create a volume and call it “testdisk”. Put a filesystem on it.

```
lvcreate -L 1GB -n testdisk mlnsan
mkfs.ext3 /dev/mlnsan/testdisk
```

2. Start the iscsitarget daemon and find a vacant target ID number:

```
/etc/init.d/iscsitarget start
cat /proc/net/iet/volume
```

   Just pick a TID which was not in use by any of the existing volumes. (If you only just installed iscsitarget, you will probarbly have no targets and the number 1 will work just fine). Use the number every time you see [number] in the next steps.

3. Add a new target and connect the LVM volume to it

```
ietadm --op new --tid=[number] --params Name=testdisk
ietadm --op new --tid=[number] --lun=0 --params Path=/dev/mlnsan/testdisk,Type=blockio
```

4. Check again if the volume is shared:

```
cat /proc/net/iet/volume
```

   You should now be able to see the target:

```
tid:[number] name:testdisk
        lun:0 state:0 iotype:blockio iomode:wt path:/dev/mlnsan/testdisk
```

Move over to corresponding section “Preparing the MLN servers” 9.5 where you will install open-iscsi and attempt to connect to this target.

9.4.3 Download the iscsi-backend script

The latest version of the backend script can be found in the Plugin section of the MLN homepage. It is a tarball containing the deamon itself and a start/stop script for the /etc/init.d folder.

```
tar xzf MLN-iscsi-backend-*.*.tar.gz
cd MLN-iscsi-backend
cp iscsi-backend /usr/local/bin
cp init.d/iscsi-backend /etc/init.d/
```

The script contains its own configuration, so before you run it, you need to edit the script. This is covered next.

9.4.4 Configure iscsi-backend and define security boundaries

The script needs to be made aware as to what LVM volume group to use and what IPs to accept requests from. Changing these settings is done very quickly: Open the script in a text editor and edit the following parts (they are onlu in the beginning of the script)

1. First you need to define the prefix name of all your iSCSI target that the script will create:

```
my $ISCSI_PREFIX = ‘‘sanserver.mydomain.com:’’;
```

2. First you need to define the prefix name of all your iSCSI target that the script will create:

```
my $ISCSI_PREFIX = ‘‘sanserver.mydomain.com:’’;
```

3. The \$LVM_VG variable needs to contain the name of your volume group:

```
my $LVM_VG = ‘‘mlnsan’’;
```

4. Next, you define where the script should look for pre-uploaded templates.

```
my $TEMPLATE_PATH = ‘‘/mln/templates’’;
```

5. Finally you need to set the access permissions. The script will allow commands based on IP addresses or ranges which are defined in an array. You will have to define the IPs or ranges as perl regular expressions. Here are some examples:

```
# only the IPs 10.0.0.1, 10.0.0.2 and 10.0.0.3:
my @ACCESS_LIST = (‘10\.0\.0\.1’,‘10\.0\.0\.2’,‘10\.0\.0\.3’);

# The range 10.0.0.*
my @ACCESS_LIST = (‘10\.0\.0\.\d+’);

# The range 10.0.0.* and 192.168.0.3
my @ACCESS_LIST = (‘10\.0\.0\.\d+’,‘192\.168\.0\.3’);
```

Save and close the file. You are ready to try the script.

9.4.5 Run the iscsi-backend script

The script can be run in the foreground or in the background as a daemon. It is advisable to run it in the foreground when you test it for the first time, in order to familiarize yourself with the output. TO run it in the foreground, simply run iscsi-backend as a command. You will have to run it as the user root, due to the other commands which this script execute.

If you want to run it in the background, you can either use the start script which was provided with the tarball:

```
/etc/init.d/iscsi-backend start
```

Or you can run it with the detach option from the command line:

```
iscsi-backend -D /var/run/iscsi-backend.pid
```

In background mode, all output will printed in the logfile /var/log/iscsi-backend. You can keep track of progress using this command:

```
tail -f /var/log/iscsi-backend
```

You will also find messages in syslog regardless if it is run in the fore- or background. The messages contain less output then the main logfile, but keep you posted on the general activity. Here is an example of the daemon starting and a filesystem being created:

```
Nov 27 14:42:41 sanity iscsi-backend[6950]: MLN iSCSI backend (v. 1.0) on mlnsan [ 81.90G left]
Nov 27 14:43:45 sanity iscsi-backend[6950]: sa9client.sa9 created,size 3.77GB - time: 50s
```

In order to stop the backgrounded daemon, you can use this command:

```
/etc/init.d/iscsi-backend stop
```

9.4.6 (Optional) Create local repository of templates

In order to reduce the impact on the network when building a new VM, it is strongly recommended to copy templates which you use often to the backend server. Decide upon a directory where you want to put them, and simply transfer them using something like scp. Make sure to specify the location in the iscsi-backend script (look in the configuration section of the script).

```
my $TEMPLATE_PATH = ‘/mln/templates’’;
```

You can use whatever method to copy the templates, i.e scp. Make sure they are not compressed.

9.5 Preparing the MLN servers

The MLN servers are the servers which run the MLN software and the virtual machines. There are only few steps necessary to make iSCSI work on them. This has been tested on Debian 4.0.

9.5.1 Download and install the open-iscsi software

The open-iscsi project is under active developement and using one of the stable releases from their site might be a better choice than the package that follows the distribution. During the time of writing, version 2.0-869 worked best in our setup, so here follows the steps to install it:

1. Download the latest stable open-iscsi package.

   ```
   wget http://www.open-iscsi.org/bits/open-iscsi-2.0-869.2.tar.gz
   ```

2. Install a compiler and header files. **Note:** Make sure you use the correct header files depending on your Xen Dom0 kernel.

   ```
   apt-get install gcc linux-headers-2.6.18-6-xen-amd64
   ```

   You can get the correct kernel version by running the uname -a command.

3. Compile and install

   ```
   tar xzf open-iscsi-2.0-869.2.tar.gz
   cd open-iscsi-2.0-869.2
   make
   make install
   ```

It is advisable to double-check the version number both on the open-iscsi service and the iscsiadm tool before moving on.

```
iscsiadm -V
```

Should produce:

```
iscsiadm version 2.0-869
```

And for the service:

```
/etc/init.d/open-iscsi start
grep iscsi /var/log/syslog
```

Should produce (amongst the entire output):

```
Nov 28 15:00:02 legolas iscsid: transport class version 1.1-646.iscsid version 2.0-869
```

Now that iSCSI is running on the server, it is time to test if we can connect to the backend iSCSI server.

Test the iSCSI connection before moving on

If you followed the instructions above in Section 9.4.2, you should now be able to connect to the disk you created on the server. In the following instructions, make sure that sanserver is exchanged with the name or IP address of the server running the iscsitarget software.

```
# Connect to the iSCSI target
iscsiadm -m node -T testdisk -p sanserver -o new
iscsiadm -m node -T testdisk -p sanserver -l
```

Check if the session is active:

```
iscsiadm -m session --show
```

Should produce something like:

```
tcp: [137] [sanserver]:3260,1 testdisk
```

The name of the disk-device will vary, based on the order of iSCSI devices connected. Instead of using the devices /dev/sdX it is better to get the accurate name from /dev/disk/by-path/. As a last check, lets mount the disk:

```
mount /dev/disk/by-path/ip-sanserver:3260-iscsi-testdisk-lun-0 /mnt
```

If this was successful, you are ready to proceed. However, it may be a good idea to close the connection first:

```
umount /mnt
iscsiadm -m node -T testdisk -p sanserver -u
iscsiadm -m node -T testdisk -p sanserver -o delete
```

Verify that the session is closed:

```
iscsiadm -m session --show
```

The session should now be gone.

9.5.2 Install the iscsi.pl plugin

The latest version of the MLN iSCSI plugin can be found on the plugin section of the MLN homepage. If you want to see if the plugin is installed and what version, you can run the following command:

```
mln write_config
```

The command will print out the configuration and default values. At the end of the output, look for:

```
iSCSI backend plugin version X.X
```

If the plugin is not installed, simply download the iscsi.pl plugin from the MLN homepage and copy the plugin to MLNs plugin directory:

```
cp iscsi.pl /etc/mln/plugins
```

Repeat the `mln write_config` command.

9.5.3 Setting up san_path in mln.conf

This step is necessary if you plan to use serveral MLN servers and you want to live-migrate virtual machines between them. MLN does not understand by default which machines are conencted to an external storage. For MLN to understant that disks shared by iSCSI are network shared, you need to provide a hint in the `/etc/mln/mln.conf` configuration file, add the following line:

```
san_path /dev/disk/by-path/ip-
```

This line will explain to MLN that all the locations of disk drives that start with `/dev/disk/by-path/ip-` are on a SAN. You are now ready to test iSCSI using MLN.

9.5.4 Building a project using the iSCSI plugin

This example project consist of only one virtual machine with its disk connected through iSCSI.

```
global {
      project mini
}

host me {
    xen
    iscsi sanserver
}
```

Before you build, make sure the iscsi-backend daemon is running on the sanserver.

```
mln build -f mini.mln
```

Amongst the output you should see som iSCSI-realated lines, such as:

```
---> me
ISCSI plugin enabled
Template Debian-3.0r0-V1.1.ext2
Contacting sanserver for ISCSI partition
got sanserver:mln.me.mini, but need to copy template manually
Waiting for block device to appear
Copying image with dd to iSCSI device, please wait.
210668+1 records in
210668+1 records out
107862336 bytes (108 MB) copied, 23.7579 seconds, 4.5 MB/s
Running fsck on filesystem
e2fsck 1.40-WIP (14-Nov-2006)
Resizing filesystem
resize2fs 1.40-WIP (14-Nov-2006)
iscsi_createFilesystem did filesystem creation
ISCSI plugin enabled for mountFilesystem action
Waiting for block device to appear
Configure host Xen
Importing modules from: /lib/modules/2.6.18-6-xen-686
Adjusting /etc/inittab for XEN
Writing XEN configuration file for me, done
ISCSI plugin enabled for umountFilesystem action
+---> PROJECT mini FINISHED
```

Although the iSCSI plugin will display extra information, most of the details are hidden. One interesting part of the output is the result of the `dd` command. This will be invoked if the template is not stored beforehand on the sanserver and means that that the MLN server is now pushing the template over the iSCSI connection onto the new disk. This process may put some strain on your network. It is therefore strongly recommended to put a copy of often-used filesystems om the sanserver as

described in Section 9.4.6.

It is now possible to boot the virtual machine:

```
mln start -p mini
```

This should produce the following:

```
Connecting to iSCSI target sanserver:mln.me.mini
Starting me.mini in screen
```

The virtual machine should boot normally. You can connect to it and verify that it cam up properly.

To shut down the virtual machine, run:

```
mln stop -p mini
```

This will output the following:

```
me... OK
Initiating background wait loop before disconnecting iSCSI device
```

The last line needs a bit of explanation. The MLN servers are not connected to all the iSCSI filesystems all the time. In fact, only when the virtual machine is started, the iSCSI connection is initiated. Likewise, when a virtual machine is stopped, the connection is terminated. However, we cannot simply shut down the iSCSI connection whlie the virtual mahcine boots down. Instead we need to wait for it until it has gracefully shut down until we terminate the iSCSI connection. This is achieved by starting a background process which will check the virtual machines status every 3 seconds until it is down. Once it is gone, the process will terminate the connection and itself.

This works also if you choose to destroy a virtual machine. In that case the wait will be very short.

9.6 Working with iSCSI - Some useful notes and tips

Before this chapter is at an end, we have collected some notes which should beuseful for admins and users who wish to run virtual machines using iSCSI. If you are unfamiliar to iSCSI, we highly recommend you glance over the topics listed underneath.

9.6.1 Keeping track of iSCSI sessions

You may at times wonder what MLN servers are connected to what disks and also which volumes are shared by the iSCSI backend at any moment. This information may be important if some virtual machines have gone down without your knowing[1]. In those cases, the iSCSI connection may still be there even if the VM is down. On the MLN servers, the following command will show you what sessions are currently registered:

```
iscsiadm -m session --show
```

On the iSCSI backend using iscsitarget, you can check both which volumes are shared and which sessions are active. This information is located in two files in the /proc directory. In order to view all sessions, type the following:

```
cat /proc/net/iet/session
```

If you want to get more information about the volumes which are shared in thos sessions, type:

```
cat /proc/net/iet/volume
```

9.6.2 Manually connecting to an iSCSI volume

MLN will automatically disconnect from an iSCSI volume and remove all traces of it once the VM is taken down using the mln stop command. If you wish to mount the filesystem on your MLN server in order to inspect it or run a filesystem check on it, you need to connect to the iSCSI volume manually. This is not hard. The only information you need is the name of the volume, which you can get by looking at the /proc/net/iet/volume file on the iscsi backend server. If the name is example.com:mln.me.mini, and the name of the iscsi backend server is sanserver, then you can connect using the following command:

```
iscsiadm -m node -T example.com:mln.me.mini -p sanserver -o new
iscsiadm -m node -T example.com:mln.me.mini -p sanserver -l
```

Take a look at the next section in order to mount the virtual machines disk, after you have connected. You may disconnect using the following commands:

```
iscsiadm -m node -T example.com:mln.me.mini -p sanserver -u
iscsiadm -m node -T example.com:mln.me.mini -p sanserver -o delete
```

[1] Like if the users have run the poweroff command inside the VM

9.6.3 Location of iSCSI disks devices on MLN servers

Normally, when you conenct to an iSCSI volume, you will have a disk device created for you which looks like your other disks, like /dev/sdg and so on. But this may easlily lead to confusion if you have many different devices conencted in different order. After a while you will loose track of which of these devices correspond to what virtual mahcines disk.

A better way to access the devices, is to use one of their more proper names. These names can be found in the /dev/disk/ folder. The most readable of the names are the ones underneath the by-path subfolder. All iSCSI connected disks will begin with ip- and contain the name of the volume too. This way it is very easy to see which one belongs to what VM. As an example, the disk corresponding to the VM me.mini which we created above will have the device like

```
/dev/disk/by-path/ip-sanserver:3260-iscsi-example.com:mln.me.mini-lun-0
```

The example.com part may look diffrerent based on wow you configure the iscsi-backend script. You can treat that disk like any other disk device and mount it:

```
mount /dev/disk/by-path/ip-sanserver:3260-iscsi-example.com:mln.me.mini-lun-0 /mnt
```

Remember, that if this is a disk for a fully virtualized virtual machine, you need to jump past the partitioning table and master boot record:

```
mount -o offset=32256 /dev/disk/by-path/ip-sanserver:3260-iscsi-example.com:mln.me.mini-lun-0 /mnt
```

9.6.4 Restarting the iSCSI service - beware!

Many online howtos which describe adding new volumes to the iscsitarget service use the same basic approach:

1. Edit /etc/ietd.conf and add the new volume to be shared.
2. Restart the iSCSI service:

   ```
   /etc/init.d/iscsitarget restart
   ```

This method works if you previously had no targets or sessions running. If you already have virtual machines which are running over iSCSI it is advised **not** to restart the service. The potential consequence is that the MLN serverl hickups its session to the volume and that the virtual machine will be unable to do IO to the

disk for a period of time. The virtual machines reaction to this is to switch the filesystem into read-only mode. A mode it will not recover from until you have rebooted the virtual machine.
This type of error can be very hard to spot from the outside, because the virtual machine will still respond to PING packages and let you log into it. However, some services inside the virtual machine may break and that is how you probarbly will notice it.

The best course of action is to not restart the iscsitarget service if you have running virtual machines. If you need to add volumes manually, then look at the iscsi-backend script and how it uses the ietadm command to dynamically add volumes without restarting the service.

9.6.5 Number of threads per iscsitarget on the backend server

In the default setup of iscsitarget, every shared volume will use eight threads. This is from the moment it is shared, not only when there is a connection from a MLN server. This may lead to a very high number of processes on the iscsi backend server and is something one should take into account. With 50 volumes, 400 threads will be on your system only for them.
Linux is powerfull enough to handle this, but you should make sure you have enough CPUs to cope with it. If you start seing performance issues as the number of volumes increases, you should also keep an eye on the number of context switches your system is performing every second. Tools like Munin may give you a nice overview over how the operating system is doing.

9.6.6 More than one iSCSI server

You can limit the pressure on the iscsi backend server by setting up more servers. You can follow the exact same procedure as described above to set up additional servers. From MLNs point of view, the iSCSI server is the value following the iscsi keyword.

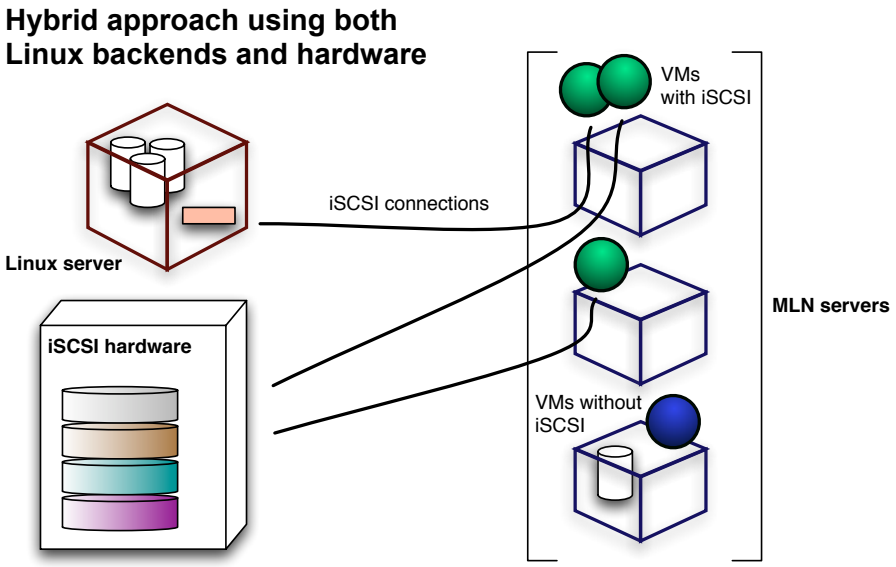

Figure 9.3: The iSCSI plugin is not exlusive in terms of backends. You can use several backends, both hardware and more dynamic ones.

You can even use different iSCSI servers for different virtual machines in the same project:

```
global {
     project example
}

host one {
    xen
    iscsi sanserver1
}

host two {
    xen
    iscsi sanserver2
}
```

9.6.7 Backend network for improved performance and security

For added security, it is advisable to keep all the iSCSI traffic on a sepparated newtork which is not reachable from the internet. This will also greatly reduce the risk that a virtual machine which is very active on the network degrades the disk performance of all the virtual machines. Most production environments today have sepaprated storage networks.

It is also advisable to have good network cards with solid drivers on this backend

network. Using the ifconfig command, look for dropped packages on the device pointing to the storage network.

Chapter 10

Using MLN in Amazon's Elastic Computing Cloud (EC2)

IMPORTANT: Using Amazon EC2 costs money

Although it is free to create an account in EC2, actual usage will be charged on your credit card. Using MLN to manage instances in EC2 will therefore also result in usage and, subsequently, money charged. MLN can not be held responsible for what you spend on EC2 and you use the software on your own risk. It is strongly advised to keep a close eye on your instances in EC2 at all times.

Cloud computing is becoming a buzzword now that virtualization technologies have established themselves in the marked. This infrastructure-for-sale concept can be a valuable resource for researchers and business, looking for a quick way to deploy numerous systems, often for a short period of time, without managing the server hardware.
The ability to utilize such frameworks is of importance for researchers and engineers who need to deploy test systems rapidly or to enhance local server capacity by scaling the number of compute nodes for a business infrastructure. Knowing how cloud frameworks work is essential for designing and developing autonomic architectures which can run on top of them.
Amazon's Elastic Computing Cloud (EC2) is a popular Xen-based cloud computing framework for businesses, private and research. It consists of a set of services and an API which offer a degree of flexibility and pricing along with a increasing wealth of tools to manage its virtual machine instances. The pay-as-you-go philosophy enables users to experiment with little cost.
This chapter teaches you how to manage virtual machines in the EC2 cloud the

same way as you would manage local Xen instances through the common MLN commands. MLN's design features such as projects, templates and plugins work well with EC2 and expand the functionality of what EC2 alone offers. While most tools only let you manage instances manually, setting up 10 instances each with special permanent storage attached to them can become cumbersome. MLN lets you automate most of the process, and is therefore especially suited for larger deployments.

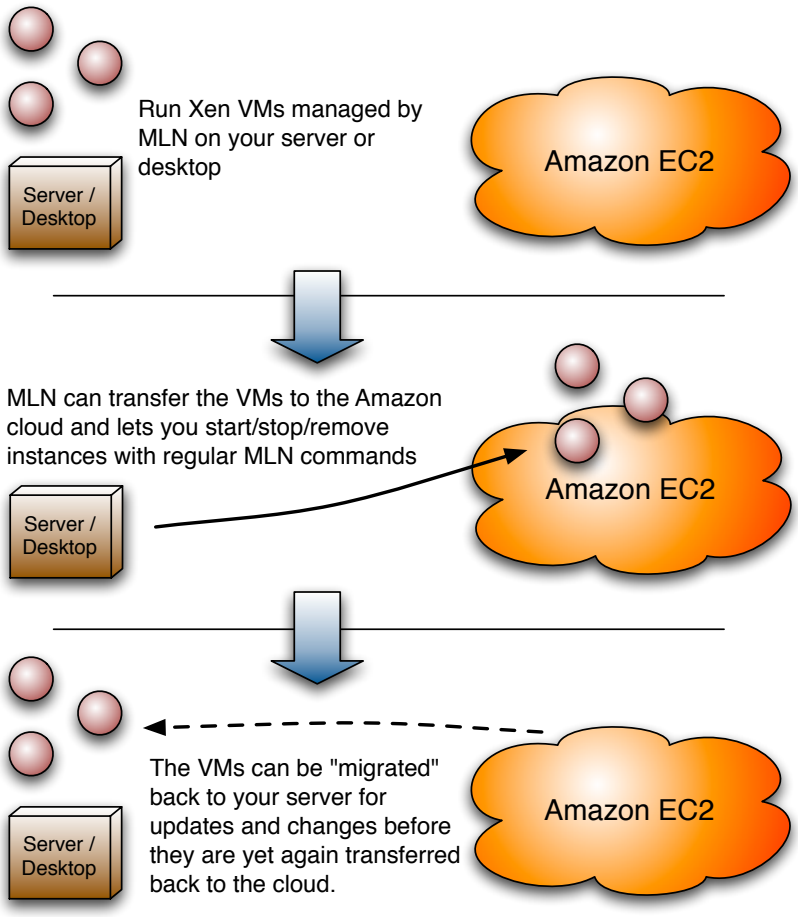

Figure 10.1: The MLN EC2 plugin allows you to migrate virtual machines into EC2 and manage them like regular MLN projects.

With the MLN plugin it is also possible to migrate Xen instances from your local server and into the EC2 cloud. This has many applications, such as creating a small cluster locally and moving it into the cloud only when you need lots of processing power and are prepared to pay for it. Also, it is possible to transparently increase virtualization capability by building temporary virtual machines in the cloud and using them to support local infrastructure while your local servers are already filled

up with virtual machines.

10.1 Amazon Elastic Computing Cloud

This section is for those who are unfamiliar with the EC2 framework. Also, note that some details may have changed over time and as a result, the information in this manual may not have been properly updated yet.

10.1.1 Instance types and machine images

In Amazon jargon, a VM is called an *instance*. Normally, you would define the hardware setup of a new virtual machine based on your needs, but in EC2 you are left with picking from specific choices. The types differ in amount of memory, storage space, CPUs and EC2 compute units. A Compute unit is simply a measurement of the power of the CPU, different from GHz. One EC2 compute unit (ECU) corresponds to about 1.0 - 1.2 GHz 2007 Xeon processor.

Every instance will have its root partition on a relatively small disk partition (max 10GB). In addition, they have one or two disks with much more capacity attached to the VM. You can specify the mountpoint in through the MLN EC2 plugin.

There are currently five instance types to chose between: The difference in comput-

Type	Arch	Memory	Disks	CPUs and ECUs
Standard Small (m1.small)	i386	1.7G	1 x 160 GB	1 CPU with 1 ECU
Standard Large (m1.large)	x86_64	7.5G	2 x 420 GB	2 CPUs with 2 ECUs each
Standard Extra Large (m1.xlarge)	x86_64	15G	2 x 845 GB	4 CPUs with 2 ECUs each
High-CPU Medium (c1.medium)	i386	1.7G	1 x 350 GB	2 CPUs with 2.5 ECUs each
High-CPU Extra Large (c1.xlarge)	x86_64	7G	2 x 845 GB	8 CPUs with 2.5 ECUs each

ing power and storage space is quite large. A small instance is suitable for a rarely used service or for a service which requires little power. The larger instances have considerably more power both in terms of ECUs and memory and can be utilized for computations on an on-demand basis. Each of these types have different costs, which we will review in a later section. This means that in order to minimize expenditure, you should pick an instance which is as close to your needs as possible and not over-provision.

10.1.2 Amazon machine images (AMI)

When creating a new virtual machine, one also picks what filesystem it is being booted from. In EC2 these filesystems are called AMIs (Amazon Machine Images). EC2 does not support booting from a CD or install media and installing on a blank disk. You have to have ready-made filesystems.

All EC2 users can create and upload their own AMIs, which are nothing short of regular VM filesystem images. The actual uploading and storage is slightly more complicated, and utilizes the Amazon Simple Storage Service (S3). All EC2 users have an account on S3 as well. Once you have uploaded a machine image, it will be given a unique identifier of the form ami-[random text and numbers], like ami-4338df2a. You use this identifier when you wish to boot an instance of it. MLN will handle these instances for you.

It is possible to make these AMIs available to other EC2 users. Tools such as rightscale.com or the elasticfox plugin to Firefox, lets you browse these and create instances from them directly.

It is important to note, that no changes which an instance makes to the AMI will be saved when the instance shuts down. This enables that you can boot multiple instances from the same AMI. We will discuss this in section 10.1.5.

10.1.3 Networking, Elastic IPs and Security groups

EC2 only supports DHCP on the virtual machines. This means that you cannot assign fixed internal IPs to each instance directly. Further, it means that each time an instance is booted, its address will have changed.

Each instance is given two addresses: one private, which is the IP address which the VM knows about through DHCP, and a public one which can be used to contact the VM from the outside. Each VM instance is also part of a security group. You can look at security groups like a set of firewall rules which apply to that particular VM. These firewall rules address the port, subnets and protocols which are allowed to access the instance through its publicly routable address.

There are ways to get a fixed address to your instance. One alternative is to use a dynamic DNS service, like dyndns.org. The other alternative is to subscribe to so-called elastic IPs which are offered by the EC2 framework. Every user can currently have a maximum of 5 elastic IPs per region (but you can apply for more). If you assign one of your elastic IPs to a running instance, it can be reached through that address. However, this does not change the instance's internal address. Also, because of the scarceness of IPv4 addresses, you are charged for the time you have an unassigned elastic IP address. This means, that you *don't* pay for it as long as you are utilizing it.

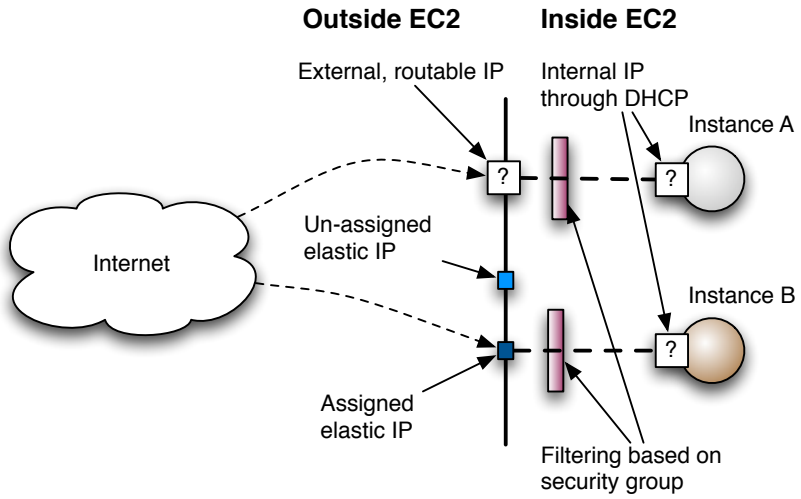

Figure 10.2: Each instance has an internal IP which it got from DHCP. The external IP is random as well and enable outside contact to the instance. Elastic IPs can be used to assign a fixed routable IP address to a running instance.

10.1.4 Permanent storage through Volumes

Since no changes are being stored to the AMI, where are you going to save your data? EC2's answer to this is to use their Elastic Block Store (EBS) to create disk volumes which are permanent. These volumes can be attached to running instances and mounted, using them as a fixed storage point.

EC2 supports making snapshots of EBS volumes and reverting back to them. This is convenient way of taking backup of your data.

EBS volumes can only be attached to one instance at a time. It can not be used as a distributed storage amongst several instances. Also, the instance has to run in the same availability zone as the volume in order attach to it.

10.1.5 How will a VM differ in the cloud from when it is on your server?

One should spend some time and think about the difference between what a VM in EC2 and the same running on your local machine. Here is a list of the most important differences:

- **No changes to the VMs filesystem will be saved**
  Yes, this is a potentially big deal. Once the instance is shutdown or has crashed, all the stored data will be gone. This means that you need to think closely about how you are going to handle permanent data. EC2's answer

to this is to create EBS volumes, which are permanent and can be attached to running instances. The MLN EC2 plugin can automate this for you. One positive side to this, is that for multiple, similar instances, you only need one machine image. So for a cluster of 32 compute nodes, only one AMI needs to be uploaded.

Some have come up with a neat workaround to this issue, by putting their systems filesystem into an EBS volume and booting only a minimal instance which pivot-roots into the EBS volume. This way you can get persistent images with a little tinkering. Read the forum thread for more info: http://developer.amazonwebservices.com/connect/thread.jspa?threadID=24091&start=0&tstart=0

- **Only one NIC and no fixed internal IPS**
  The consequence is that you will have a hard time pre-configuring services to use specific IP addresses. You will also have difficulties creating internal network topologies since you only have one NIC. The MLN EC2 plugin supports assigning elastic IPs but there is also a dyndns plugin which can be used.

- **Fixed root partition and hard drives**
  All instance types have specific disk sizes and numbers. You have to be smart on how to mount them at boot time. For instance, if you have an image with users already on it, you cannot simply mount a disk on /home as that would blind the existing home directories. The MLN plugin supports specifying the mount points of the disk drives.

Still, with these limitations, EC2 seems to be a popular choice among engineers. Once one is familiar with them and knows a few workarounds, systems can still be created which function pretty well in the cloud.

10.1.6 Regions and Availability Zones

The entire cloud spans two continents. Significant difference in network times can be achieved based on where you chose to locate your instances. Currently, two regions exist: US and EU, called us-east-1 and eu-west-1 respectively. Each of these regions consist of so-called availability zones.
All availability zones are physically and network-wise separated so that an outage would only affect nodes within the affected zone. Instances will be put into zones and it is advisable to put instances which are going to communicate heavily into the same zone. It is not possible to "share" the same zone as a different EC2 user. In fact, the actual zone mapping is different between users.

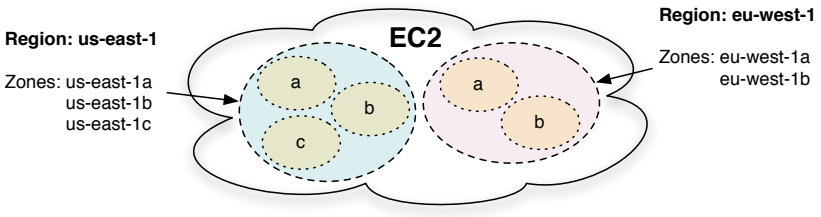

Figure 10.3: The entire EC2 framework is divided into two regions, US and EU with three and two availability zones respectively.

10.1.7 Pricing

Getting a user account on EC2 is free and no usage will result in no charges. This is a benefit with EC2, since it allows gradual and careful experimenting before one commits larger amounts of money to it.

EC2's pricing is based on usage and uptime metrics on the different services they offer. The more powerfull, complex your instance is, the more you pay. The more network in and out of the cloud, the more you pay. One interesting thing is that you pay for the time the instance is up, not the amount of CPU cycles it has consumed while being up. This means that an instance which no-one uses practically seeps money. The new type of system administrator needs to be smart about this and create dynamic deployments which are only up when they are supposed to be used. This ability strongly affects the end-outcome on your monthly bill.

The following prices are based on numbers gathered in March 2009:

- **Instances, per hour**

Region	**US**		**EU**	
Type	Linux/UNIX	Windows	Linux/UNIX	Windows
m1.small	$0.10	$0.125	$0.11	$0.135
m1.large	$0.40	$0.50	$0.44	$0.54
m1.xlarge	$0.80	$1.00	$0.88	$1.08
c1.medium	$0.20	$0.30	$0.22	$0.32
c1.xlarge	$0.80	$1.20	$0.88	$0.128

- **Data Transfer**
  This is "in" and "out" of Amazon EC2 itself. (Not traffic from instances themselves.)

Data In	
All Data Transfer	$0.10 per GB
Data Out	
First 10 TB per Month	$0.17 per GB
Next 40 TB per Month	$0.13 per GB
Next 100 TB per Month	$0.11 per GB
Over 150 TB per Month	$0.10 per GB

- **Regional Data Transfer**
  This is data transferred between instances within the same region but in separate availability zones.

Data In/Out	
All Data Transfer	$0.01 per GB

- **Elastic IP address**

  If you have elastic IP addresses, you will be charged for the period in which they are unassigned. Mapping an address to an instance is free for the first 100 maps per month.

Elastic IP addresses	
Non-attached elastic IP per hour	$0.01 per hour
First 100 remaps	$0.00 per month
Additional remaps over 100	$0.10 per month

- **Amazon Elastic Block Store**
  Most long-term deployments will need permanent storage of some sort. If you use EBS volumes, then you are charged based on the following metrics:

Region	**US**	**EU**
EBS Volumes		
Provisioned storage (GB) per month	$0.10	$0.11
Per 1 million I/O requests / month	$0.10	$0.11
EBS Snapshots		
GB of data stored / month	$0.15	$0.18
Per 1,000 PUT requests when saving a snapshot	$0.01	$0.012
Per 10,000 GET requests when loading a snapshot	$0.01	$0.012

The prices, partitioned like this, make it difficult to comprehend the final cost of a actual running system. I have therefore created three examples with calculations in order to showcase what the costs would be like. I have not double checked these numbers with officials from Amazon, so please do the math yourself if you want to be completely sure.

Amazon offers a monthly pricing calculator for cost estimation:
http://calculator.s3.amazonaws.com/calc5.html

Example: Small server with little traffic

Let's assume you want to run one small instance which is up all the time and has only about 3GB of network traffic each month. The uploaded filesystem bundle is 1,5 GB. The running instance has two volumes, one for logfiles (5GB) and the other for users (20GB).

We will disregard the cost for keeping the AMI in S3, which would be $0.376 each month. The remaining cost for this instance is shown in the table below.

The numbers are rounded in the output for the sake of formatting. The most

	Storage		**Network**		**Uptime**		**Total**	
	Month	Year	Month	Year	Month	Year	Month	Year
US	$2.5	$30	$0.406	$4.87	$74.4	$892.8	$77.31	$927.7
EU	$2.75	$33	$0.406	$4.87	$81.84	$982.1	$85.0	$1019.95

striking finding in this example is the dominance of uptime costs for the instance relative to storage. If one would find a way to schedule the instance's uptime as to only be up when really needed, the cost can be cut dramatically. Also notice, that the EU is close to %10 more expensive than US.

Example: Website with lots of traffic and permanent storage

In the next example, we will consider a larger site with three webservers, each running a type c1.medium instance. In addition there is a database server of type m1.large. Each of the webservers have 5GB volumes for logfiles and 2GB volumes for web content. The database has a 50GB volume for logging and a 100GB volume for the database. The expected traffic totals to 5GB in and 30GB out per month for the entire site.

Again, we disregard the cost of keeping the S3 machine images and the small investment it was in uploading them to S3.

	Storage		**Network**		**Uptime**		**Total**	
	Month	Year	Month	Year	Month	Year	Month	Year
US	$17.1	$205.2	$5.6	$67.21	$744.0	$8928	$766.7	$9200.4
EU	$18.81	$225.72	$5.6	$67.21	$818.4	$9820.8	$842.8	$10113.7

Again, the costs for uptime dwarf the costs for storage and networking. Even though we have simplified the calculation and disregarded volume snapshots for backups, something which would be very likely, the uptime for four semi-powerful instances contributes to over %97 of the total estimate. This should be especially

concerning if the site is not fully utilized most of the time. Unfortunately this is the common scenario for most sites. Sysadmins, therefore, need to be smart about their deployments and scale based on the actual need instead of leaving everything on all the time.

Let us now assume, that the sysadmins are able to boot up webservers based on the actual usage. As a result only one webserver is needed %70 of the time while two webserver were running %20 of the time and only %10 of the total time called for all three webservers to be up. In the original calulation, the number of machine hours for the three webservers was calculated as follows:

$$24 * 31 * 3 = 2232$$

With the scaling of the system, we have instead:

$$24 * 31 * (1 + 0.3 + 0.1) = 1041.6$$

The difference is 1190.4 compute hours, corresponding to a reduction in cost of $238.08 per month and $2856.96 annually for the US. So with simple starting / stopping of instances according to the load can reduce the total cost with %31 in our case. Again, these numbers are only examples and one should always do the analysis based on local factors. However, it is obvious that these kind of reductions are the holy grail for many admins (and sales people). One rarely goes into the discussion of how many man-hours are needed to manage a dynamic site rather than keeping the static one, something which can be costly to overlook.

Example: 16 node (128 CPU) cluster which is only used when needed

In this example, we are interested in an on-demand cluster. Imagine a case where a small company needs computing power only on special occasions and that they cannot afford to buy and maintain their own infrastructure. This could be a media company which create small animated movies like commercials or documentaries. Their solution is to use EC2 to boot up their 16 node render farm each time someone has to render a movie.

The render farm consists of 16 c1.xlarge instances, the most powerful EC2 offers. This gives a sum of 128 CPUs (or cores) and 112 GB of RAM to the cluster. Every node has 1GB of permanent storage for logs. In addition, there is a manager node of type c1.medium. Its job is to provide a VPN link to the companies network and manage the compute nodes. The traffic is relatively small, except when a larger movie in high quality is completed and subsequently downloaded into the companies network before the cluster is shut down. In case the download should fail, we assume a 150GB storage for the manager node for eventual downloads. Note, that the compute nodes can be shut down as soon as rendering is completed and only the manager node stays up for the duration of the download of the movie. The

network speed between EC2 and the company is measured at 6 Mbit.

Calculating the cost per month makes no sense here, so we will calculate it for each job instead. Lets imagine a rendering job which takes 5 hours to complete. The result is a movie file of 4.7 GB. We assume the size of the job is 150 MB. Uploading the job to the manager node takes less than 5 minutes. Downloading the resulting file will take approx. two hours.

Region	Storage	Uptime	Network	Total per job
US	($15*)	$65.6	$0.872	$66.47
EU	($16.5*)	$72.16	$0.87	$73.03

Table 10.1: * the price for the EBS volume is per month and not per job

This example is a very good case for cloud computing as a concept. The ability to rent large amounts of infrastructure for short periods of time can be very cost-effective. Only $66 to render a movie in a 128 CPU cluster sounds cheap (though I am not a rendering expert.) Notice that the cost of the EBS volume is in the table, but not in the sum at the end. The reason is that the cost of the volume will be permanent each month, and not directly related to the individual job. Also notice, that IO requests to the volume are not part of the calculation. This is mostly because it is difficult to assess how many IO operations are needed.

10.2 The MLN EC2 Plugin

Amazon offers an API to their storage and cloud framework. As a result, several tools have emerged. MLN makes use of two bundles of command line tools to manage filesystems and instances respectively. You can practically do everything from the command line once the bundles are installed and configured. However, setting up an instance and managing it can be cumbersome for more than one instance. One example is a lack of inherent grouping mechanism in EC2. Further, the naming scheme for all the components in EC2 (instances, machine images, volumes) makes it difficult for a human to keep track of what runs where and is connected to who.

MLN tries to mitigate these issues by offering its management commands to the user and keeping track of all the details under the hood. You can go and find these details yourself, but most of the time they are just in the way. Through MLN you can build a VM which is running in EC2 instead of on your local machine. The plugin tries to be as transparent as possible, so once the EC2 command-line bundles are installed, MLN will call the proper commands for you. This is the typical build process of a new VM which is managed through MLN:

1. You specify inside that particular VM that it is an EC2 instance by adding a `ec2` block. (We will cover all available ec2 keywords later in this chapter.)

```
host nomad {
        xen
        template Debian-4.0.ext3
        ec2 {
                type m1.small
        }
        network eth0 {
                address dhcp
        }
}
```

2. During the build process this VM is built like a local VM. It gets a copy of its filesystem with eventual size metrics and configures it with regard to other MLN plugins or options (users, software etc.)

3. Once the filesystem is finished, the EC2 plugin will edit some files inside the VMs filesystem. Most notably /etc/fstab to fit with EC2s design and to add the single or two disk drives which every instance gets. Some startup scripts may be created as well, depending on the ec2 features you are using (like attaching to a volume once the instance boots). Also, the kernel modules which correspond to the kernel which EC2 is using are copied into the filesystem. This can be either i386 or x86_64 modules based on the chosen type.

4. The filesystem is now made into a so-called *bundle*. This is an Amazon packaging format necessary before the image can be uploaded. The bundle consists of of an XML file called the manifest, and a series of part-files which contains and encrypted and compressed piece of the image.

5. The bundle is then uploaded into the region where the instance belongs. This may take time depending on the network and size of the bundle. It is therefore wise to start with a small image first when testing.

6. The bundles are uploaded into *buckets*, which are like containers. Each region needs its own bucket. MLN will automatically create the buckets mlneu and mlnus for the EU and US respectively. Before the bundle can be used it needs to be registered. When the entire process is finished, MLN will have the ami identifier for the image and store it in the project folder.

7. Finally, start and stop scripts are created. They contain checks to see if the VM is already up and the necessary commands to create and shutdown the instance. The instance ID is stored in the project folder if MLN managed to start the virtual machine.

Since EC2 is based on Xen, it is recommended to use templates which you know work with Xen. However, it is not mandatory to have Xen installed on the system which is running MLN and the EC2 plugin. If you solely want to run instances in

EC2, no local Xen installation is necessary (which is good news for many laptops). That said, having Xen on the local machine gives you more features, like the possibility to run an instance on your machine and then move it into the cloud. This is great for testing, if you want to have local access to the VM first and configure it just right before moving it out and starting to pay for it.

10.3 Getting an Amazon account

In order to use EC2, you need to create an account first and then download a set of credentials which the API tools use when they connect to Amazon. Account creation is done on the http://aws.amazon.com. Once your account is created, you will need to go with your browser to "Your account" and "Access identifiers". EC2 requires you to keep track of several items. It is recommended that you create a directory, ~/.amazon, in your home folder and store these items there in order for the plugin to find them.

```
mkdir ~/.amazon
```

- **Your account number**
  This number represents your account. It is somewhat hidden away on the page (upper right corner, underneath the "Sign Out" link). If you can't find it, search for "Account Number" in the browser.
  Store this number in ~/.amazon/user.txt

  ```
  echo "<user>" > ~/.amazon/user.txt
  ```

- **Access key ID**
  This is a string consisting of letters and numbers.
  Store this string in ~/.amazon/access.txt

  ```
  echo "<access>" > ~/.amazon/access.txt
  ```

- **Secret Access key**
  If you don't have a key there yet, click on the "Generate" button to create one. Store this string in ~/.amazon/secret.txt

  ```
  echo "<secret>" > ~/.amazon/secret.txt
  ```

- **The X.509 Certificate file and private key**
  The easiest is to make Amazon generate the certificate right there for you and to download the private and key and certificate.

Store the certificate as ~/.amazon/cert.pem
Store the private key as ~/.amazon/pk.pem

```
echo "<private-key>" > ~/.amazon/pk.pem
echo "<certificate>" > ~/.amazon/cert.pem
```

10.3.1 Additional tools

Since the API is available to everyone, we have a range of tools to chose from, where MLN is only one option. The other tools are mainly graphical and some offer additional features with extra cost for business. Here is a non-complete list of tools which you can use in conjunction with MLN.

AWS management console

Amazon has launched their own management console (they hadn't before) which is readily available for all EC2 users. The console provides easy access to most of the basic functionality. It can be accessed here:

http://console.aws.amazon.com

RightScale

One example of a service which offers additional features is http://www.rightscale.com. Once you have an Amazon EC2 account, you can create an account on RightScale and manage your instances without extra cost.

RightScale has a nice interface for managing security groups.

Elasticfox

Elasticfox is an Amazon EC2 plug-in to the popular Firefox web-browser. It is easy to set up and install and allows you to manage instances and volumes.

Command-line tools

There is no reason to forget the command-line tools which will be installed anyway if you are going to use the EC2 plugin. Many simple commands can be run just to check the status of your isntances. For example, even if there is a way to check which virtual machines are up through MLN, many still use `xm list` because they prefer that output. Likewise, if you want to see what EC2 instances are running, you can type `ec2-describe-instances` or simply `ec2din`.

10.4 Setting up EC2 command-line tools on your machine

This section describes a step-by-step process of setting up the EC2 command-line tools on your system. Completing these steps is a prerequisite for the MLN EC2 plugin to work properly.

1. Install curl. Some Linux distributions do not have curl installed by default:

   ```
   apt-get install curl
   ```

2. Install Java runtime environment. If you are an Debian Lenny (Version 5.0.0) user, you need to download java from http://java.sun.com. The OpenJDK pacakges contain a bug (per march 2009) which makes the api tools unable to run.

   If: you run Ubuntu, or any other distribution with sun Java packages, install them with your package manager. Example from Debian Etch (4.0):

   ```
   apt-get install sun-java5-bin
   ```

   Else: Download the JRE binaries from http://java.sun.com, and install them. Make sure you download for the correct platform.

   ```
   chmod +x jre-6u13-linux-i586.bin
   ./jre-6u13-linux-i586.bin
   mkdir /usr/lib/jvm
   mv jre1.6.0_13 /usr/lib/jvm
   ```

3. Download API-tools

   ```
   wget http://s3.amazonaws.com/ec2-downloads/ec2-api-tools.zip
   unzip ec2-api-tools.zip
   mv ec2-api-tools-1.3-30349 /usr/local
   ```

   Note: The actual version numbers may have changed, do not copy this text verbatim.

4. Insert credentials or follow the steps described in 10.3

   ```
   mkdir .amazon
   echo “<private-key>” > .amazon/pk.pem
   echo “<certificate>” > .amazon/cert.pem
   echo “<access>” > .amazon/access.txt
   ```

```
echo “<secret>” > .amazon/secret.txt
echo “<user>” > .amazon/user.txt
```

5. Install AMI tools

```
apt-get install ruby libopenssl-ruby1.8
wget http://s3.amazonaws.com/ec2-downloads/ec2-ami-tools.zip
unzip ec2-ami-tools.zip
mv ec2-ami-tools-1.3-31780 /usr/local/
```

   Note: The actual version numbers may have changed, do not copy this text verbatim.

6. Make everything work at next login. Edit .bashrc:

```
export EC2_PRIVATE_KEY=/root/.amazon/pk.pem
export EC2_CERT=/root/.amazon/cert.pem
export EC2_AMITOOL_HOME=/usr/local/ec2-ami-tools-1.3-31780
export EC2_HOME=/usr/local/ec2-api-tools-1.3-30349
export JAVA_HOME=/usr/lib/jvm/java-1.5.0-sun-1.5.0.14
export PATH=$JAVA_HOME/bin:$EC2_HOME/bin:$EC2_AMITOOL_HOME/bin:$PATH
```

   Note: The actual version numbers may have changed, do not copy this text verbatim. Also, if you installed Java from Sun, make sure the JAVA_HOME path is set correctly.

10.4.1 Test the command-line tools

Once the lines are added to **.bashrc** you can test if everything works from a new shell. Simply start a new shell and try the following tests:

1. **Test the API-tools**
   This should work without any authentication and should print information about the different security groups (firewall rules) you can chose from. If you just created an account, only the default group will be printed. You can create new and change these groups yourself.

```
ec2-describe-group
```

2. **Test the AMI-tools**
   If the API tools worked, the AMI tools will probarbly work too, so there is no strict need to test if they work. However, if you want to be sure, or you are trying to debug a problem, this command will create a bundle for you from a filesystem:

```
ec2-bundle-image --batch -p test \
-i /opt/mln/templates/Debian-4.0r0-V1.0.ext3  -u $(cat.amazon/user.txt ) \
--cert .amazon/cert.pem --privatekey .amazon/pk.pem
```

Also, if you want to test wether uploading such a bundle works, the following command should do the trick:

```
ec2-upload-bundle -m /tmp/test.manifest.xml \
-a $(cat .amazon/access.txt ) -s $(cat .amazon/secret.txt ) -b mln
```

If the upload was a success you might as well delete the bundle after it was uploaded in order not to waste money storing it:

```
ec2-delete-bundle --bucket mln --manifest /tmp/test.manifest.xml \
-y -a $( cat .amazon/access.txt ) -s  $(cat .amazon/secret.txt )
```

10.5 Download a filesystem template

If you are already using MLN, then you probarbly have some templates installed. If not, then the Debian-4.0 and Lenny template have been tested and found to work in the EC2 cloud. It is a good idea to start with a simple template first and advance to bigger templates when everything works as expected. You can download a plugin using the command:

```
mln download_templates
```

10.6 Installing The Plugin

1. Download and install the latest version of the plugin. The plugin can be found on the plugin page on http://mln.sourcefoge.net. You can install it with the following command:

   ```
   wget -O /etc/mln/plugins/ec2.pl http://mln.sourceforge.net/files/ec2.pl
   ```

2. Download kernel modules for the default Xen 32bit kernel used by EC2

   ```
   wget http://s3.amazonaws.com/ec2-downloads/modules-2.6.16-ec2.tgz
   tar xzf modules-2.6.16-ec2.tgz
   mv lib/modules/2.6.16-xenU /opt/mln
   ```

3. Download kernel modules for the default Xen 64bit kernel used by EC2

```
wget  http://s3.amazonaws.com/ec2-downloads/ec2-modules-2.6.16.33-xenU-x86_64.tgz
tar xzf ec2-modules-2.6.16.33-xenU-x86_64.tgz
mv lib/modules/2.6.16.33-xenU /opt/mln/
```

10.6.1 Testing the plugin

You should be able to see the plugins version number when you run the **write_config** command:

```
root@server# mln write_config
[ omitted output ]
AMAZON EC2 plugin version 1.0
```

10.6.2 Configuring the plugin

If you saved some of the files on different locations, you need to edit the plugin and change some of its settings. This is not difficult and efforts have been made to make it as easy as possible. If you open up the plugin in an editor, you will find the following block at the beginning:

```
####################
# CONFIGURATION
####################
my $EC2_MODULES_i386 = "/opt/mln/2.6.16-xenU";
my $EC2_MODULES_x86_64 = "/opt/mln/2.6.16.33-xenU";
my $EC2_USER = "/root/.amazon/user.txt";
my $EC2_SECRET = "/root/.amazon/secret.txt";
my $EC2_ACCESS = "/root/.amazon/access.txt";
my $EC2_CERT = "/root/.amazon/cert.pem";
# Valid types: m1.small, m1.large, m1.xlarge, c1.medium, c1.xlarge
my $EC2_DEFAULT_TYPE = "m1.small";
my $EC2_DEFAULT_AVAILABILITY_ZONE = "";
my $EC2_DEFAULT_KERNEL = "";
my $EC2_DEFAULT_RAMDISK = "";
my $EC2_DEFAULT_GROUP = "default";
my $EC2_DEFAULT_MOUNT1 = "/mnt";
my $EC2_DEFAULT_MOUNT2 = "/mnt2";
my $EC2_DEFAULT_REGION = "US"; # Alternative: EU
my $EC2_DEFAULT_ARCH = "i386"; # Alternative: x86_64
####################
```

Many of these variables are self-explanatory and can be changed by you. The most important thing is to make sure the paths are correct.

10.7 Using the EC2 Plugin

Once the plugin is installed, the only thing you need to do to make an EC2 instance, is to include the ec2 block inside of the host declaration. This is a minimal project

with only one instance.

```
global {
        project ec2
}
host one {
        xen
        ec2 { }
        network eth0 {
                address dhcp
        }
}
```

The xen keyword needs to be present, as this is a Xen virtual machine, even when it runs in the Cloud. It is, however not necessary to have Xen installed on your system when you build. The network block is important, as this is the only way a virtual machine can get online (and you access it). Multiple network cards are ignored, as this is not currently supported by EC2 anyway.

The ec2 block works with superclasses. In this example, we have three hosts, all running in EC2:

```
global {
        project musketeers
}
superclass common {
        xen
        ec2 { }
        network eth0 {
                address dhcp
        }
}
host athos {
        superclass common
}
host porthos {
        superclass common
}
host aramis {
        superclass common
}
```

One can override ec2 settings locally in each host. You are not limited to having EC2-only projects. You can have local virtual machines in addition, just leave out the ec2-block. This makes sense if you want to deploy instances which are supposed to be visible on your local LAN. In that case you might want to have a local gateway VM as part of the project which the instances can open tunnels to and would provide local ethernet bridging to them.

The only way to access your EC2 instances is through SSH (but you can *read* the boot console if you need to debug something). Make sure you use templates

which have the SSH service installed and that you have users either on the template directly or include them in the MLN configuration directly. One idea is to use the MLN SSHkey plugin to automatically copy SSH keys into the virtual machines at build time for password-less access.

Building a project

A project is built the same way as regular projects:

```
mln build -f musketeers.mln
```

The output is too long to present here. Notice, during the building of the project a series of lines starting with "EC2: ". These are messages coming directly from the EC2 plugin. Some of this output is from the EC2 API calls which the plugin executes and putting the EC2 prefix in front of its output.

The time it takes to build a project depends on the size of the compressed bundle and your network connection. The upload of the filesystem bundle dominates the building progress. The EC2 plugin will try to estimate the time it takes to upload the bundle and calculate the average speed, but the result is somewhat misleading, since the API upload command spends a considerable amount of time waiting for the upload to actually begin.

Starting/Stopping a project

Starting the project is done using the common MLN syntax. In this example, we have a project ec2 which only contains the virtual machine one:

```
server:~# mln start -p ec2
Starting one in Amazon cloud OK: i-ca365ea3
```

Notice, that the instance ID is printed out along with the message that the virtual machine has started in the cloud. There is no need to write down or remember this ID, as MLN stores it in the project folder.

Stopping the project is done in the same fashion:

```
huldra:~# mln stop -p ec2
Terminating one in Amazon cloud
INSTANCE        i-ca365ea3      running shutting-down
```

Stopping the virtual machine will initiate a graceful shutdown and after a period of time the virtual machine will actually be down.

Checking the status and removing a project

The command mln status -p <project> can be used to check upon the status of a given project. In the case of EC2, the EC2 will print out additional information:

```
server:~# mln status -p ec2
ec2 host one up i-340d655d ec2-174-129-138-128.compute-1.amazonaws.com
```

The instance ID and external DNS address is printed. The last item is useful for conecting to the VM, because (unless you use a elastic IP or dynamic DNS) you have no other way of knowing where to connect.

If you want to check the status directly from the command-line through the EC2 API tools, run the command ec2-describe-instances or its alias ec2din. This will give you the same information which you would also see in the other EC2 tools available, like Firefox's elasticfox plugin. Note, that the virtual machines started by MLN also will show up there, but that they only show you the instance ID. There is no way to currently tag an instance with a better description, like its hostname, although RightScale provides this feature internally on their webpages. In essence, MLN does the same, so the status output links the hostname with the instance ID.

10.7.1 EC2 plugin syntax

Without any keywords, the EC2 plugin will use mostly default values. This translates to an instance of the smallest kind, i386 architecture and no additional volumes or elastic IP addresses. The additional hard-drive will be mounted on /mnt There are many ways in which this can be changed. Here follows a summary of the syntax before the individual settings are discussed:

```
ec2 {
        type <ec2 type>
        region < EU | US >
        zone <availability-zone>
        elastic_ip <IP>
        bucket <bucket>
        arch <arch>
        user_data <user data>
        user_data_override <user data>
        user_file <path>
        credential_folder <path>
        use <host>
        group <security group>
        mount1 <path>
        mount2 <path2>
        volumes {
                persistent_<size> <device> <mountpount> <options>
                <size> <device> <mountpoint> <options>
                <vol-id> <device> <mountpoint> <options>
        }
}
```

Some of these features are not without further explanation and discussion. They are highlighted in the next subsections below. The ones that do not require much explanation are:

- **type**
  You can specify all instance types which are offered by EC2: m1.small, m1.large, m1.xlarge, c1.medium and c1.xlarge.

- **region**
  The region in where the image will be uploaded to and where the instance will be located. Two values are possible, EU and US. Default is US, which can be changed in the ec2.pl plugin.

- **bucket**
  Upload the image to a different bucket other than the default mlnus for the US and mlneu for EU.

- **zone**
  Specify the availability zone. This availability zone has to be located in the region the instance is meant to run in. Also, if you connect the instance to a volume, the zone will be overridden to match the same zone where the volume is located.

- **group**
  Specify which particular EC2 security group to be part of.

- **arch**
  Specify the architecture. Possible values are: i386 and x86_64. Default is to follow the architecture that corresponds to the instance type.

- **mount1 and mount2**
  These can be used to specify where the extra disks will be mounted. In the case of the type m1.small, only one disk is offered. In the other cases, two disks are available.

In the following subsections, we will look at some more important features, which can improve performance and use some of the additional EC2 features, such as elastic IPs and volumes.

10.7.2 Re-using filesystems for faster uploads

MLN creates one filesystem per instance. Building a project with three instances will therefore result in three filesystems being bundled and transferred into EC2. This will increase the time it takes to deploy the project, as well as the bill, because more data is sent over the network. This is also particularly wasteful if the three instances are practically identical except their hostnames.

MLN offers a solution to this, which lets instances point to other instance's machine image. This is accomplished through the use keyword in the ec2 block. One can point to other virtual machines in the same project and will then boot off their filesystem instead of their own. Here is a new version of the Three Musketeers where two of them point to the one. All for one, and one for all, if you will.

```
global {
        project musketeers
}
superclass common {
        xen
        ec2 { }
        network eth0 {
                address dhcp
        }
}
host athos {
        superclass common
}
host porthos {
        superclass common
        ec2 {
                use athos
        }
}
host aramis {
        superclass common
        ec2 {
                use athos
        }
}
```

When this project is built, only the filesystem belonging to athos will actually be uploaded, and the deployment time should therefore be only a third compared to the original project.

Many will at this point wonder if this will not result in confusion, when the project is started and we end up with three instances with the hostname athos? MLN utilizes a feature called user-data which EC2 offers to send a text string to each individual instance at boot time. This string will be different for each of the three instances, containing the string "h=hostname;", for example "h=porthos;" in the case of porthos. Inside the instance is a little shell script which MLN created which will look for this information and change the hostname if it finds this string. This way we will end up with three instances all with different hostnames but from the same image.

Notice that this feature only works if MLN was able to install the script on your Linux distribution in a way that it actually was executed. You can pass other information as well using the user-data functionality. The next subsection discusses this in more detail.

10.7.3 Supplying extra information to each instance through user data

EC2 offers two ways in which you can pass information to individual instances at creation time. This is not the same as when the machine image is bundled and uploaded, but works when a new instance is booted from an image. The typical case when to use this is when there are multiple virtual machines which are practically identical except perhaps their hostname and some additional hints about what the instance is supposed to do. Since network configurations are identical for instances (they use DHCP) there is little left to individualize. Savvy programmers will create images that contain sophisticated software which can grow the instance into any particular role. Tools like cfengine could be used to make a base virtual machine image into a webserver, just by passing it a little hint.

EC2 offers two distinct ways to send data to your instance: One way is the user-data option, which basically adds the string to the command which starts the instance. The other is user-file, which sends the contends of a file as data. Inside the virtual machine, the data can be pulled using a command like:

```
wget --timeout 15 -q -O - http://169.254.169.254/1.0/user-data
```

This command works on all instances regardless of user-file or user-data was used.

MLN offers both ways to append data. The most notable difference is that if you use the following:

```
ec2 {
    user_data this is my string
}
```

Then the text "This is my string" will be sent to the instance every time it boots. However, if you send the contends of a file like this:

```
ec2 {
    user_file /path/to/file
}
```

The the contents of the file will be *read* every time the instance is created. This means, that if the contents of that file have changed, then the information sent to the instance will have changed too. This is therefore a more dynamic solution.

EC2 does not support the use of both user-data and user-file. If MLN encounters both used in the same block, then user_file will override user_data. One important thing to consider, is that if the use keyword is used, the EC2 plugin will actually

send something using user-data. If you have specified some string as well, then the result is the concatenation of those two strings. Consider the following inside the host pathos:

Code	Result in user-data

e2 {
user_data foobar
}
	foobar

e2 {
user_data foobar
use athos
}
	h=pathos;foobar

e2 {
user_data_override foobar
use athos
}
	foobar

e2 {
user_data foobar
user_file /path/to/file
}
	<contents of file>

e2 {
user_file /path/to/file
use athos
}
	<contents of file>

Notice, that using user_data_override or user_file will break the mechanism that ensures individual hostnames when the use keyword is used. Smart sysadmins will ensure individual hostnames on their own accord.

10.7.4 Using elastic IPs

Elastic IPs enables you to have a consistent connection point to your instance. You can request an elastic IP using the different management tools available to EC2. From the command line, you can run the following to get a new IP:

```
ec2-allocate-address
```

To view the addresses you have currently, type:

```
ec2-describe-addresses
```

In MLN, an elastic IP will be assigned to a host at boot time. In the host block, use the following:

```
ec2 {
    elastic_ip X.X.X.X
}
```

10.7.5 Using EBS volumes

EBS volumes provide persistent storage for your instances. Volumes have to be attached to existing instances only, meaning you cannot say "start instance X with volume Y as /var/log" with the existing EC2 tools. This is possible with MLN. MLN supports three different ways in which you can use volumes in your projects:

- **Volumes are created and destroyed along with the project**
  This is the most basic approach. For each line in the volumes block that starts with a size, a new volume is created. MLN stores the volume ID for that volume and adds a line to the start script of the instance which attaches the volume every time the instance is booted. Inside the filesystem, fstab is modified so that the volume is mounted on the correct position. However, a volume comes without any filesystem, so MLN also creates a start-script which will create the filesystem the first time the volume is mounted. This results in a fully automatic process for adding volumes to instances.

  ```
  ec2 {
      volumes {
          2G hda1 /opt ext3 defaults
      }
  }
  ```

  Once the host is removed, these volumes will be deleted with it. This means that you need to fetch eventual valuable information from the volume before the project is removed or rebuilt. It is generally recommended to use hdaX as disk descriptors. The otherwise common sdaX will be used for the main partitions. Most importantly, make sure the VMs template has block-devices in /dev for the disks you want to use.

- **Volumes created along with the project but remain after the vm / project is removed**
  This feature is similar to the one above in that volumes are dynamically created during the build process of the project. The main difference is that once the project is removed, the volumes stay behind and have to be removed manually by yourself.

  ```
  ec2 {
      volumes {
          persistent_2G hda1 /opt ext3 defaults
  ```

```
        }
    }
```

There is no way to tag volumes or add descriptions to them. It is therefore hard, in retrospect, to know which volume belonged to what instance. The only hint is the date the volume was created. Amazon is aware that several users have asked for the possibility to tag or add descriptions to volumes. MLN will support this as soon as it is available. For now, you can take personal notes by looking in the folder called ec2 in the projects folder. There you will find files named after the disk and hostname. The contents of those files are the respective volume IDs.

- **An existing, manually created volume is used**
  Sometimes you already have a volume which you want to connect to a virtual machine. In that case, use the volume ID instead of size on that given line. In the case of a pre-existing volume, MLN will not assume that there is a need to format it. You need to make sure yourself that there actually is a filesystem on that volume, or else it will fail to mount (but it will be connected to the instance, so you can log in and format the volume the first time it is used.) You can create a volume manually on the command line. This command will create a volume in the US in the availability zone us-east-1a with the size of 3 GB:

```
ec2-create-volume --region us-east-1 -z us-east-1a -s 3
```

The output of this command will give you the volume ID, which you can use in the project file:

```
ec2 {
    volumes {
        vol-jd8372hd hda1 /opt ext3 defaults
    }
}
```

If you are unsure about which volumes you currently have, use the command ec2-describe-volumes to get a listing. You can also use the AWS management console, elsticfox or rightscale. In the two latter you can add descriptions to the volumes which are only visible inside the given tool, but helps you keep track.

It is possible to combine these features and to have more than one volume attached to a VM. In the following example, we want a volume of size 2GB mounted on /var/log and also connect a previously created volume to /opt:

```
ec2 {
    volumes {
        vol-jd8372hd hda1 /opt ext3 defaults
        2G hda2 /var/log ext3 defaults
    }
}
```

Things you should know about volumes

The most important thing about volumes, is that they are located in availability zones and can only be attached to instances within the same availability zone. If you assign an existing volume to a VM, MLN will check the location of that volume at build time and deploy the VM in the same zone. If you assign two volumes, each in a different zone, only one of them will end up being connected.
The location of the volume overrides the zone-keyword. So if the three musketeers were all set to be deployed in the zone us-east-1b, but athos would have a volume, which is located in us-east-1c, then athos would be deployed there instead. This can create confusion and sometimes lead to extra costs, since traffic between zones in the same region is charged $0.01 per GB. It is therefore important to know what you are doing when deploying large projects and existing volumes. In the case of transient volumes (i.e those that are created at build time) then MLN will adhere to the zone keyword and create the volume where the instance is supposed to run.

Another important thing about transient volumes is that they are created at build time and unlike instance, you pay for as long as they exist. So if you build a project in the US with ten 4GB volumes, you will be charged $4.0 (10 * 4 * $0.1) every month even if the project is not running, until the project is removed.

Attaching / Detaching volumes manually

You can attach/detach volumes to running instances. The main difficulty is identifying the correct instance first before you attach anything to it. If the instance was created using MLN, you can look up the current identifier of the instance by reading the corresponding file in the projects ec2 directory. So for the porthos instance, we can run:

```
cat /opt/mln/projects/root/musketeers/ec2/porthos.instance
```

Once we know the identifier, we can run the following command to attach a volme:

```
ec2-attach-volume  vol-3411f05d -i i-48b6d121 -d hda2
```

If the volume has never been used before, one needs to add a filesystem to it. This is done from the instance. In order to detach the volume, one can run:

```
ec2-detach-volume  vol-3411f05d -i i-48b6d121 -d hda2
```

10.7.6 Using different EC2 accounts for different projects / instances

For the most part, this documentation has been assuming that one EC2 user will be using MLN to manage its instances. It is, however, possible that one sysadmin or provider might manage several projects for different customers. In that case, MLN offers a way to individualize the ownership of virtual machines to different EC2 users. The only thing necessary is to have a folder with the EC2 users credentials in it in the same way as the .amazon folder we created above.

In the following example, we create a project consisting of two webservers and a database server for a customer. The credentials of the customer are located in the folder /customers/customer1. Notice how volumes are used in order to provide permanent storage. All instances will get a 2GB volume to store their logfiles in. The webservers each have a small partition at /var/www while the database has a 10GB partition for the database files. In order to speed up deployment, www2 will use www1's filesystem. It is important to realize, that since www2 will be a copy of www1, they need to have the same volumes block too, or else they will be expecting partitions which are not present. Even though www2 is a copy, it will still have its individual volumes connected to it.

The complexity in this project is a good example of how MLN can ease the deployment process. Keeping track of three instances and manually (and correctly) connecting six volumes each time the instances are started will already be a complicated task.

```
global {
    project customer1
}

superclass common {
    ec2 {
        credential_folder /customers/customer1
        volumes {
            2G hda1 /var/log ext3 defaults,noatime
        }
    }
    xen
    network eth0 {
        address dhcp
    }
}
```

```
host web1 {
        superclass common
        template webserver.ext3
        ec2 {
                volumes {
                        1G hda2 /var/www ext3 defaults
                }
        }

}

host web2 {
        superclass common
        template webserver.ext3
        ec2 {
                volumes {
                        1G hda2 /var/www ext3 defaults
                }
                use web1
        }
}

host db {
        superclass common
        template database.ext3
        ec2 {
                type c1.medium
                volumes {
                        10G hda2 /var/lib/mysql ext3 defaults,noatime
                }
        }
}
```

One important aspect, is that MLN will run all its commands relative to that project as customer1, but when you later run commands on the command-line, you will be the original EC2 user. This means that you cannot expect to see customer1's instances just by running ec2din. Managing these virtual machines should therefore be done only through MLN's commands to avoid confusion.

10.7.7 Adding more nodes to existing project

One attractive aspect of cloud computing is the ability to scale your infrastructure based on the current load. This would then directly connect usage to cost and keep your infrastructure cheap when there is low activity. Technically, the main idea is to have some sort of load balancing and an array of backend servers which you increase/decrease based on your demands. Load balancing can be achieved in several ways, depending on the service you run. Let's assume that customer1 from the previous section has DNS loadbalancing which is not explicitly part of the MLN project. However, when the load increases, they would like to increase the number of webservers. In MLN this can be achieved with the mln upgrade command. The only thing necessary is to make an updated version of the customer1.mln file with an additional webserver:

```
global {
	project customer1
}

superclass common {
	ec2 {
		credential_folder /customers/customer1
		volumes {
			2G hda1 /var/log ext3 defaults,noatime
		}
	}
	xen
	network eth0 {
		address dhcp
	}
}

host web1 {
	superclass common
	template webserver.ext3
	ec2 {
		volumes {
			1G hda2 /var/www ext3 defaults
		}
	}

}

host web2 {
	superclass common
	template webserver.ext3
	ec2 {
		volumes {
			1G hda2 /var/www ext3 defaults
		}
		use web1
	}
}

host web3 {
	superclass common
	template webserver.ext3
	ec2 {
		volumes {
			1G hda2 /var/www ext3 defaults
		}
		use web1
	}
}

host db {
	superclass common
	template database.ext3
	ec2 {
		type c1.medium
		volumes {
			10G hda2 /var/lib/mysql ext3 defaults,noatime
		}
	}
```

```
}
```

One could add more than one virtual machine, off course. Notice, that the new virtual machine, web3 also uses web1's filesystem. This means that deploying this additional instance will be quite fast, as no new filesystem needs to be uploaded into EC2. In order to do the actual upgrade, type:

```
mln upgrade -S -f customer1-2.mln
```

The option -S will ensure that new virtual machines are started right after the upgrade, so the effect of this command is that a new virtual machine is actually running as part of the project.

Scaling DOWN your cluster is just as important as scaling it up. You basically have two alternatives:

1. Keep the virtual machines in the project, but shut them down instead of removing them.
2. Remove the redundant virtual machines from the project entirely

The first solution is easily accomplished with MLN's start and stop commands. Say that customer1 wants to shut down the extra webserver they just deployed. Then the following is sufficient:

```
mln stop -p customer1 -h web3
```

Now, if the pressure increases, the instance can be booted up just as easy:

```
mln start -p customer1 -h web3
```

The benefit of this approach is speed and that the persistent storage will keep logfiles and everything intact between boots. The drawback is that you keep paying for the volumes used by web3 even when it is not running. This basically boils down to the amount of persistent storage and the periods these extra virtual machines will be unused.

In order to upgrade to a version of the project with less virtual machines (Alternative 2), supply a new version of the project file with the virtual machines removed. This usually means supply the original or a previous project file:

```
mln upgrade -S -f customer1.mln
```

10.7.8 Moving an existing project / vm into EC2 and back again

One of the most exiting features of MLN combined with EC2, is that you can move virtual machines into EC2 from your own server. This means you can have a running project where you log into the virtual machines and configure them just right before you then push the whole thing into EC2. This enables you to transition between a weak local setup, where only the configuration is important, into a powerful cluster when you need to do some production work.

The way to do this, is to utilize the upgrade command in MLN. By adding the ec2 block to the hosts which are to run in the cloud. It is possible with projects where only some of the virtual machines run in the cloud and some locally.

In this example, host nomad had the following configuration in the first version of the project file:

```
host nomad {
        xen
        users {
                joe
        }
        network eth0 {
                address 10.0.0.3
                netmask 255.255.255.0
                gateway 10.0.0.1
        }
        template etch.ext3
        free_space 4000M
}
```

After editing the MLN-file for that project, nomad now has the following configuration:

```
host nomad {
        ec2 {
        }
        xen
        users {
                joe
        }
        network eth0 {
                address dhcp
        }
        template etch.ext3
        free_space 4000M
}
```

Notice, that the network configuration was changed too. This is very important. Without DHCP, the virtual machine will not get online inside EC2 and you need

to make that change yourself for now. By simply adding the empty block `ec2 { }` we have made this into a cloud instance with all settings picked from default values. Next, you need to effectuate the change by running the MLN upgrade command:

```
mln bupgrade [-S] -f new_version.mln
```

By adding the -S option, the virtual machines will be automatically started after the upgrade.

Common features in MLN which no longer work in EC2

Some parts of the configuration for a virtual machine will no longer have any effect once the virtual machine is transfered to the EC2 cloud. Most important is memory and network interfaces, since this is predetermined by EC2. All the features which address the filesystem itself, such as users, passwords, groups and startup commands should work like before, except the network configuration. Since EC2 only supports DHCP, you have to change your configuration of the `eth0` interface to DHCP or else the instance will not get online.

10.8 Known Issues

These are the main issues you should be aware of:

Filesystems are created even if "use" keyword is used

MLN will always create a filesystem from the template and configure it even though it is not going to be uploaded. This will slow down a scaling process due to the time it takes to make a copy of the image.
The reasoning behind this is that should the project be migrated back from EC2, it would need a filesystem locally. Future versions of this plugin may have a feature to explicitly not create any filesystem.

The eth0 interface has to be set to dhcp by you

MLN will not check if the eth0 interface is actually dhcp. This should be easily accomplished with an additional plugin or inside the EC2 plugin. Right now, however, you need to make sure of it yourself.

Writing Plugins

Chapter 11

Writing MLN plugins

MLN offers support for extra plugins that can configure the virtual machines in new ways i.e adding apache-conf specifications directly into the project file. They can also support new virtualization technologies, like VMware or KVM. A plugin may also function as a wrapper for a special kind of storage, like iSCSI. Writing a plugin is supposed to be easy for the user. This chapter shows how to create plugins and how to work with the MLN data structure for projects. The same data structure can be used to debug you plugins before you actually build the project.

Plugins allow you to basically insert code into strategic places during MLN's execution, without diving into the MLN code itself. Based on the way MLN includes the plugins, only plugins in the PERL programming language are supported.[1]

The entry points for where your code can be executed are called plugin hooks'. Which hooks you want to use depends on what you want to achieve. You will seldomely have to make use of all the possible hooks. This chapter will try to explain the signifficance of each hook and show examples on how they are used.

11.1 The MLN data structure

Before we go into the details of the hooks, we need to learn about the internal data structure used by MLN to store information about a project. This data structure is important to allmost all plugins in some way or another.

The data for the entire project is stored in a tree-structure. One can get/set values from that structure inside a plugin. When mln parses a project file it will build the tree structure automatically even with new keywords and blocks, so there is no need to write parsing code in the plugin.

[1] It should not be too difficult to write a perl wrapper for your favourite programming language.

The data is addressed by its path in the directory structure. At the root is the name of the project followed by three mandatory blocks: hosts, superclasses and global. Every block contains a set of regular lines and a set of subblocks. The data can be accessed in different ways depending on what you want. This is a very liberal approach. It means that the plugin can place data where it wants and in the style it wants as long as it does not break the semantics of the tree structure. Lets see an example:

```
global {
  project myparse
}

superclass hosts {
  memory 128M
}

host one {
  superclass hosts
  network eth0 {
    address dhcp
  }
}
```

If you want to see the resulting raw tree structure from a project file, you can use the mln parse command:

```
mln parse -f myparse.mln
global {
        project myparse
}
switch {
}
superclass {
        hosts {
                memory 128M
        }
}
host {
        one {
                superclass hosts
                network {
                        eth0 {
                                address dhcp
                        }
                }
        }
}
```

Notice, how the host and and network blocks are divided into a subdirectories. This is the only major difference between the project textfile and the data tree.

11.1.1 Querying the data tree

More specific queries can be run in case you are interested in certain aspects of the data. The same mechanism is available as an API and can be used inside a plugin to access the data. One has three quesry mechanisms depending on the data you are interested in: a single Scalar value, an Array block or a Hash subtree. These queries can be triggered by the -S, -A} and \texttt{-H arguments followed by a query path. The result of the queries takes variables and inheritance in account, so what you get as a result is the same that MLN will get internally, as will your plugins.

Examples

```
mln parse -f myparse.mln -S /host/one/network/eth0/address
dhcp
```

This query will get the scalar value corresponding to the keyword "address" at "/host/one/network/eth0". It can also be used to check if a block is present or not:

```
kyrre@fs~$ mln parse -f myparse.mln -S /host/one/network/eth0
1
```

If you want to get all the lines in a block, use the -A query type:

```
kyrre@fs~$ mln parse -f myparse.mln -A /host/one
memory 128M
superclass hosts
```

The "-A" query will give you all the lines belonging to a block, but no sub-blocks. This is usefull if you have a block with a list of lines in it that are not keyword/value pairs, like a list of modules to load or a list of commands to run.

If you are more interested in an entire subtree, you can use the -H query to get it:

```
kyrre@fs~$ mln parse -f myparse.mln -H /host/one
network {
        eth0 {
                address dhcp
        }
}
memory 128M
superclass hosts
```

The entire tree can be displayed with this query:

```
kyrre@fs~$ mln parse -f myparse.mln -H /
global {
      project myparse
}
switch {
}
superclass {
      hosts {
            memory 128M
      }
}
host {
      one {
            network {
                  eth0 {
                        address dhcp
                  }
            }
            memory 128M
            superclass hosts
      }
}
```

11.2 Writing plugins

MLN will load plugins from the following locations: /etc/mln/plugins}, \texttt{\$HOME/.mln_plugins. The name of the plugin is the same as the filename. A local plugin will override a global defined one. A plugin needs to be written in the PERL programming language. MLN will at certain points in the build process look for a method in the plugin with a corresponing name on it. If the method is there it will be executed. So a plugin is basically a file with a set of PERL subroutines. One can choose which one to define, so it opens for very simple and small plugins.

11.2.1 Example plugin and Available subroutines

This is an example of a plugin placed at /etc/mln/plugins/myplugin.pl. The plugin will look for a "myplugin" block in the host and write all the lines it finds there into a file called /var/myplugin.dat on the virtual machine.

The project used to test this plugin is plugintest.mln:

```
global {
      project plugintest
}

host nemo {
    myplugin {
          line1
          line2
          line3
```

```
    }
}
```

The MLN parser will read everything into the tree structure even if it is not a known keyword or block. So it should be possible to see the plugin with a query without adding parsing information:

```
mln parse -f plugintest.mln -A /host/nemo/myplugin
line1
line2
line3
```

The output shows that the parser has added myplugin to the datastructure and we can proceede with the actual plugin. We start out with the mandatory routine, which is required to write the version number and perhaps some info about the plugin.

```
# MyPlugin

sub myplugin_version {
    print("myplugin version 1");
}

1;
```

Notice the "1;" at the end of the plugin. This is required for all plugins and it will fail without it. Next one only has to write subroutines that we want to use.

The next subroutine is written for the user. Before MLN starts to build the project, it writes out data about the machines to be buildt. In addition it looks for a method called plugin_printHost and calls it for each host it finds. For this plugin, the method could look like this:

```
sub myplugin_printHost {

my $hostname = $_[0];

# first we check if this host has a myplugin block:
my @myplugin_lines = getArray("/host/$hostname/myplugin");
  if ( @myplugin_lines ){
    out("myplugin is enabled on this host:\n");
    my $line;
    foreach $line (@myplugin_lines){
        out("$line\n");
    }
  }
}
```

Notice the use of the getArray method to get the data we are interested in. The same queries can be issued from inside the plugin, returing either a scalar, and array or a hash. The result can be seen if we run mln build with the simulation flag on:

```
mln build -s -f plugintest.mln

++ printing Superclasses ++

++ printing Hosts ++
--> Host nemo
nemo: template = Debian-3.0r0-V1.0.ext2
myplugin is enabled on this host:
line1
line2
line3
++ printing global settings ++
Project:       plugintest
Default uml kernel:  /opt/mln/uml/uml-2.6.12-rc2-mm3/linux ()
Default module path: /opt/mln/uml/uml-2.6.12-rc2-mm3/

Parsing complete
```

The last method we need is the one that does the acctual writing to file. MLN has already some functions for file modification, so we can use them. Once configuration process for each filesystem is done, MLN looks if any plugins have a plugin_configure method and calls it. This is where we put our code too:

```
sub myplugin_configure {

    my $hostname = $_[0];
    # first we check if this host has a myplugin block:
    my @myplugin_lines = getArray("/host/$hostname/myplugin");
      if ( @myplugin_lines ){
          out("Writing myplugin lines to /var/myplugin.dat\n");
          writeToFile($hostname,"/var/myplugin.dat",\@myplugin_lines,"600");
      }
}
```

This is basically it for a simple plugin. Let us just have a quick glance at modification of the data structure. Lets say, that for the writer of this plugin it is important to have more than the default ammount of 32M or RAM. The way to solve this, is to check the ammount of ram present and if it is too low, then it is increased. The best place to do this is right after the project file has been parsed. The method name for this is plugin_postParse. This method is only called once, so we need to check all the hosts from that single method. Here is how it could look:

```
sub myplugin_postParse {
    # we need to check all the hosts in the same method.
    # set memory to be at least 64M

    # we use 'getHosts()' instead of 'keys getHash("/host")'
```

```
    # because it has better performance and we
    # dont need the entire hash anyway
    my $host;
    foreach $host ( getHosts("/host") ){
        # we check for the presence of a myplugin block
        if ( getScalar("/host/$host/myplugin") ) {
            my $memory = getScalar("/host/$host/memory");
            # we strip the 'M' from the value:
            $memory =~ s/(\d+)M/$1/;
            if ( not $memory or $memory < 64 ){
                setScalar("/host/$host/memory","64M");
            }
        }
    }
}
```

We use the setScalar() call to modify a single keyword/value pair in the data structure. Similar methods exist for hash and arrays. Since we make the modification right in the beginning, we are able to check the result without actually building. We issue a query instead:

```
kyrre@fs:~$ mln parse -f plugintest.mln -H /host
nemo {
        myplugin {
                line3 1
                line1 1
                line2 1
        }
        memory 64M
}
```

We see, that the amount of memory has been added to the host nemo. Note, that the lines inside the myplugin block are in a different order and that they are followed by a "1". This is typical for when one gets the hash. The array query should be used directly on the block that matters if order is important.

This summarizes the current available methods for a plugin that MLN will call. If there should be a need to have even more such methods, please let the MLN developers know of it and it will be added.

11.3 Plugin API suppary

11.3.1 Reading The Data Tree

You have three different query options:

```
my $memory = getScalar("/myproject/host/memory");
```

Returns the string corresponding to the value belonging to the keyword memory. This method is reccomended if you have keywords that you know and want to see if they are defined. Examples are: "/myproject/host/jumbo/netowork/eth0/address"

```
my @modules = getArray("/myproject/host/jumbo/modules");
```

This one is better for blocks that will not contain any keyword/value pairs, but rather a list of lines that you don't know what will be. This is usefull for things like the list of modules, startup commands or users. Note that getArray will ignore sub-blocks.

```
my %keywords = getHash("/myproject/host/jumbo/network");
```

Here, MLN will create a nested hash of the subtree from that address on. Both lines and blocks are present. One needs knowledge of what to expect in order to traverse the hash afterwars. The first word of every line is always considered to be the keyword and is the key in the hash.

11.4 Modifying The Data Tree

You can modify the contents of the data tree almost the same way as you get data. You can insert a single scalar, array or hash at a given position in the tree. The typical case is to change some single keyword/vaules at some positions:

```
setScalar("/host/jumbo/network/eth0/address","dhcp");
```

If the entire path does not exist, then mln will create it. Chanegeing an array can be done the same way, but one has to remember, that setArray will overwrite the array at that position, so if you want to simply add or modify a line, you will have to get the array first, modify it, and then put it back again:

```
my @array = getArray("/host/nemo/myplugin");
$array[1] = "line2 was modified";
push(@array,"last line");
setArray("/host/nemo/myplugin",\@array);
```

The same can be done with an getHash/setHash. It is recommended that the modifications to the data tree are made in the _postParse plugin method so that the changes are visible to mln as quicly as possible. Also, that way it is easy to check for the desired results using mln's command line queries.

11.4.1 Tips for the plugin writer

1. If you make a specialized plugin, make sure it is not triggered for all hosts, but that it checks fore some hint in the data if it is enabled, just like the example above.

2. If your plugin modifes a confguration file, make sure the changes are kept consistent and convergent through serveral runs of the plugin. This means that in the case of an upgrade, the plugin will run on a filesystem it ran on previously, so the changes should not end in double lines but should alwasy try to produce the same result. Check how your plugin behaves in an upgrade.

3. Use the `mln parse -f file.mln` with additional queries to check the result.