Blueprints

The developer's approach to installing and managing KVMs

Blueprints

The developer's approach to installing and managing KVMs

Note

Before using this information and the product it supports, read the information in “Notices” on page 17.

First Edition (March 2010)

Contents

The developer's approach to installing and managing KVMs 1
Scope, requirements, and support 1
KVM overview 2
Host preparation 3
Installing a KVM guest 3
  Creating an XML definition file for your KVM . . 3
  Using virsh to create a KVM 8
  Tips for installing your guest operating system . . 9
  Editing the KVM definition file after operating system installation 10
Propagating your KVMs 11
Managing your KVM 11
Optional: Enabling stateless offloads to improve system performance 12

Appendix. Related information 15

Notices 17
Trademarks 18

The developer's approach to installing and managing KVMs

With Kernel-based Virtual Machine (KVM), you can host different guest operating systems. This Blueprint shows you how to use an XML definition file in combination with the **virsh** tool to install and manage a KVM guest the developer's way. This Blueprint starts by showing you the steps to create a KVM and install an operating system on it. Then it shows you how to manage KVM using the same set of tools. Lastly, it presents you instructions on how to enable stateless offloads to improve system performance. Key tools and technologies discussed in this demonstration include Kernel-based Virtual Machine (KVM), **virt-install**, **virsh**, stateless offloads, and XML definition files.

Scope, requirements, and support

You can learn more about this blueprint, including the intended audience, the scope and purpose, the hardware and software requirements for the tasks detailed in this blueprint, and the types of support available to you.

Intended audience

This Blueprint is intended for advanced Linux system administrators and programmers who want to use KVM to create a virtualized environment.

Scope and purpose

This Blueprint provides information about managing KVMs with XML files and the virsh tool. This is the developer's method for working with KVM. The instructions in this document assume that the KVM guests to be created are using local storage. The process of installing an operating system on a KVM guest is the same as any other installation and is outside the scope of this Blueprint.

Test environment

The instructions in this Blueprint were tested on a System x® HS21 blade system that runs an Intel Xeon chip and is running the stock RHEL 5.4 kernel:

```
# uname -a
Linux testmachine.ibm.com 2.6.18-164.el5 #1 SMP
Tue Aug 18 15:51:48 EDT 2009 x86_64 x86_64 x86_64 GNU/Linux
```

Hardware requirements

Host machines must be running either Intel VT chipsets or AMD-V chipsets that support hardware-assisted virtualization.

For a list of supported chipsets, see http://en.wikipedia.org/wiki/X86_virtualization.

For more information about hardware requirements, see the "Enabling KVM support on your hardware" section of the *Quick Start Guide for installing and running KVM* at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/kvminstall/liaaikvminstallstart.htm.

Other considerations

This Blueprint assumes that the created KVM guests use local storage, however, live migration is only possible if a guest is running on shared storage on RHEL 5.4. If you would like to preserve the possibility of live migration in the future, you must set up shared storage. Or, if you are running a later release of RHEL, check to see if live migration is possible on local storage.

Author names

Monza Lui

Other contributors

Kersten Richter

Santwana Samantray

Sridhar Samudrala

Subrata Modak

Heather Crognale

IBM® Services

Linux offers flexibility, options, and competitive total cost of ownership with a world class enterprise operating system. Community innovation integrates leading-edge technologies and best practices into Linux.

IBM is a leader in the Linux community with over 600 developers in the IBM Linux Technology Center working on over 100 open source projects in the community. IBM supports Linux on all IBM servers, storage, and middleware, offering the broadest flexibility to match your business needs.

For more information about IBM and Linux, go to ibm.com/linux (https://www.ibm.com/linux)

IBM Support

Questions and comments regarding this documentation can be posted on the developerWorks® Systems Management Blueprint Community Forum:

http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1272

The IBM developerWorks discussion forums let you ask questions, share knowledge, ideas, and opinions about technologies and programming techniques with other developerWorks users. Use the forum content at your own risk. While IBM attempts to provide a timely response to all postings, the use of this developerWorks forum does not guarantee a response to every question that is posted, nor do we validate the answers or the code that are offered.

Typographic conventions

The following typographic conventions are used in this Blueprint:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text like what you might see displayed, examples of portions of program code like what you might write as a programmer, messages from the system, or information you should actually type.

KVM overview

Kernel-based Virtual Machine (KVM) is a hardware-assisted, fully virtualization solution for Linux on x86 and x86_64 hardware that contains virtualization extensions – specifically Intel VT or AMD-V.

After you install KVM-related software, you can run multiple guests (virtual machines), with each one running a different operating system image. Each of these virtual machines is assigned private, virtualized hardware.

For a list of KVM-supported guest operating systems, see http://www.linux-kvm.org/page/Guest_Support_Status.

For more information about the technology, see http://www.linux-kvm.org.

Host preparation

Before completing the instructions in this Blueprint, some preparation must be done to the host system.

See the *Quick Start Guide for installing and running KVM* at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/kvminstall/liaaikvminstallstart.htm and complete the instructions in the following sections to prepare your host for creating KVMs:

- Enabling KVM support on your hardware
- Installing and configuring KVM related software
- (Optional) Setting up a network bridge in the host

Installing a KVM guest

There are at least three methods of creating a guest on a KVM host machine: the **virt-manager** tool, the **virt-install** tool, and the **virsh** tool (in conjunction with XML files). This Blueprint will focus on demonstrating the **virsh** tool and XML files to manage KVMs.

For additional information about the **virt-manager** tool and the **virt-install** tool, see the *Quick Start Guide for installing and running KVM* at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/kvminstall/liaaikvminstallstart.htm.

This Blueprint includes instructions for creating the XML definition file manually. Alternatively, you can use the **virt-manager** tool and the **virt-install** tool to create XML definition files for you, and then manage the KVMs with the XML files and the **virsh** tool.

To install a KVM guest, complete the following steps, which are described in this section of the Blueprint:

1. Create an XML definition file which you will use with the **virsh** tool to create a KVM and boot to install.
2. Install the operating system on the guest KVM.
3. Change the XML file and redefine the KVM so that the next time the KVM guest boots, it will boot from the installed image instead of the installation.

Creating an XML definition file for your KVM

Understanding an XML definition file is beneficial for managing KVMs the developer's way. In this section, the anatomy of XML definition files is discussed.

Steps associated with some sections of the file are also explained. You can see how efficient this method is if you need to manage many KVMs or create many KVMs of similar configurations.

This Blueprint is not meant to be a comprehensive guide to writing the XML file. Only the most likely scenarios are covered here. Consult the official libvirt site about KVM XML format at http://libvirt.org/formatdomain.html for more details.

An example XML file to install Linux booting from the installation kernel

The following example XML file was used to create a KVM and boot up to the Linux RHEL 5.4 installation screen. You should be able to use this file as a template to install any Linux guest.

The installation **vmlinuz** and **initrd** images that come in the installation media are used as the boot media. This way, you are able to install Linux from installation source that is either remote or local to the system.

```
1  <domain type='kvm'>
2    <name>kvm3</name>
3    <uuid>bdbb89fb-57d1-4d01-b3b7-ff33a9346ae6</uuid>
4    <memory>2048000</memory>
5    <currentMemory>1024000</currentMemory>
6    <vcpu>4</vcpu>
7    <os>
8      <type arch='x86_64' machine='pc'>hvm</type>
9      <kernel>/tmp/vmlinuz-rhel54</kernel>
10     <initrd>/tmp/initrd-rhel54.img</initrd>
11     <boot dev='hd'/>
12   </os>
13   <features>
14     <acpi/>
15     <apic/>
16     <pae/>
17   </features>
18   <clock offset='utc'/>
19   <on_poweroff>destroy</on_poweroff>
20   <on_reboot>destroy</on_reboot>
21   <on_crash>destroy</on_crash>
22   <devices>
23     <emulator>/usr/libexec/qemu-kvm</emulator>
24     <disk type='file' device='disk'>
25       <source file='/var/lib/libvirt/images/kvm3.img'/>
26       <target dev='hda' bus='ide'/>
27     </disk>
28     <interface type='bridge'>
29       <source bridge='br0'/>
30       <mac address="3B:6E:01:69:3A:11"/>
31     </interface>
32     <input type='mouse' bus='ps2'/>
33     <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
34   </devices>
35 </domain>
```

The following information explains each line of the definition file in detail.

Start of the KVM guest definition.

```
1  <domain type='kvm'>
```

Short name of the guest. This short name must consist only of letters and numbers, but no blank spaces.

```
2    <name>kvm3</name>
```

Each guest needs a *universal unique identifier* (*uuid*).

```
3    <uuid>bdbb89fb-57d1-4d01-b3b7-ff33a9346ae6</uuid>
```

You must generate one for your guest by running the following command, and then copy and paste the identifier into the XML file:

```
# uuidgen
```

Maximum memory the guest can use, in kilobytes (KB). In this example, 2 GB is the maximum amount of memory the guest is allowed to use without needing to reboot the guest.

```
<memory>2048000</memory>
```

Memory allocation when started. This number can be changed to not larger than what is set in the memory element using the **virsh setmem** command.

```
<currentMemory>1024000</currentMemory>
```

Number of virtual processors allocated to the guest.

```
<vcpu>4</vcpu>
```

The *os* element specifies how your guest is booted. You can use `i686` in the *arch* attribute to set up an x86 guest. The `hvm` type indicates that this guest runs an operating system that was designed for bare metal and is therefore a fully virtualized guest.

```
<os>
  <type arch='x86_64' machine='pc'>hvm</type>
  <kernel>/tmp/vmlinuz-rhel54</kernel>
  <initrd>/tmp/initrd-rhel54.img</initrd>
  <boot dev='hd'/>
</os>
```

The *kernel* and *initrd* elements are especially useful if your Linux installation source is remote. You can use these elements to boot from the installation kernel. In RHEL 5, these two files are located in the **images/pxeboot** directory and are named **vmlinuz** and **initrd.img**. In SLES 11, the files are located in the **boot/<arch>/loader** directory and are named **linux** and **initrd**. Copy them to the local disk and provide their locations to the *kernel* and *initrd* elements. In this example, these files were copied from the RHEL 5.4 installation media, placed in the **/tmp** directory, and renamed to include “-rhel54” in their names.

If SELinux is enabled in the host machine, you also need to change the type of security context for the files to **virt_image_t** to allow **libvirtd** to access them for booting:

```
# chcon -t virt_image_t /tmp/vmlinuz-rhel54
# chcon -t virt_image_t /tmp/initrd-rhel54.img
```

After you change the security context, verify that the correct security context is assigned to them as shown in the following example:

```
# ls -Z /tmp|grep virt
-r--r--r--  root root root:object_r:virt_image_t         initrd-rhel54.img
-r--r--r--  root root root:object_r:virt_image_t         vmlinuz -rhel54
```

The *boot* element specifies hd as the next boot device, which indicates a hard disk drive. You can have repeated lines of boot elements with different values to form a boot order list.

Optionally you can also include the boot options in the *cmdline* element. The following example demonstrates the use of the **kickstart** file for installation:

```
<os>
  <type arch='x86_64' machine='pc'>hvm</type>
  <kernel>/tmp/vmlinuz-rhel54</kernel>
  <initrd>/tmp/initrd-rhel54.img</initrd>
  <cmdline>method=http://10.1.1.212/install/rhel5.4/x86_64
    ks=http://10.1.1.212/install/autoinst/c20m2n05v3</cmdline>
  <boot dev='hd'/>
</os>
```

Processor features to be included.

```
<features>
  <acpi/>
  <apic/>
  <pae/>
</features>
```

Clock time.

```
<clock offset='utc'/>
```

This set overrides the default actions when a power off, reboot, or crash occurs in the KVM environment. The `destroy` action is also used for the *on_reboot* element in this example because once installation is completed, the wanted action is for the guest to stop so that the definition of the guest can be edited to boot from the installed guest operating system rather than from the installation kernel or ISO. Other allowed actions include: `restart`, `preserve`, and `rename-restart`.

```
<on_poweroff>destroy</on_poweroff>
<on_reboot>destroy</on_reboot>
<on_crash>destroy</on_crash>
```

The definition of the devices element begins.

```
<devices>
```

The emulator element should be **qemu-kvm** for all KVM guests.

```
  <emulator>/usr/libexec/qemu-kvm</emulator>
```

In this example, the disk element identifies the local **/var/lib/libvirt/images/kvm3.img** file as the storage of the KVM, and it shows up as an IDE device named `hda` inside the guest.

```
  <disk type='file' device='disk'>
    <source file='/var/lib/libvirt/images/kvm3.img'/>
    <target dev='hda' bus='ide'/>
  </disk>
```

Because this method is a manual method of creating the KVM, you are expected to create the image file. You can create this file with 10 GB of storage by running the following command:

```
# qemu-img create /var/lib/libvirt/images/kvm3.img 10GB
```

Note: The **/var/lib/libvirt/images/** directory is the default directory for KVM images.

More than one set of disks can be defined and exist in the KVM guest, just like any physical machine can have more than one disk.

Other common possibilities of a disk element definition are:

- Logical volume device:

  ```
  <disk type='block' device='disk'>
    <source 'dev=/dev/mapper/VolGroup01-LVM1'/>
    <target dev='hdb'/>
  </disk>
  ```

- Disk partition:

  ```
  <disk type='block' device='disk'>
    <source dev='/dev/sda4'/>
    <target dev='hdb' bus='ide'/>
  </disk>
  ```

- CD-ROM device:

  ```
  <disk type='block' device='cdrom'>
    <source 'dev=/dev/sde'>
    <target dev='hdb'/>
    <readonly/>
  </disk>
  ```

It is important that each KVM has a unique MAC address, just like its *uuid*. Otherwise, it might cause traffic disruptions to all machines in the same subnet. In this example, the Linux bridge you set up previously is used and a randomly generated MAC address is assigned. Random MAC addresses can be generated in many ways. For example, to see a **python** script suggested by RedHat, see the *RHEL 5 Virtualization Guide* at http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/Virtualization-en-US/ch19s22.html.

```
    <interface type='bridge'>
      <source bridge='br0'/>
      <mac address="3B:6E:01:69:3A:11"/>
    </interface>
```

The following example demonstrates the use of the default virtual network instead of the bridge:

```
 <interface type='network'>
   <source network='default'>
   <mac address="3B:6E:01:69:3A:11"/>
 </interface>
```

Defines the input device it simulates.

```
    <input type='mouse' bus='ps2'/>
```

Defines the graphical device being used to interact with the guest. In this example, the `vnc` protocol is specified. A port value of -1 indicates that a port number is automatically assigned. If you are using other VNC viewers to connect to the host, you can find out the port number used by running the following command:

```
# virsh vncdisplay <KVM Guest Name>
:0
```

This example output indicated that port 0 is used. Alternatively, you can specify the port number you would like to use in this field.

```
    <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
```

End of the devices element definition.

```
  </devices>
```

End of the KVM's definition.

```
</domain>
```

An example XML file to install an operating system from an ISO image

The following example XML file can be modified to create a KVM and boot to any operating system from an ISO image or a CD-ROM drive.

```
<domain type='kvm'>
  <name>kvm4</name>
  <uuid>85badf15-244d-4719-a2da-8c3de0641373</uuid>
  <memory>1677721</memory>
  <currentMemory>1677721</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='cdrom'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='localtime'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>destroy</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/kvm4.img'/>
      <target dev='hda' bus='ide'/>
    </disk>
    <disk type='file' device='cdrom'>
      <source file='/tmp/SLES-11-DVD-i586-GM-DVD1.iso'/>
      <target dev='hdb' bus='ide'/>
      <readonly/>
    </disk>
    <disk type='file' device='cdrom'>
      <source file='/tmp/SLES-11-DVD-i586-GM-DVD2.iso'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
    </disk>
    <interface type='bridge'>
      <source bridge='br0'/>
      <mac address="0E:1F:35:AB:45:0C"/>
    </interface>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
  </devices>
</domain>
```

Compared to the previous example, you can see that if you want to boot from a CD or ISO image, you must first specify that you want to boot from the CD-ROM drive:

```
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='cdrom'/>
  </os>
```

Then you need to specify the location of the ISO device in the host system. Also, this definition suggests it to be assigned as the `hdb` device inside the KVM guest:

```
  <disk type='file' device='cdrom'>
    <source file='/tmp/SLES-11-DVD-i586-GM-DVD1.iso'/>
    <target dev='hdb' bus='ide'/>
    <readonly/>
  </disk>
```

In this example, the CD-ROM drive is the host's `/dev/sde` (block) device, in which this definition suggests it to be assigned as the `hdc` inside the KVM guest:

```
<disk type='block' device='cdrom'>
  <source dev='/dev/sde'>
  <target dev='hdc'/>
  <readonly/>
</disk>
```

If your host has SELinux enabled, you must re-label the security context of the ISO or CD-ROM device so that **libvirtd** is eligible to use the device file:

```
#chcon -t virt_image_t /tmp/SLES-11-DVD-i586-GM-DVD*
```

Check the labels by running the following command:

```
#ls -Z /tmp/
-rw-r--r--  root root system_u:object_r:virt_image_t   SLES-11-DVD-i586-GM-DVD1.iso
-rw-r--r--  root root system_u:object_r:virt_image_t   SLES-11-DVD-i586-GM-DVD2.iso
```

Using virsh to create a KVM

When you are finished creating or editing your XML definition file, create your KVM using the virsh tool.

Procedure

1. Define your KVM by running:
   ```
   virsh define <Name of XML definition file>
   ```

For example:

```
# virsh define kvm4.xml
```

In the background, a copy of the XML file is copied to the /etc/libvirt/qemu directory and the new KVM is officially defined.

2. Start the KVM so that the installation of its operating system can begin:

   ```
   virsh start <Name of KVM>
   ```

   For example:

   ```
   # virsh start kvm4
   ```

   If your KVM does not start, complete one of the following to redefine the KVM:

   a. Edit the defined KVM definition by running the following command and revising the corresponding definition file:

      ```
      virsh edit <Name of KVM>
      ```

      For example:

      ```
      # virsh edit kvm4
      ```

   b. Undefine the KVM and then edit your copy of the file and redefine it as in Step 1. To undefine an existing KVM, run the following command:

      ```
      virsh undefine <Name of KVM>
      ```

      For example:

      ```
      # virsh undefine kvm4
      ```

3. To check which KVM guests are running, run the following command:

   ```
   # virsh list
   Id Name                State
   ----------------------------------
   1 kvm1               running
   2 kvm2               running
   3 kvm3               running
   4 kvm4               running
   ```

Tips for installing your guest operating system

Some tips for installing your guest operating system, including references to additional information for installing RHEL 5, SLES 11, and Windows XP, are included in this topic.

After the KVM is started, or for any reason you lost the screen, connect to the install screen using **virt-viewer**, **virt-manager**, or your choice of VNC viewer:

- To connect using the **virt-viewer** tool, run the following command:

  ```
  # virt-viewer <Name of KVM>
  ```

- To connect using the **virt-manager** tool, run the following command:

  ```
  # virt-manager
  ```

  After the **virt-manager** window opens, click the KVM and click the **Open** button on the menu bar.

- If you forgot the name of your guest, run the following command to list all the running KVM guests:

  ```
  # virsh list
  ```

- To figure out which port the KVM is connected to, in case you want to connect with other VNC viewers, run the following command:

  ```
  # virsh vncdisplay <Name of KVM>
  ```

- If your mouse is trapped in the installation screen during installation, regain control of the mouse by pressing Alt+Ctrl.

For network configuration:

When your installation screen appears, you can start installing your KVM guest's operating system as you would in any other physical machine. If you are using the Linux bridge for a network connection, specify it during the network setup step. If you are using the default virtual network, be aware that choosing to use DHCP gives the guest a 192.168.122.x address. If you have a preference as to which 192.168.122.x address is assigned to the guest, choose manual configuration and use any unused address from this subnet. The subnet mask should be 255.255.255.0 and the gateway should be 192.168.122.1, the host's address.

For operating system installation:

The following links provide additional information about how to install RHEL 5, and SLES 11, and Windows XP:

Red Hat Enterprise Linux 5 Installation Guide at http://www.redhat.com/docs/manuals/enterprise/.

SLES 11 Installation Quick Start at http://www.novell.com/documentation/sles11/.

Quick Start Guide for installing and running KVM at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/kvminstall/liaaikvminstallstart.htm.

Editing the KVM definition file after operating system installation

To boot to the installed guest after the guest operating system installation is complete, revise the guest definition so that it will boot from its hard disk drive.

The following example shows you how the revised XML definition file looks if the **kvm4** guest is to boot from the installed disk:

```
<domain type='kvm'>
  <name>kvm4</name>
  <uuid>85badf15-244d-4719-a2da-8c3de0641373</uuid>
  <memory>1677721</memory>
  <currentMemory>1677721</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='localtime'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/kvm4.img'/>
      <target dev='hda' bus='ide'/>
    </disk>
    <interface type='bridge'>
      <source bridge='br0'/>
      <model type='virtio'/>
      <mac address="0E:1F:35:AB:45:0C"/>
    </interface>
    <input type='tablet' bus='usb'/>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
  </devices>
</domain>
```

The *boot* element at line 9 is now pointing to hard disk, `hd`, instead of CD-ROM, and the ISO disk definitions are removed because they are no longer needed.

To redefine the KVM to this definition, complete one of the following steps:
- Undefining the KVM, editing your copy of the definition file, then defining the KVM again.
- Revising the master copy using the **virsh edit <Name of KVM>** command.

Both of these options are explained in "Using virsh to create a KVM" on page 8.

Propagating your KVMs

With an understanding of how the underlying XML definition file works, you can manipulate and manage the KVMs in various ways. For example, if your task is to create 10 identical KVMs that have simple configurations, you can propagate existing KVMs rather than manually installing 10 identical KVMs.

Procedure

1. Create one KVM using **virt-install**. For this example, the created KVM is named templateKVM.
2. Finish the operating system installation and confirm that the installation was successful.
3. Stop the KVM with the **virsh** tool:
   ```
   # virsh shutdown templateKVM
   ```
4. Make copies of the KVM disk image by using the **qemu-image** command with the convert option:
   ```
   # qemu-img convert templateKVM.img -O raw NewKVM1.img
   ```
5. Make copies of the templateKVM XML definition file. You can find it in the **/etc/libvirt/qemu/** directory or you can run the following command to see the definition file.
   ```
   # virsh dumpxml templateKVM
   ```
6. Edit the definition files and define the new KVMs as demonstrated in "Using virsh to create a KVM" on page 8. It is important that a unique MAC address is used in the definition file to make sure that the network functions without a problem.
7. Start one KVM at a time and reconfigure its network so that a unique set of MAC and IP addresses is used.

Managing your KVM

You can use the **virsh** tool to perform several managing tasks on your KVM. Some commonly used **virsh** commands that are not mentioned previously in this Blueprint are described in this topic.
- To shut down a running KVM gracefully:
  ```
  # virsh shutdown <Name of KVM>
  ```
- To shut down a running KVM without notifying the guest operating system:
  ```
  # virsh destroy <Name of KVM>
  ```
- To list all virtual networks:
  ```
  # virsh net-list
  Name                 State      Autostart
  -----------------------------------------
  default              active     yes
  ```
- To get the XML definition of a particular network:
  ```
  # virsh net-dumpxml <Name of network>
  ```
  For example:
  ```
  # virsh net-dumpxml default
  <network>
    <name>default</name>
    <uuid>ff2df37e-d22e-4bdd-8a04-4e5280488b29</uuid>
    <forward mode='nat'/>
  ```

```
    <bridge name='virbr0' stp='on' forwardDelay='0'/>
    <ip address='192.168.122.1' netmask='255.255.255.0'>
      <dhcp>
        <range start='192.168.122.2' end='192.168.122.254' />
      </dhcp>
    </ip>
  </network>
```

- To display information about the host:

  ```
  # virsh nodeinfo
  CPU model:           x86_64
  CPU(s):              8
  CPU frequency:       1596 MHz
  CPU socket(s):       2
  Core(s) per socket:  4
  Thread(s) per core:  1
  NUMA cell(s):        1
  Memory size:         8150212 kB
  ```
- To set the running memory of a KVM:

  ```
  # virsh setmem <Name of KVM> <Amount of memory in KB>
  ```

More information about how to use the **virsh** tool is available in the man pages and in the "Managing guests with virsh" section of the *RHEL Virtualization Guide*, available at http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5.4/html/Virtualization_Guide/.

You can also find more information about other KVM management tools, such as **virt-manager** and **ovirt**, at http://www.linux-kvm.org/page/.

Optional: Enabling stateless offloads to improve system performance

Stateless offloads, such as checksum offload, segmentation offload, and large receive offload, can improve system performance by moving some of the processor-intensive tasks that do not require a connection to the network adapter.

Procedure

1. Make sure that all the available stateless offloads are enabled by running the following command on the host's physical interface that corresponds to the network bridge you set up previously:

   ```
   # ethtool -k eth0
   Offload parameters for eth0:
   rx-checksumming: on
   tx-checksumming: on
   scatter-gather: on
   tcp segmentation offload: on
   udp fragmentation offload: off
   generic segmentation offload: on
   generic-receive-offload: off
   ```

   The following offloads can help boost performance when they are enabled. The corresponding identifier is used by the **ethtool**:

Table 1. Offloads that can help boost performance

Offload parameter	**Corresponding <*offload-option*> value**
rx-checksumming	rx
tx-checksumming	tx
scatter-gather	sg
tcp segmentation offload	tso
generic-receive-offload	gro

2. To enable an offload, issue the following command:

   ```
   # ethtool -K ethX <offload-option> on
   ```

   For example, to enable the generic-receive-offload, run the following command:

   ```
   # ethtool -K eth0 gro on
   ```

3. Run the **ethtool -k ethX** command again to verify that all five target offloads are enabled:

   ```
   # ethtool -k eth0
         Offload parameters for eth0:
         rx-checksumming: on
         tx-checksumming: on
         scatter-gather: on
         tcp segmentation offload: on
         udp fragmentation offload: off
       generic segmentation offload: on
       generic-receive-offload: on
   ```

Appendix. Related information

You can find additional information about KVM and the tools used to manage KVM at any of the listed sources.

KVM at http://www.linux-kvm.org/page/Main_Page

RHEL 5 Virtualization Guide at http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5.4/html/Virtualization_Guide/

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. LRAS/Bldg. 903
11501 Burnet Road
Austin, TX 78758-3400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Trademarks

Printed in USA