University of Victoria
Faculty of Engineering
Spring 2009 Work Term Report

Creating and Controlling KVM Guests using libvirt

Department of Physics
University of Victoria
Victoria, BC

Matthew Vliet
V00644304
Work Term 2
Computer Engineering
mvliet@uvic.ca

April 30, 2009

In partial fulfillment of the requirements of the
Bachelor of Computer Engineering Degree

Supervisor's Approval: To be completed by Co-op Employer

I approve the release of this report to the University of Victoria for evaluation purposes only.
The report is to be considered (**select one**): ☐ NOT CONFIDENTIAL ☐ CONFIDENTIAL

Signature: _____ Position: _____ Date: _____

Name (print): _____ E-Mail: _____ Fax #: _____

If a report is deemed CONFIDENTIAL, a non-disclosure form signed by an evaluator will be faxed to the employer. The report will be destroyed following evaluation. If the report is NOT CONFIDENTIAL, it will be returned to the student following evaluation.

Contents

List of Figures

Creating and Controlling KVM Guests using libvirt

Matthew Vliet
mvliet@uvic.ca

April 30, 2009

Abstract

The Kernel-based Virtual Machine(KVM) hypervisor provides users with the ability to run one or more virtualized computer systems concurrently one the same physical hardware. Although KVM provides the tools necessary to both create and run virtual machines, management and control of these virtual machines can be a daunting task. The libvirt library and accompanying projects provide a simple and convenient method for the control of virtual machines in a largely hypervisor unspecific way. This report will explore the use of the libvirt library and accompanying projects to simplify the management of KVM based virtual machines.

1 Report Specification

1.1 Audience

This report is directly targeted towards the members of the High Energy Physics Grid Computing Group at the University of Victoria. Indirectly this report is targeted to anyone who works with or has an interest in using the KVM hypervisor or the libvirt virtualization library.

1.2 Prerequisites

To obtain the most from this report, it is assumed that one has a general knowledge of virtualization technologies. In specific, a knowledge of general Linux systems as well as the KVM hypervisor would be very beneficial to the reader.

1.3 Purpose

This report provides the reader with an overview of the creation and control of virtual machines using the KVM hypervisor and libvirt virtualization tools.

2 Introduction

2.1 KVM

The Kernel-based Virtual Machine(KVM) is a virtual machine hypervisor that runs on Linux computer systems. KVM, now under development by Red Hat, consists of two main components, a set of loadable kernel modules and a KVM userspace. Due to its integration within the Linux kernel, KVM is set to replace Xen as the hypervisor of choice for Red Hat distributions.

The KVM userspace is a modified version of the QEMU emulator that is used for emulating hardware devices such as displays or networking chipsets. At the time of writing this report, QEMU and KVM are well underway in the process of merging their source trees into one main tree that will allow for faster and easier development for both projects.

The KVM kernel modules consist of three modules, only two of which must be loaded for a given system. A main module, and depending on what architecture your computer runs, a platform specific module must be loaded. The main module provides all core virtualization infrastructure, and the platform specific module provides access to the CPU virtualization extensions for you platform. At the current time, both AMD-V and Intel VT virtualization extensions are supported. If a machine lacks the CPU virtualization extensions KVM will not be able to function properly.

2.2 libvirt

Libvirt, the virtualization API, provides a common layer of abstraction and control for virtual machines deployed within many different hypervisors. The main components of libvirt are a control daemon, a stable C language API and a corresponding set of Python language bindings, as well as a shell environment. There are currently a number of open source projects, such as virt-manager and virt-install, that use libvirt as their underlaying virtual machine control mechanism. Libvirt stores information such as the disk image and networking configuration in an XML file that is independent of the hypervisor in use.

2.3 virt-install

Virt-install is a python script developed and maintained by the Virt-Manager project. The script enables users to easily create and define guest virtual machines. Configurations created by virt-install are stored within `/etc/libvirt/qemu/`

3 Creating Guest Images

There are many options available for creating virtual machine guest images. At one end of the spectrum, one can use a graphical program such as Virt-Manager and follow a wizard like configuration, at the other end of the spectrum a text editor can be used to manually write a libvirt XML file describing a guest virtual machine configuration. These options both have their own benefits and also draw backs. When using the graphical install program you lose fine grain control on your configuration, yet if you were to manually write a configuration file it would take much more time and be more prone to errors. A good middle ground between the two is to use virt-install. Although you must have a greater understanding of a virtual machine configuration than with the graphical installation method, you can more closely control specific aspects of a guest such as what driver to use for the network interface.

3.1 Minimal Configuration

An absolute minimum configuration for a libvirt managed guest virtual machine consists of the virtual machine name, an amount of ram to give an instance when it starts up, a disk image, and a location for the installation media. In the case of example in Figure 1 the location is given by the `--cdrom` option which points to an ISO image.

Figure 1: virt-install minimal options.

```
virt-install --connect qemu:///system \
    --name sl53 \
    --ram 256 \
    --disk path=/home/mvliet/sl53-pvm.img,size=10,sparse=true \
    --accelerate \
    --cdrom /home/mvliet/SL.53.031809.DVD.i386.disc1.iso
```

Once you start the virt-install program with all the necessary options, it will guide a user through the rest of the install process. This process is the same as the installation of any Linux distribution and as such will not be explored further. When the installation is completed an XML configuration file will have been created under `/etc/libvirt/qemu`.

Although the `--connect` option is not explicitly needed, it is a good practice to included it when creating a quest to avoid any ambiguity between which hypervisor you are using. The `--accelerate` option is required when creating a KVM guest image, without it libvirt will fallback to using QEMU emulation with no KVM hardware acceleration.

3.2 Disk Image Options

There are many options available when creating disk images. The most common options used will be the path to the disk image, the size of the disk, the device bus type and the permissions given to the disk. When specifying the path of the disk, if a non-existent path is given libvirt will attempt to create a new disk image at the given location. If the path already exists, then libvirt will attempt to use the specified file as the disk image.

3.3 Networking Options

Libvirt provides some amount of flexibility when it comes to creating networking configurations. By default libvirt provides NAT forwarding, also known as virtual networks, to guests. This is the simplest type of networking available, and as such provides only limited connectivity options. Only outgoing connections can be established with machines, either virtual or physical, not on the same virtual network. This however is sufficient for the vast majority of networking requirements.

The preferred type of networking for use within the UVIC HEP group is a full bridging network, which shares the physical device. Although this method takes some amount of configuration on the host node, it allows each virtual machine to access the network as if it had its own physical network interface. Guests are able to obtain IP addresses via DHCP, or some other means available on the network, and appear as a physical machine to the rest of your network. To make use of bridged networking, the host machine must have the `brctl` package installed and the main network interface must be configured to be attached to a bridge. When a guest instance is started libvirt will add a new tap device to the bridge and assign that tap to the guest instance allowing for automatic network configuration of each guest instance.

Each guest instance can be configured with an arbitrary number of networking interfaces, as well as a mix of bridged and NAT networking. This allows large flexibility in the configuration and connectivity of groups of instances running on a node.

3.4 Paravirtualized Device Drivers

Since KVM is based on hardware virtualization, every device that a guest instance uses must be emulated. This emulation can be both complex and very inefficient. In an effort to provide better performance to guests, KVM supports a set of drivers named virtio. These drivers, when installed on a guest machine, have the ability to communicate directly with the hypervisor without the need for device emulation leading to large performance increases. Currently virtio supports network and block devices, the two most I/O intensive types of devices on a machine. Enabling virtio in a guest instance is a very simple matter provided that the guest kernel has the virtio drivers either built-in or available as loadable modules. By simply adding a `bus=virtio` option to the `--disk` virt-install switch virtio will be enabled for your disk image.

3.5 Display Options

Libvirt makes use of the VNC protocol to provide access to the graphical screen of a guest instance. Connection to the guest instance can be accomplished using any available VNC viewer. Within the HEP group at

UVIC, graphical screens are not used for virtual machines so adding the `--nographics` option to virt-install is often useful.

3.6 Serial Console

When working on a headless node it is often a requirement to have the ability to access a guest instance via a console. Although one could access an instance via ssh, this is only possible if the IP or host name of the instance is known ahead of time. Knowledge of the IP address and host name is not always known, especially if DHCP is being used to control IP address assignment on the network. In these cases it is necessary to have the ability to login to an instance from the host node. Since KVM uses full hardware virtualization, the only way to currently do this is to configure your guest images with a serial console.

Although many Linux distributions come preconfigured with a serial console, distributions such as Scientific Linux do not. To configure a serial console within SL, the code in Figure 2 must be added to `/etc/inittab`, as well as adding `ttyS0` to `/etc/securetty`.

Figure 2: Serial console code

```
T0:123:respawn:/sbin/agetty -L ttyS0 9600 vt100
```

4 Controlling Virtual Machines

4.1 virsh

Virsh is a console program maintained by the Virt-Manager team. It provides a easy to use console interface to the libvirt library for controlling guest instances. Each of the commands available within virsh can be used either from within the virsh environment itself, or called from a standard Linux console.

To start a virsh environment one must simply run the `virsh` shell program with no options. This will open a new console like environment in which you can run any of the built in commands for virsh. An example of this can be seen in Figure 3.

Figure 3: Example of using virsh in interactive mode.

```
[root@node ~]# virsh
Welcome to virsh, the virtualization interactive terminal.

Type:  'help' for help with commands
       'quit' to quit

virsh # list
 Id Name                 State
----------------------------------
  1 sl53                 running
  2 Lenny                running
```

To use the virsh commands from within a Linux terminal one simply runs `virsh` followed by the command name and command options. An example of this can be seen Figure 4.

Figure 4: Example of using virsh standalone.

```
[root@node ~]# virsh list
 Id Name                 State
----------------------------------
  1 sl53                 running
  2 Lenny                running
```

4.2 Defining Instances

Before using a virtual machine that has been created, libvirt must first be informed of its existence. The process of informing libvirt about configuration details is referred to as defining a virtual machine. If a virtual machine was created from virt-install it will already have defined with libvirt. If however a virtual machine has been downloaded from the internet or was created on a different machine, it must be defined before it can be used. Using the virsh `define` option accomplishes this, as seen in Figure 5.

Figure 5: Defining a virtual machine with virsh

```
[root@node ~]# virsh define /root/images/sl53.XML
Domain sl53 defined from /root/images/sl53.XML
```

Any changes made to the virtual machine configuration requires the virtual machine to be redefined for those changes to take effect. Redefining can only be done when the virtual machine instance is not running.

4.3 Starting Instances

Starting an instance is as simple as using the `start` option for virsh followed by the name of the virtual machine to start. This can be seen in Figure 6.

Figure 6: Starting an instance with virsh.

```
[root@node ~]# virsh start sl53
Stared domain sl53
```

When an instance is started it is assigned a unique ID number. ID numbers are assigned sequentially starting at 1, there is no way to reset the counter, or alter an instances ID number. Libvirt, and in turn virsh, uses either the instance name or its unique ID number to refer to running instances when running commands.

4.4 Interacting with Instances

To interact with a virtual machine instance there are three main options. The first involves using SSH to run remote commands on your instance, since this is common practice for Linux/Unix environments this will not be explored further. The other options are using VNC to connect to you instance and interact via a graphical environment, and to use a serial console to access the instances terminal.

When connecting via VNC any standard VNC client can be used, however to make the matter simpler a program called `virt-viewer` can be used. Virt-viewer can connect to virtual machines by using their libvirt name or unique ID number instead of a full VNC network address.

When connecting via a serial console it is much the same as using SSH, with the exception that you are connecting from a local machine and not via the network. Again, the virsh program is used. Specifying the `console` option followed by a unique instance identifier such as the name or ID number will open the serial console. This can be seen in Figure 7.

Figure 7: Opening a serial console with virsh.

```
[root@node ~]# virsh console 1

Scientific Linux SL release 5.3 (Boron)
Kernel 2.6.18-128.1.6.el5 on an i686

localhost.localdomain login:
```

To exit from the serial console, the `CTRL-]` key combination is used to return to the host terminal.

4.5 Suspending Instances

KVM has the ability to pause a running instance and later resume it to its running state. When an instance is in a paused state it consumes no CPU resources, it does however still consume memory just as any stopped process does. Libvirt provides easy access to pausing and resuming instances via the virsh program and the `pause` and `resume` commands.

4.6 Shutting Down and Destroying Instances

There are two options available to stop a running instance. The first option is to destroy an instance. When an instance is destroyed it is analogous to pulling the power on a computer. This can have its advantages as the instance will be immediately terminated, yet it also has its disadvantages as it can cause corruption to data just as pulling the power on a real computer can. The second option is to use the `shutdown` command. When using the shutdown command, an ACPI shutdown signal is sent to the instance and a proper shutdown sequence is initiated.

5 Manually Modifying Guest Configurations

If the need arrieses to modify the configuration file of a virtual machine, the XML configuration can be modified by hand. Extreme caution bust be taken when editing this file as it can easily be misconfigured.

5.1 XML Structure

The overall structure of an XML configuration file is straight forward can be seen in Figure 8. Most of the sections are very self explanatory such as name and memory. Other sections require more explanation.

Figure 8: libvirt XML configuration layout.

```
<domain type='kvm'>
    <name></name>
    <uuid></uuid>
    <memory></memory>
    <currentMemory></currentMemory>
    <vcpu></vcpu>
    <os>
        #operating system type
        #boot device
    </os>
    <features>
        # features such as ACPI are defined here
    </features>
    <devices>
        <emulator>/usr/bin/qemu-kvm</emulator>
        # Disks, network interfaces, etc. are defined here.
    </devices>
</domain>
```

5.1.1 The os Section

This section contains information about the architecture of the machine, as well as the type of virtualization being used and the boot device. The code in Figure 9 defines the machine as an i686 PC hardware virtualized machine booting from the main hard disk.

Figure 9: XML os section example.

```
<os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='hd'/>
</os>
```

5.1.2 The features Section

This section contains common hardware features such as ACPI. The code in Figure 10 defines that the virtual machine has ACPI, APCI, and PAE available.

Figure 10: XML features section example.

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

5.1.3 The devices Section

This is the most complicated section in the configuration file. Within this section all hardware interfaces, such as disks and network interfaces, are defined. An example of this section is listed in Figure 11.

Figure 11: XML devices section example.

```
<devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='cdrom'>
        <target dev='hdc' bus='ide'/>
        <readonly/>
    </disk>
    <disk type='file' device='disk'>
        <source file='/home/mvliet/sl53virtio/sl53-pvm.img'/>
        <target dev='vda' bus='virtio'/>
    </disk>
    <interface type='bridge'>
        <mac address='54:52:00:1c:6e:71'/>
        <source bridge='br0'/>
        <model type='virtio'/>
    </interface>
    <serial type='pty'>
        <target port='0'/>
    </serial>
    <console type='pty'>
        <target port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
</devices>
```

If a secondary disk image needs to be mounted within a virtual machine, ading a new `<disk>` target is all that is required. Similarly for adding a second network interface the `<interface>` target can be used.

6 Conclusion

Combining the power and flexability of the libvirt virtualization tools with the KVM virtual machine hypervisor allows for easy creation and control of virtual machines. The combination of the two technologies show a great deal of promise for the creation of new tools to ease the management and deployment of virtual machines.

7 Future Work

There are many features of libvirt not explored in this report that would be of use to the UVIC HEP group. In perticular, the remote management capabilities built into libvirt would be of imense use in the creation of new tools for the remote deployment of clusters of virtual machines. Included in the remote management capabilities is the ability to authenticate via x509 certificates and Kerberos authentication servers. Since much of the physics community already make use of x509 certificates, it would reduce the effort needed to interface with existing or new technologies.

Another area where libvirt may be useful is in the creation of a new backend for the Nimbus software currently in use. At the current time Nimbus only supports the Xen hypervisor. Since libvirt can make use of many different hypervisors while providing a common interface, it would provide a good intermediate interface for Nimbus.

8 Acknowledgments

I would like to thank everyone who I have had the pleasure of working with while at the UVic Grid Research group. Specificaly I would like to thank Dr. Randall Sobie for the opportunity of working in such an emerging filed of work, as well as Ian Gable for day to day help and guidance on projects. Additionaly I thank Patrick Armstrong, Chris Tooley, Ron Desmarais, as well as everyone on the KVM and libvirt mailing lists.

9 Glossary

VNC Virtual Network Computing. A networking protocol used to remotely interact with a graphical desktop environment.

Image A representation of a physical harddrive. Not to be confused with visual image or picture.

IP Internet Protocol. Commonly used to refer to an IP address.

SL Scientific Linux. A distrabution of Linux based on Red Hat Enterprise Linux.

DHCP Dynamic Host Configuration Protocol.

HEP High Energy Physics

API Application Programming Interface.

Hypervisor A layer of software allowing one to virtualize a node into one or more virtual machines.

Domain An instance of a virtual machine running ontop of a hypervisor

Node A physical Machine.

HVM Hardware Virtual Machine

KVM Kernel-based Virtual Machine. Open-source virtual machine manager intgrated into the Linux kernel.

PVM Para-Virtualized Machine

VM Virtual Machine, an instance of a machine(computer) running in software.

VMM Virtual Machine Monitor, used for managing virtual machines.

Xen Open-source virtual machine hypervisor.

References

[1] Nimbus http://workspace.globus.org/index.html

[2] libvirt - Virtualization API http://libvirt.org/

[3] KVM - Kernel-based Virtual Machine http://www.linux-kvm.org

[4] UVic Grid Research http://www.grid.phys.uvic.ca/index.html

[5] Xen - Virtual machine Hypervisor http://www.xen.org/

[6] Virt-Manager - GUI based virtual machine creator http://virt-manager.et.redhat.com/