The MLN Manual
mln version 0.82b15

Kyrre M Begnum, John Sechrest

August 31, 2006

Contents

Introduction

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

- Support for Xen 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 1. However, this document will also provide in-depth explanations of the motivation for MLN as well as its features and advantages.

0.1 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
     }
}
```

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. Preinstalled 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-Vm.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 `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 `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 :=, 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 Blocks

5.2.2 `global`

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
     }
 }
```

`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.

5.2.3 `switch`

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 `host`

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

`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 boottime 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.

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 `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:

Block Type	function
global	
switch	
host	
superclass	A special class

Table 5.1: The different 1. level blocks.

```
    users {
        jack lkjlkadlfasd
    }

    groups {
          admin {
               jack
          }
    }
```

5.4 summary

5.4.1 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`.

5.4.2 Keywords

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 uninstalled 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
```

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
\begin{verbatim}

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

\begin{verbatim}
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

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.

Chapter 7

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.

7.0.2 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.