Virtualization made easy with MLN

Kyrre Begnum - kyrre@iu.hio.no

README

- Introduction to virtualization (Xen, UML)
- Challenges we face often
- MLN, and how it may solve them

Break

- Examples and demos
- Q & A

Intended Audience

- Technicians and researchers who want to learn how to use virtualization as a test-bed
- People who want to use virtualization but find it difficult
- Students who wish to experiment with larger networks on their own machines

Virtualization

- The ability to run one or more isolated operating systems “on top” of another.
  - Xen, User-Mode Linux, Vmware, Virtual PC, Qemu
- The virtual machine has the ability to be either isolated or a part of the LAN

UML and Xen

Both are two open source virtual machine platforms with their own strengths

User-Mode Linux

- A specialized Linux kernel runs as binary
- No root access required
- Flexible
- Slow compared to others

Xen

- More powerfull platform
- Can support different operating systems
- Impressive performance
- Requires root access
- Complicated

Proposed benefits

- Financial savings
- Logistic benefits
- Live migration of virtual machines
- Security through partitioning
- More interesting student assignments
- Faster to set up
- Increased control and convenience

Challenges

- Requires a high technical skill level
- Monitoring and management
- Designing large networks of virtual machines
- Filesystems need to be hand-configured

How can they be solved?

- Our approach
  - A configuration language for virtual machines
  - Logical groups of virtual machines are grouped into *projects*
  - A tool that would parse the configurations and act on the projects

MLN

- A management front-end for virtual machines
- Supports both Xen and User-Mode Linux
- Easy-to-use configuration language
- Used first time to create lab networks for a Firewall/IDS course in 2004

MLN projects

A project can consist of more one or more virtual machines and ethernet switches.

```
global {
        project tutorial
}

switch lan { }

host fish {
        network eth0 {
                switch lan
        }
}

host chips {
        network eth0 {
                switch lan
       }
}
```

Here, two hosts - fish and chips - are connected via a switch called lan.

Configuration range

Guest OS

- Networking interfaces and switches
- Users and groups
- Startup commands
- Additional files that should be copied into the filesystem

Language

- Superclasses
- Variables
- Plug-ins can extend the language

Physical server

- Extra partitions to mount
- Number of CPUs to use (Xen)
- Filesystem backend
- Background console or xterm
- Virtualization platform
- Memory and filesystem size

Interface

- Start and stop entire projects
- Upgrade running projects
- Build across several servers

Superclasses

```
global {
        project tutorial
}

superclass common {
        memory 64M
        free_space 1000M
        template ubuntu-server.ext3
        network eth0 {
                switch lan
                netmask 255.255.255.0
        }
}

host fish {
        superclass common
        network eth0 {
                address 10.0.0.1
        }
}
```

Superclasses allow for configurations of hosts to be consistent and to save space

```
host chips {
        superclass common
        network eth0 {
                address 10.0.0.2
        }
}
```

Variables

You can use variables to keep information consistent across keywords

```
global {
        project tutorial
        $dns_address = 10.0.0.1
}

host dns {
        network eth0 {
                address $dns_address
        }
}

host fish {
        name_server $dns_address
}
```

The variable `$dns_address` is used to make sure the othe hosts point to the right address.

How does MLN create the virtual machines based on a project file?

How it works

You write down the specification of a project in an ascii file:

```
global {
        project tutorial
}

host big {
        template ubuntu-desktop.ext3
        memory 256M
        vcpus 2
        xen
        free_space 1000M
        network eth0 {
                address dhcp
        }

        users {
                kyrre bNfjJIK/hJlfc
        }
        groups {
                admin { kyrre }
        }
}
```

And then you run:

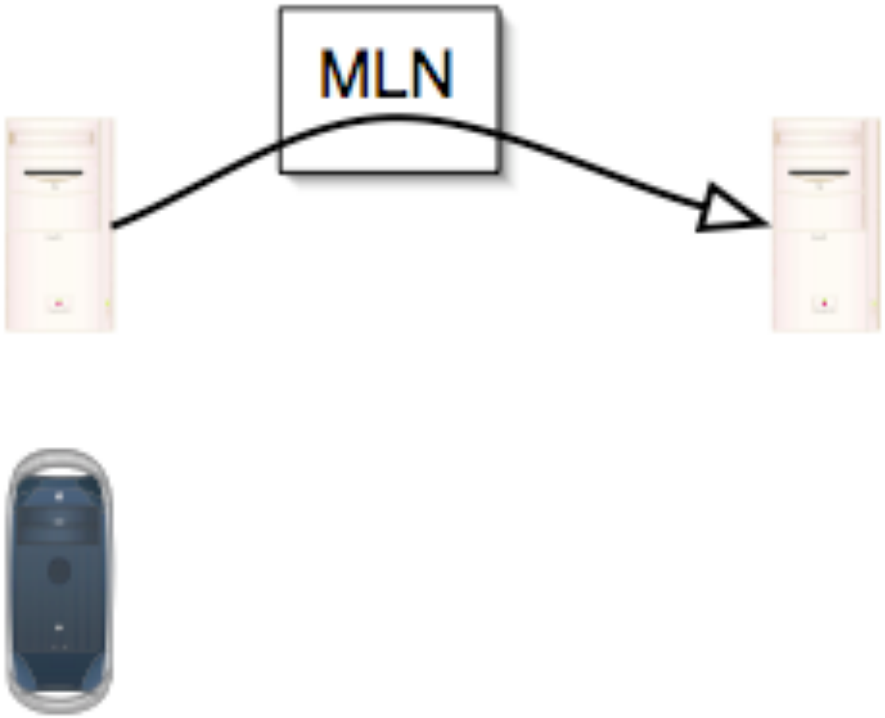
```
mln build -f tutorial.mln
```

How it works

MLN picks the base for a virtual machine based on the default or specified template

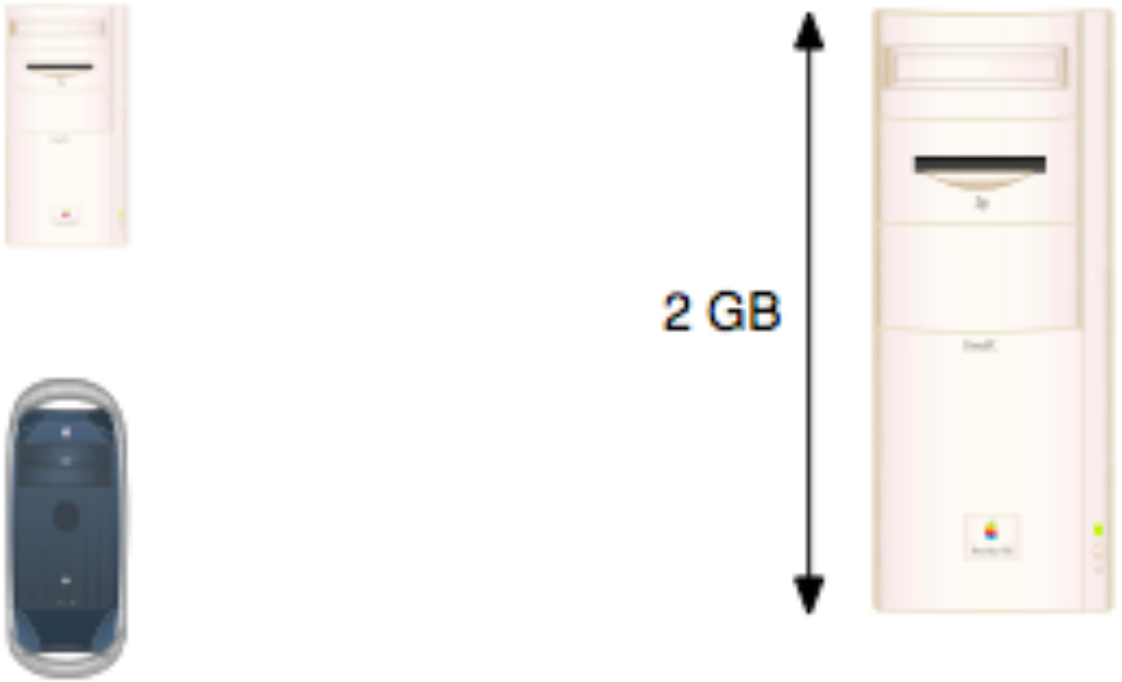
How it works

Next, MLN will make a copy of the template for each new virtual machine.



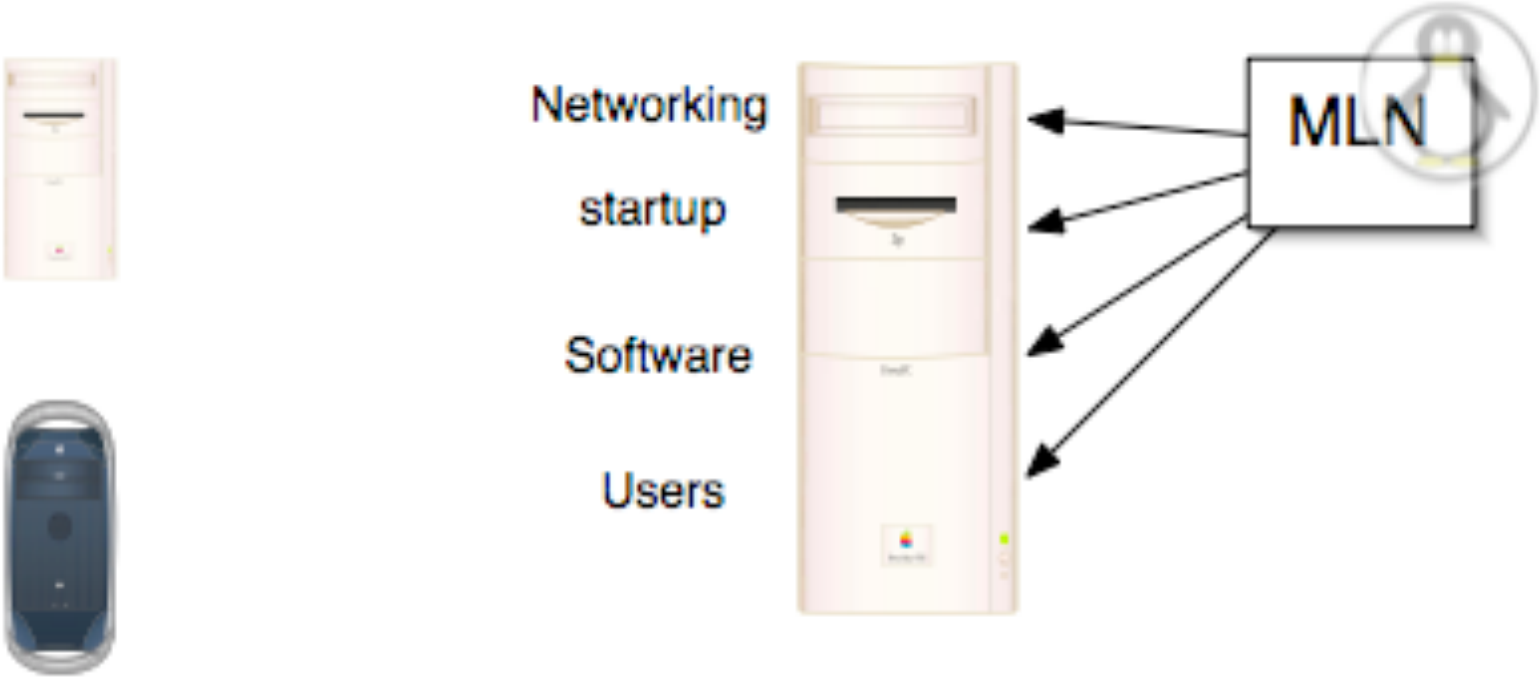
How it works

The new filesystem is resized according to its configuration.



How it works

MLN configures the filesystem according to its specification



How it works

The virtual machine is ready and can be started:

```
mln start -p tutorial
```

A word about templates ...

- Templates are ready-made filesystems
- MLN supports Debian and Ubuntu.
- RedHat and Busybox templates exist also
- Templates can contain installed software and be used as *virtual appliances* like the blimp template.

Examples and Demos

How to set up MLN/Xen/UML

1. Install Xen (optional)
   Many have the most troubles at this point. (http://mln.sf.net/files/xenify.sh)
2. Download MLN

```
wget http://mln.sf.net/files/mln-latest.tar.gz
```

3. Unpack and install

```
tar xzf mln-latest.tar.gz
cd mln-latest
./mln setup
cp mln /usr/local/bin
```

4. Download and register additional templates

MLN will install UML for you.

MLNs configuration

Three important folders:

- **Projects** - Where all the projects and virtual machine filesystems are stored. It may grow very big.
- **Templates** - Where MLN keeps all the templates
- **Files** - MLN can copy files from this location into a guest filesystem at compile-time

Look for the resulting configuration
in `/etc/mln/mln.conf` or `~/.mln`

Example 1 - A simple network

```
global {
          project example1
}

host fish {
          network eth0 {
                    switch lan
                    address 10.0.0.1
                    netmask 255.255.255.0
          }
}

host chips {
          network eth0 {
                    switch lan
                    address 10.0.0.2
                    netmask 255.255.255.0
          }
}

switch lan {}
```

Commands:

```
mln build -f example1.mln
```

```
mln start -p example1
```

or

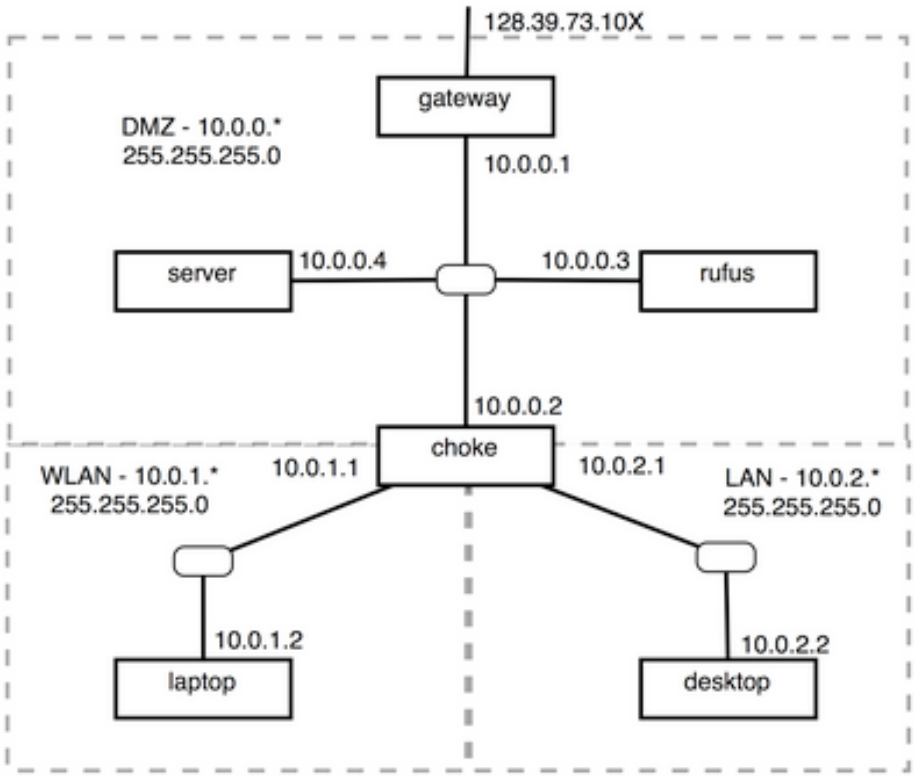
```
mln start -t screen -p example1
```

```
global {
          project example2
}
superclass common {
          xen
          template ubuntu-server.ext3
          term screen
          memory 64M
          nameserver 128.39.89.10
          network eth0 {
                    switch lan
                    netmask 255.255.255.0
          }
}
host gateway {
     superclass common
     network eth1 {
          address dhcp
     }
     network eth0 {
           address 10.0.0.1
     }
     startup {
     iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
     echo 1 > /proc/sys/net/ipv4/ip_forward
     }
}
host backend {
          superclass common
          network eth0 {
                    address 10.0.0.2
                    gateway 10.0.0.1
          }
}
switch lan { }
```

Example 2 - Gateway and backend

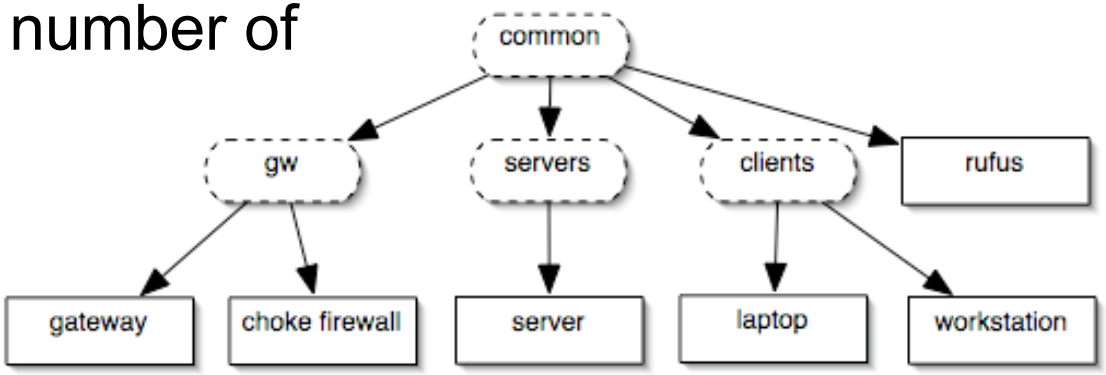
Example 3 - A large network

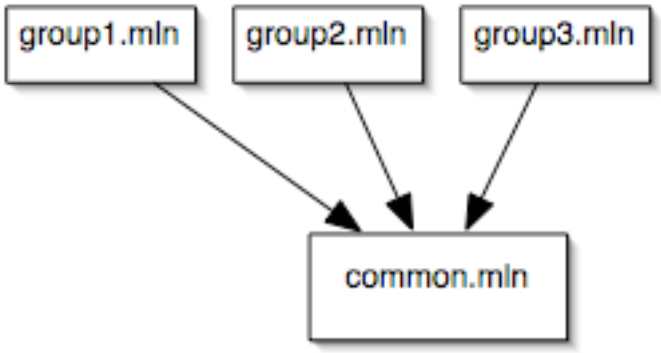
This is the setup from last years course in firewalls and intrusion detection (MS004A). Each student group will have one business network to use for assignements.



The design

A hierarchy of superclasses is used to cut down the number of lines.





...

Next, the files are split up into one project for each group. They all include the same “base”.

Result

The project files for each group were quite small:

```
global {
        project group2
        $owner = group2
        $address = 128.39.73.102
        $mountTarget = /home/group2/ms004a
}


#include common.mln
```

All of the groups could be started using this command:

```
mln -P /huldra/ms004a start -a
```

Example 4 - Managing across several servers

```
global {
        project example4
}

host fish {
     term screen
     service_host shadowfax.vlab
     network eth0 {
         address dhcp
     }
}

host chips {
     term screen
     service_host huldra.vlab
     network eth0 {
         address dhcp
     }
}
```

The keyword `service_host` can be used to assign a vm to a particular server. A mln daemon needs to run on the other servers and its mln.conf modified.

Example 4 - continued

On huldra.vlab:

```
mln daemon
```

On shadowfax.vlab:

```
mln build -f example4
mln start -p example4
```

Example 5 - Upgrading projects

A typical starting point:

```
global {
        project example5
}

host fish {
        free_space 200M
        template ubuntu-server.ext3
        network eth0 {
                address dhcp
        }
}
```

But you end up wanting this:

```
global {
        project example5
}

host fish {
        template ubuntu-server.ext3
        memory 128M
        free_space 1000M
        network eth0 {
                address dhcp
        }
}
```

MLN has the ability to upgrade projects. It will show the diff between the old and the new project and enact the changes.

```
mln upgrade -f example5.mln
```

What we have not covered:

- How to build your own templates
- How to write plug-ins that extend the MLN language
- Performance

Interesting Future Projects

- Self-modifying / self-migrating virtual machines
- Interconnected grids of virtual machines across multiple locations
- Hosting management as a product
- GUI front-ends to MLN
- Support more virtualization platforms

Thank You

Learn more on http://mln.sf.net

Comments / Code / Suggestions are warmly welcomed