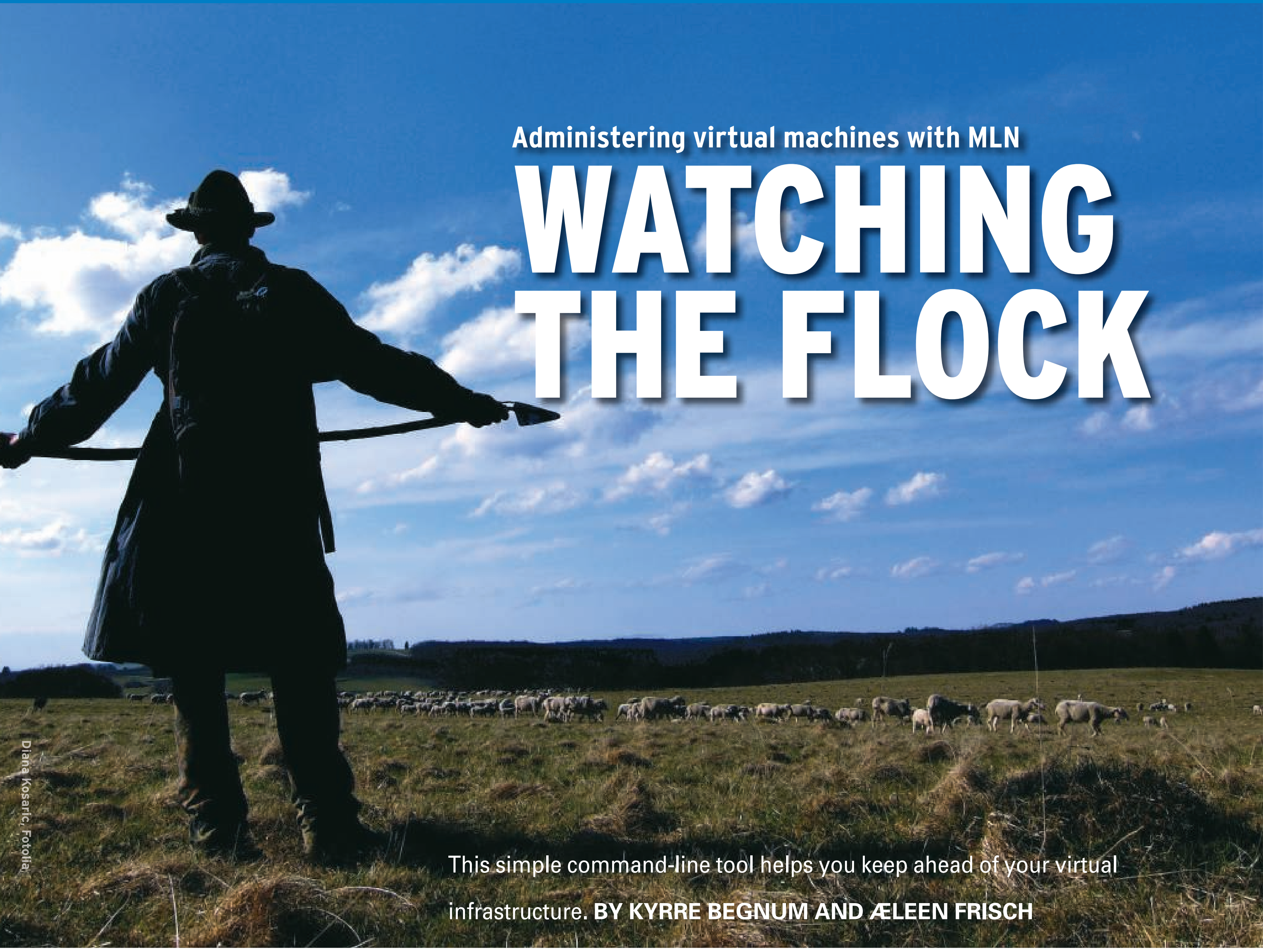

Administering virtual machines with MLN

WATCHING THE FLOCK

This simple command-line tool helps you keep ahead of your virtual infrastructure. **BY KYRRE BEGNUM AND ÆLEEN FRISCH**

The simplest virtualization scenarios are very easy to manage. In the real world, though, virtualization does not always simplify system administration. In fact, many of the most powerful uses of virtualization result in additional administrative tasks and challenges. For example, large data centers use virtualization to consolidate several servers onto the same physical hardware, thereby minimizing space, cost, and power requirements.

These virtual machines (VMs) can migrate between servers to achieve higher uptimes and flexible load balancing. However, managing such a complex virtual infrastructure is at least as challenging as administering the equivalent number of separate server systems, and it requires very different skills.

Many high-end commercial virtualization products come with sophisticated VM management features. If you are wondering whether the free software community offers an equivalent application, the open source MLN VM administration tool might be just what you need. MLN, which stands for "Manage Large Networks," lets you build sophisticated, dynamic, virtual infrastructures with the use of freely available virtualization platforms. The versatile MLN currently supports the widely used VMware Server [1] and Xen [2] packages, as well as User-Mode Linux [3].

For Xen, MLN also provides live migration of entire network topologies, which is otherwise available in applications only at the highest end. MLN's simple command-line and config file interfaces serve the needs of complex virtualization scenarios, but it is easy enough for administrators who are new to virtualization.

About MLN

The MLN environment helps you manage large numbers of virtual machines through a simple command-line interface. MLN has three main design goals:

- The user should be able to design and implement complex VM deployments.
- The process from design to actual deployment should be as efficient as possible.

The configuration language should be extensible, so users can customize MLN easily for their unique needs.

MLN is designed to work in concert with other VM management tools. For example, if you have a VMware Server, you can use MLN for efficient building and redeploying of complex virtual networks and also use VMware Server Console for starting and stopping individual virtual machines.

Download MLN from SourceForge [4]. The installation process is simple and similar to the procedure for other Linux programs:

```
# wget ↩
http://mln.sourceforge.net/↩
files/mln-latest.tar.gz
# tar xzfv mln-latest.tar.gz
# cd mln-latest
# ./mln setup
```

Defining Projects

MLN organizes virtual machines into units called *projects*. Projects are groups of virtual machines and their associated virtual networks. The simplest project consists of a single virtual machine. A more powerful approach is to group virtual machines that belong together: all virtual machines that are web servers, an entire replica of a small business network, all the test systems for a software test suite.

Projects are defined in MLN configuration files. The configuration file shown in Listing 1 (*ldaptest.mln*) defines two virtual machines connected by a virtual LAN. Annotations in the file highlight the most important configuration settings.

Listing 1 starts with the *global* stanza, wherein the name of the project is defined. Next follows a declaration of a virtual machine called *ldapclient* – a Xen host with 128MB of memory built from

Listing 1: ldaptest.mln

```
global {
 (Settings affecting all VMs and
  switches in the file.)
   project ldaptest     Project name.
}

host ldapclient {
   xen              This is a Xen VM.
   memory 128M   Memory size for VM.
   lvm              Create logical volume
                     for storing VM.
   size 2GB      Size of the VM.
   template ubuntu-client.ext3
   nameserver 10.0.0.15  Set the
                              name server.
   network eth0 {      Config. virt.
                        net.interface.
     address dhcp      Use DHCP for
                        IP address.
     switch LDAPlan    Connect to
                         virt. switch.
   }
   users { kyrre 147/Y.NtB9p7w }
   (Create user and set encoded
    password.)

host ldapserver {
   xen              Another Xen VM.
   memory 256M
   lvm
   size 2GB
   template ubuntu-ldap.ext3
   (Initial OS image.)
   nameserver 10.0.0.15
   network eth0 {     This VM uses a
                       static address.
     address 10.0.0.2
     netmask 255.255.255.0
     gateway 10.0.0.1
     switch LDAPlan  Connect to
                      virtual LAN.
   }
}

switch LDAPlan { }
 (Create virtual switch connecting
  the VMs.)
```

1 2 3 4 5 6

HPC Your Way

Intel or AMD. Ethernet or InfiniBand. Linux or Microsoft Windows HPC Server. Now you can have a uniform set of HPC compilers and tools across all of your x64 clusters. PGI CDK compilers and tools are available directly from most cluster suppliers. Take a free test drive today at www.pgroup.com/reasons

PGI CDK® Cluster Development Kit®

Table 1: Working with the mln Command

Action	Command
Build VM	*mln build -f project-config-file [-r]*
Reconfigure VM (including live migration)	*mln upgrade -f modified-config-file*
Start VM	*mln start -p project [-h host]*
Stop VM	*mln stop -p project [-h host]*
Query status of all/specified projects	*mln status [-p project]*
List defined projects	*mln list*
Remove a project	*mln remove -p project*
Register a template	*mln rt -t file-system-image-file*
Backup an entire project	*mln export -p project -d location [-z]*
Restore a saved project	*mln import -p project -d location*

the template *ubuntu-client.ext3*. The host has one network interface configured to use DHCP for its IP address. The third stanza defines another Xen host, *ldapserver*, with similar settings, although it is given more memory. Note that the second host uses a different template as its initial operating system image and has a static IP address. The final stanza defines a virtual *switch* to which both virtual machines are connected; this switch allows them to communicate with one another.

Most of the settings are straightforward, but the template specifications deserve special mention. Templates are pre-fabricated VM filesystems. Each time you build a new virtual machine, you specify a template to use as its base. The virtual machine will be individualized by MLN during the build process. Some people like to use very generic templates for all their virtual machines and customize them later, whereas others create very specialized templates, like a fully functional LAMP server or a copy of a typical employee desktop.

The definition for a VMware Server virtual machine differs only slightly from the host definition in Listing 1. The *vmware* keyword is used in the VM *host* definition stanzas instead of *xen*, and a VMware format template is used instead of a filesystem image. The *switch* definition stanza also requires the *vmware* keyword.

Valid *host* and *switch* names are case sensitive and can contain lowercase and uppercase letters, numbers, and hyphens.

Managing VMs with the mln Command

Once you have defined a project in a configuration file, you can use the *mln* command to build the virtual machines and virtual network inside it. For example, the following commands create the virtual hosts and network defined in Listing 1 and then start the hosts and virtual network:

```
# mln build -f ldaptest.mln
# mln start -p ldaptest
```

The *build* subcommand instantiates the entities in the specified project file, and the *start* subcommand boots the virtual machines and activates the virtual network.

Typically, the *build* subcommand is used only once – initially to create the project – whereas the *start* subcommand and its sibling *stop* are used to activate and deactivate the project hosts and network.

By including the *-h* option, you can apply the command to a single host within a project:

```
# mln stop -p ldaptest -h
ldapserver
```

Table 1 summarizes the most important subcommands related to project management.

Clearly, writing *host* stanzas for large numbers of virtual machines would get tedious. The MLN configuration language allows you to define superclasses: named groups of settings that can be applied to multiple hosts.

Superclass

Listing 2 illustrates the use of a superclass. The superclass, which is called *basic* in Listing 2, provides a bundle of settings that are then available for *host* definitions. Listing 2 defines three hosts based on the superclass. The third host illustrates the technique of overriding a setting from the superclass. Of course, many more settings could be placed into the *superclass* or individual *host* stanzas as required. One project file can include multiple superclasses, and the superclasses can even be nested.

Substanzas, such as network interface definitions, which appear both within *superclass* and *host* definitions, are merged. Directives that appear in the *host* definition have precedence over directives within the *superclass*.

Distributed Projects and Live Migration

So far, all the examples have been located on a single server. However, MLN supports distributed projects in which virtual machines are located on one or more remote hosts. The *host* and *switch* definitions in a distributed project require an additional *service_host* directive, which specifies the server location:

```
host rem-vm {
   vmware
   template rhel5.vmdk
   ...
   service_host bigvmbox
}
```

Listing 2: Working with a Superclass

```
global { project italy }

superclass basic {
   vmware
   template rhel5.vmdk
   memory 128M
   network eth0 {
      address dhcp
      switch lan
   }
   service_host milano
}

host uno { superclass basic }
host due { superclass basic }
host tre {
   superclass basic
   memory 256MB
}

switch lan { vmware }
```

Superclasses can include the *service_host* directive as well (as shown in Listing 2).

For servers that will host virtual machines, the MLN daemon must be running. In addition, the MLN daemon's configuration file, */etc/mln/mln.conf*, must define the server to be referenced in project files and must grant access to remote systems from which projects will be managed. For example, the MLN daemon configuration file on the target computer for the *rem-vm* virtual machine would contain directives like the following:

```
service_host bigvmbox
daemon_allow 192.168.10.*
```

In this case, the *daemon_allow* setting permits access only from systems on the local subnet.

The argument to *service_host* can be either a resolvable hostname or an IP address.

With MLN, it is a very small step from distributed projects to live VM migration in Xen environments.

To move virtual machines from one server to another, all you need do is modify the *service_host* settings within the project file to reflect the desired new destination and then rebuild the project with the *mln upgrade* command. For example, the following commands modify and rebuild the *italy* project shown in Listing 2, resulting in the virtual machines migrating from the server denoted by *bigvmbox* to the one denoted *newvmbox*:

```
# vi italy.mln
Change bigvmbox to newvmbox
# mln upgrade -f italy.mnl
```

Note that this command can run even when the virtual machines within the project are currently running.

Listing 3: MLN with a SAN

```
host odysseus {
     xen
     lvm
     lvm_vg volume-group-name
     service_host scylla
     ...
   }

host ulysses {
     xen
     filepath /nfs-mount-point
     service_host charybdis
     ...
}
```

```
host odysseus {
     xen
     lvm
     lvm_vg volume-group-name
     service_host scylla
     ...
   }

host ulysses {
     xen
     filepath /nfs-mount-point
     service_host charybdis
     ...
}
```

Energy Efficient PCs

30 watts average

ITX	
Processor	Intel Atom
Memory	Up to 2GB of DDR2
Graphics	Intel 945
Hard disks	SATA hard disks with low power consumption
OS	Various Linux distributions available
Power	30 watts average
Price	**From £225**

40 watts average

Deskcube D500	
Processor	AMD Athlon X2 or Phenom X4
Memory	Up to 8GB of dual channel DDR2
Graphics	NVIDIA GeForce 8200
Hard disks	SATA hard disks with low power consumption
OS	Various Linux distributions available
Power	40 watts average
Price	**From £317**

Digital Networks United Kingdom

www.dnuk.com sales@dnuk.com 0161 366 6600

Prices as of 24th October 2008.
Goto www.dnuk.com for full details.

Listing 4: Use of the #include Statement

```
global {
   $gnum = 03
   $userpasswd = unique-value-1
   $rpasswd = unique-value-2
   $vncpasswd = unique-value-3
   project = os$[gnum]
}

#include oscourse.mln
```

Live migration also requires a few preparatory steps:

- All of the servers must run the MLN daemon. Access control for the daemon must grant access to the other servers involved in the migration.
- The virtual machines must use shared storage to hold images that will be accessible to all relevant servers. The best solution is usually a storage area network (SAN) controlled by the logical volume manager (LVM), but you can also use a shared NFS directory for testing, especially when performance is not a consideration. The location is specified in the *host* definitions in the project (see Listing 3).

In addition, with a SAN, the */etc/mln/mln.conf* file on all relevant servers must contain the *san_path* directive. The argument for *san_path* specifies the location of the SAN, which is either the volume group name (like *lvm_vg* within the *host* definition) or the local mount point for the SAN.

Xen must be configured to allow migration on all relevant servers via the following entries in the Xen daemon's configuration file */etc/xen/xend-config.sxp*:

```
 (xend-address '')
 (xend-relocation-hosts-allow ↩
'^localhost$ ^.↩
*\\.ahania\\.com$')
```

The argument in the second directive is a single-quoted list of regular expressions specifying allowed hosts. In this case, we specify the local host and hosts anywhere within the *ahania.com* domain.

One Is Easy, and So Are 60

Consider the case of an operating system class taught to, say, 120 students. The students are organized in groups of two for lab work, with each group using a network of one Linux and one Windows virtual machine to solve the course exercises. The Linux virtual machine will have two network cards and share its connection to the LAN with the Windows virtual machine.

The challenge for the instructor is to create 60 of those mini-networks quickly, each with an individual public IP address and password.

The main philosophy for this task is "design once, deploy often." All 60 mini-networks must be as consistent as possible, and the system also must be easy to reconfigure, so if you later decide that the Windows virtual machines need more memory, you can modify all of them easily. Putting all of the virtual machines into a single project might seem like an easy way to accomplish this, but a single project would make it more difficult to manage one group's virtual machines separately.

A better solution is to use one project per student group. However, it will not be necessary to write 60 complete project definitions. The *#include* statement lets you compress as much information as possible, so each project file only contains the information unique to that project. For example, you can create 60 small project files and let them all point to the same main configuration file (see Listing 4).

The items beginning with a dollar sign are variable definitions, which will be used within the main configuration file.

Listing 5 shows the main file, *oscourse.mln*. (Note that this file has no *global* stanza.)

The *hvm* directive in the definition of the virtual machine running Windows XP indicates that it uses Xen full (hardware) virtualization. The Linux virtual machine uses Xen paravirtualization. As such, networking configuration is internal to this virtual machine (as IP address 10.0.0.2), so it is not configured by MLN and there is no *network* substanza. This configuration file also introduces the *root_passwd* keyword, as well as two keywords that set up the VNC display.

With a shell or Perl script, you can loop over the various project files with the *mln create*, the *mln update*, or both commands if necessary, as in when a change occurs to a customized template file used by the Linux virtual machine. Also, you can run *mln* commands manually.

Defining Virtual Clusters

Virtualization is also an effective way of using a cluster of physical machines. A cluster can essentially act as a server farm for virtualization; in fact, the student networks described previously could run on the various nodes in a cluster. The physical cluster can serve as physical hardware for one or more virtual clusters. Listing 6 shows a project file that automatically creates a virtual cluster of 36 nodes.

This project uses the *autoenum* plugin (included with MLN) to automatically

Listing 5: oscourse.min

```
superclass common {
   xen
   lvm
   memory 256M
   network eth0 {
      switch lan
      netmask 255.255.255.0
   }
   users {
      osuser$[gnum] $userpasswd
   }
   root_passwd $rpasswd
}

switch lan { }

host ubuntu {
   superclass common
   template os_linux_1.ext3
   memory 128M
   network eth0 {
      address 10.0.0.1
   }
   network eth1 {
      address dhcp
   }
}

host win {
   superclass common
   hvm
   template winXP.template
   vncpasswd $vncpasswd
   vncdisplay 300
}
```

Anzeige
kommt
separat

Listing 6: Creating a Virtual Cluster

```
global {
  project bioinfo
  autoenum {  Plugin to auto assign
              IP and service host.
    superclass cluster-node
    numhosts 36  Number of VMs to
                 create.
    address auto    Assign IP
                    addresses ...
    addresses_begin 150
  (starting with this host number ...)
    net 128.39.73.0    ...on this
                          subnet.
    service_hosts {  Service hosts
                     to use for
                     VMs ...
      #include /bio/servers.txt
     (...stored in an external file.)
    }
  }
  $gateway_ip = 192.168.33.211
}

superclass cluster-node {
  template ubuntu_mpi.ext3
  memory 312M
  free_space 1G
  (Required free space in the
   storage filesystem.)
  network eth0 {
    gateway $gateway_ip
    (Default gateway setting.)
  }
}
```

Listing 7: Plugins

```
global {
  project webserv
  apache {
    doc_root /var/www
    }
  sshkey {
    nagios /home/nagios/.ssh/
pubkey.outgoing
  }
}

host web14 {
  ...
  apache {
    max_connections 300
  }
}
```

generate virtual machines with successive IP addresses and service hosts taken from the list specified in the plugin's *service_hosts* substanza.

Listing 6 creates 36 hosts on the basis of the settings in the *cluster-node* superclass, and the list of service hosts comes from an external file referenced with the *#include* directive.

The *superclass* definition also introduces two more configuration file entries that specify the required amount of free space within the filesystem in which the VM image will be stored (another way of specifying its size) and the default gateway (router) IP for a network interface.

Plugins for Special-Purpose Configuration Options

The preceding example illustrated the use of a plugin designed for a specific configuration task: in this case, assigning sequential IP addresses and service hosts.

The MLN configuration language has an open architecture that simplifies the task of writing plugins. MLN plugins are Perl routines that are invoked within the *global* stanza of a project file.

Listing 7 shows two more examples of plugins. One (*apache*) sets various configuration parameters for the Apache web server within a virtual machine. Another (*sshkey*) installs the specified SSH key file(s) into the designated user account. This technique is very practical because it allows you to build virtual machines that can be automatically accessed from other systems, such as by a monitoring tool like Nagios.

The *apache* plugin will cause MLN to set the Apache document root location and the maximum number of simultaneous connections (once again, the directives in the *global* stanza and the *host* stanza are merged).

The *sshkey* plugin will copy the public key from the local *nagios* user account to the *.ssh/authorized_keys* file in the home directory of the user *nagios* in the virtual machine *web14* (creating the file and directory as needed).

Configuring MLN Defaults

Within the main configuration file, */etc/mln/mln.conf*, you can set many MLN defaults. The installed version of this file contains extensive comments documenting the available entries. Some of the most useful and important settings are shown in Table 2.

Note that the *-P*, *-T*, and *-F* options to the *mln* command, which override de-

Table 2: MLN Configuration File Entries

Directive	Use	Default
projects /path	Project files directory (override with mln's *-P* option)	*/opt/mln/projects/$USER*
templates /path	Template directory (override with mln's *-T* option)	*/opt/mln/templates*
default_template file	Default VM template	None
default_memory nnnM	Default VM memory size	64MB
default_size	Default VM filesystem size	250MB
service_host name-or-IP	Define server as a location for VMs	None
daemon_allow ip-address-pattern	Specify remote access control	None
daemon_listen_address ip	Network interfaces on which to listen for remote requests	Listen on all interfaces
daemon_status_query host	Include host in *mln daemon_status* command output	None
lvm_vg vg-name	Volume group for VM volumes	None
san_path vg-name	SAN volume group	None
san_path /mount-point	SAN local mount point	None
files /path	Location for files to be copied to VM with files directive (override with mln's *-F* option)	*/opt/mln/files/$USER*

Anzeige
kommt
separat

fault directory locations, appear before the desired subcommand (for example, *mln -P /mln/projects start ldaptest*).

Controlling the MLN Daemon

The MLN daemon, *mlnd*, is started at boot time via the file */etc/init.d/mlnd* (linked to the appropriate *rcn.d* directory).

Also, you can run this script manually with the usual *start*, *stop*, and *restart* arguments.

To start the daemon manually, use the following command:

```
# mln daemon -D /var/run/mln.pid
```

The following command will display the status of the MLN daemon on all hosts specified in the *daemon_status_query* lines in */etc/mln/mln.conf*:

```
# mln daemon_status
```

When you set up MLN to manage virtual machines and networks, it is a good idea to use LVM for flexible VM storage, including expansion capabilities.

Anticipate resource use before deploying virtual machines, and monitor it on an ongoing basis with software like Munin or Cacti.

To limit remote VM management and live migration, use access control and don't forget security. Virtual machines are not inherently more secure than physical systems, contrary to many vendor claims. In fact, in the absence of precautions, they can even be less secure because they offer new forms of attack.

Apply the usual system hardening techniques to virtual machines and, especially, to the physical servers that host them.

Backups

Also, think about backups. Either you can choose to back up virtual machines in the usual manner, within your enterprise backup scheme, or back up virtual machines at the virtual level.

Conclusions

Virtualization products are everywhere. What makes MLN so different is its ability to work in a very wide range of deployments. MLN works well for virtualization beginners because it removes the gritty details of VM configuration files, and, at the same time, you can use MLN to deploy far more complex scenarios than most vendors offer. ■

INFO

[1] VMware Server: *vmware.com/products/server*

[2] Xen: *www.xen.org*

[3] User-Mode Linux: *user-mode-linux.sourceforge.net*

[4] MLN at SourceForge: *mln.sourceforge.net*

Creating Templates for Virtual Machines

Under both Xen and various free and commercial flavors of VMware, creating a virtual machine starts with making an empty virtual machine. On the first boot, an operating system is installed just as it would be on physical hardware, often from the same installation media, or, more recently, from the corresponding CD/DVD image files. Once you have a virtual machine with an installed virtual system, its image could be copied to create new virtual machines, although the copies might require customization.

MLN is designed for complex virtualization tasks. As such, it does not install operating systems from standard media or ISO images; rather, it relies on installed operating system image files – what it calls templates – as the basis for instantiating virtual machines (relying on the ability of VMware and Xen to create fully installed virtual machines as well as empty ones).

A few of the options for creating MLN templates are as follows:

- Copy existing virtual machines and use them as templates. Before copying, however, it is a good idea to boot the virtual machine and make it as generic as possible to allow for different deployment contexts and scenarios. This includes removing local users and groups (because MLN can configure these as required for each virtual machine created from the template); eliminating specific network configurations; and removing device names, */etc/fstab* entries, and so on.
- Download Xen templates from Internet sites (e.g., *jailtime.org*). Note that such images typically correspond to paravirtualized virtual machines, meaning that the included operating system knows that it is running in a Xen virtual environment and contains special features for efficient execution. This also means these systems do not contain bootable kernels but rather rely on the kernel and initial ramdisk on the virtualization server for booting.
- Download VM images for VMware Server from Internet sites (e.g., *virtualappliances.net*, *jumpbox.com*). These images are typically special-purpose virtual machines ready to run a specific application or fulfill a specific purpose. They are normal VMware VM image files (usually *-flat.vmdk* preallocated disk image files). Note that you can convert VMware images for use with Xen with the *qemu-img convert* command.
- The Xen Tools package provides an easy way to create templates for Xen paravirtualized virtual machines from Debian, Ubuntu, Fedora, and other Linux distributions. See the *xen-create-image* command for easy creation and customization.
- With the *dd* command, you can harvest an image from an installed operating system, copying the entire partition into an image file.

Once you have a template, you can modify it easily by mounting it in loopback mode, as in the following examples.

For Xen images:

```
# mount -o loop guest.img ⮐
/somewhere
```

and VMware images:

```
# mount -o loop,offset=⮐
32256 guest-flat.vmdk ⮐
/somewhere
```

If the image is a Linux operating system, you can *chroot /somewhere* to access the image. This allows you to use the VM operating system's own tools to make modifications, something that is especially helpful for ensuring proper functioning when you add software. If the image is a Windows operating system, you will have to use external tools to modify items within it.

Once prepared, templates must be registered with the MLN daemon before you can use them to build virtual machines:

```
# mln register_template -t ⮐
file-system-image-file
```

Also, you can use *rt* as an abbreviation for the *register_template* subcommand.