kernels-in-kernels

kernels within kernels by Lee Elston

Version 1.0

The C-ALE (Cloud & Container Apprentice Linux Engineer) is a series of seminars held at existing conferences covering topics which are fundamental to a Linux professional in the Linux Cloud and Container field of computing.

This seminar will spend equal time on lecture and hands on labs at the end of each seminar which allow you to practice the material you've learned.

> This material makes the assumption that you have minimal experience with using Linux in general, and a basic understanding of general industry terms. The assumption is also made that you have access to your own computers upon which to practice this material.

More information can be found at **https://c-ale.org/**

Contents

Chapter 1

Kernels within Kernels

1.1 Kernels within Kernels

Kernel within Kernels

- An introduction to:
  - kvm
  - libvirtd
  - virsh

1.2 KVM

KVM: Kernel Virtual Machine

- An open source virtualization technology built into Linux
  - turns Linux into a hypervisor that allows virtual guests to be created
  - uses Linux for process management such as memory management, io, scheduling
  - included in kernel version 2.6.20 and later
- KVM uses Virtualization extensions in the processor chips
  - Intel: VT-x (also known as vmx)
  - AMD: AMD-V (also known as svm)

KVM: the modules

KVM is comprised of two kernel modules, a common module and a processor specific module:

- **kvm.ko**
  - common to all architectures
  - provides the virtualization infrastructure
  - accessible through the **kvm api**
- **kvm-intel.ko** or **kvm-amd.ko**
  - specific processor type module
  - kernel driver module for specific hardware and related extensions.

QEMU: Quick Emulator, the glue

- Is a Virtual Machine Monitor (VMM) or Hypervisor
  - emulates various CPU's
  - provides device models
  - executes as an emulator using dynamic binary translation
  - uses hardware virtualization extensions when coupled with KVM
- The combination of QEMU and KVM with hardware virtualization extensions can achieve near native performance within the VM's

kvm overview

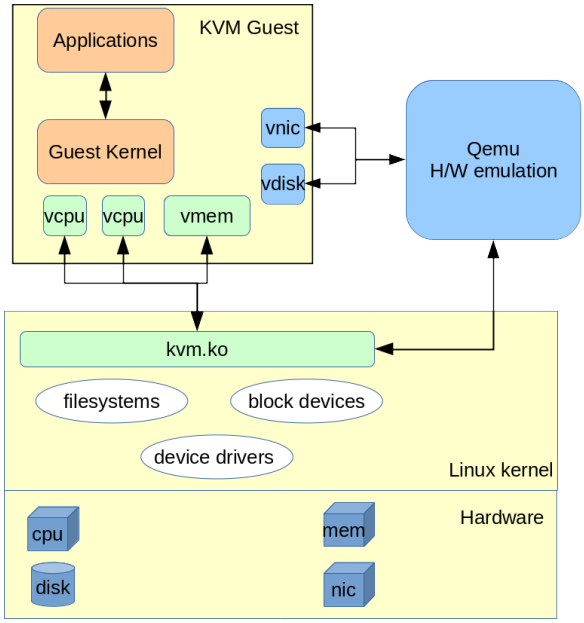

Figure 1.1: **Overview of kvm**

KVM/QEMU configuration

Configuring and starting a **VM** with **KVM/QEMU** can be done at the command line. There are a few steps:

- Confirm or load the appropriate **KVM** module
- Create an image file to be used as a disk
- Start **qemu-kvm** with a list of options:
  - disk image for storage
  - disk (or image) of a CD-ROM to install from
  - memory size directive
  - a boot command
- Install the operating system on the **VM**
- Stop the VM after the install is complete
- Restart the VM like before but select the disk image to boot from

1.3 libvirt

libvirt

libvirt is an `open-source` management toolkit for managing virtualization platforms.

The project home page is `https://libvirt.org/`

libvirt overview

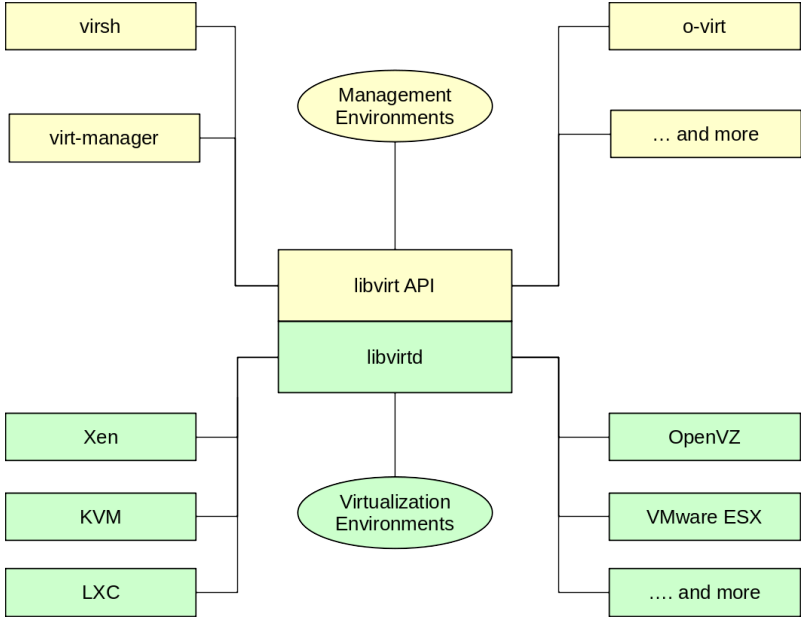

Figure 1.2: **Overview of libvirt**

libvirt-daemon overview

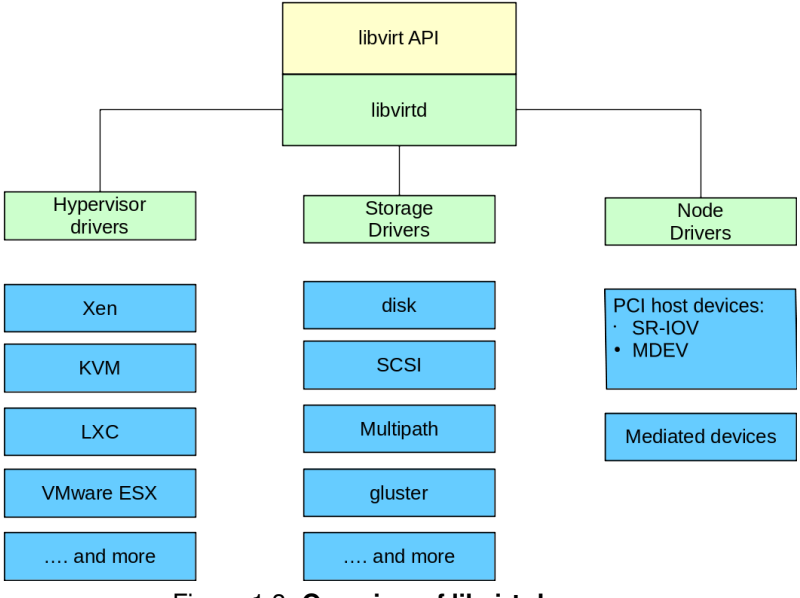

Figure 1.3: **Overview of libvirt-daemon**

libvirtiAPI overview

libvirtAPI overview

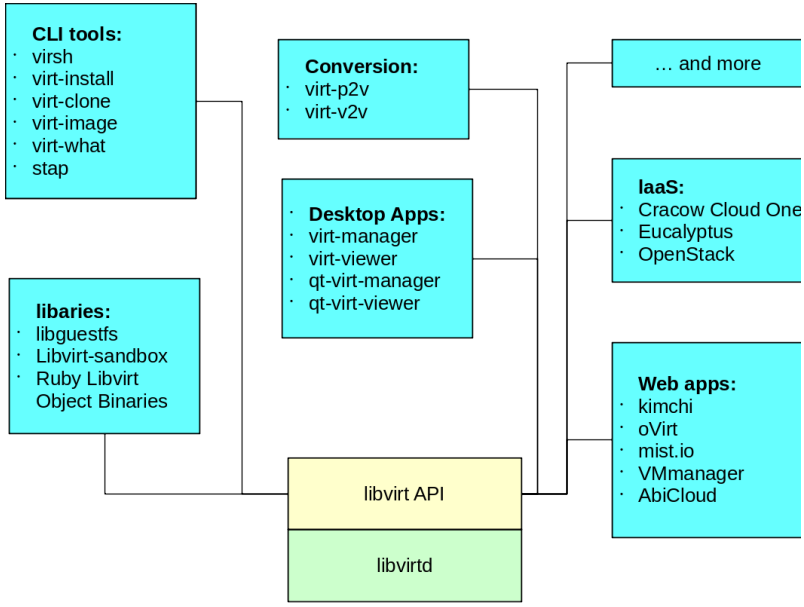

Figure 1.4: **Overview of libvirtAPI**

1.4 virsh

virsh: introduction

virsh is the main interface to manage services provided by **libvirt**. Some of the services that virsh can manage are:

- Guest Domains
- Device
- Virtual Network
- Virtual interface
- Storage Pool

The **virsh** documentation is located at https://libvirt.org/virshcmdref.html , the **man 1 virsh** page and the internal help text with the **virsh help** command.

virsh: Getting started

Before using **virsh** to manage our virtual machines a virtual machine configuration needs to be defined, of course this can be done through the command line, however, a short **XML** file can ease the process.

For our first VM we will use a preexisting disk, taking advantage of several defaults, we can create the following sections in our **html** file:

- main system: mem, name
- OS: memory, architecture,hypervisor, boot device
- devices: emulator,disk, console
- graphics:vnc
- and finally test the application.

virsh: define

In order for **libvirt** to recognize the virtual machine the xml configuration file needs to be registered with **libvirt**.

- validate the xml file to be used for input
- **define** the virtual machine based on the xml file
- verify the virtual machine is available

virsh define configuration file

The **virsh define** command creates an xml file that represents the virtual machine. During the define or creation process the input xml file is parsed and its contents override the default configuration parameters.
The resultant configuration file is located: `/etc/libvirt/qemu/name/xml`.
When libvirt initializes it looks in this directory for virtual machine configurations. If the xml file is not here libvirt cannot see or manage the virtual machine.
DO NOT edit the files in the /file/etc/libvirt/qemu/ directory.

virsh: starting and stopping

The commands to start and stop a registered domain are:

- **virsh start domain**
- **virsh shutdown domain**
- **virsh destroy domain**

virsh connecting to the VM

When connecting to a VM there are two common methods:

- text console
- graphical console

virsh undefine

The virsh command **undefine** removes the registered domain configuration.

Some examples of the **virsh undefine** command:

- delete the libvirt configuration for vm blue.

```
# virsh undefine blue
```

- delete the libvirt configuration for vm blue and remove all associated storage devices.

```
# virsh undefine blue --remove-all-storage
```

1.5 virt-manager

virt-manager

virt-manager project is the home to several common virtual machine management tools based on **libvirt**. Some of the tools are:

- **virt-manager** graphic tool for creating and managing virtual machines, KVM,Xen and LXC.
- **virt-viewer** graphic tool for connecting to the client VM's, uses VNC or SPICE protocols
- **virt-install** command driven installer for creating VM's

For additional information and documentation see
https://virt-manager.org

virt-install

The **virt-install** is an application that uses the **libvirt** interface, some of the features are:

- command line interface
- text or graphic installer interface
- local or network install
- an import only option

More information may be obtained from the associated man page or the project web site, **virt-install** is part of **virt-manager** project. Additional information can be found at https://virt-manager.org/.

virt-install –import

The option **import** to the **virt-install** command:

- skips the install process
- creates a default configuration for the new VM
- incorporates the command line options
- starts the new VM in a **virt-viewer** session

The resulting configuration may be viewed with the **virsh dumpxml name** command.

1.6 Labs

Exercise 1.1: Verify system is ready for KVM

Verify the CPU chip extensions are available and enabled to support virtualization. Confirm the required packages are available.

This exercise requires a 64 bit architecture system with a minimum of 4GiB if memory and 5GiB of free disk space. Internet connectivity is recommended but not critical.

It is common practice to run virtual machines as a non-root user, but in this exercise we will execute the commands as root.

Solution 1.1

1. Verify the cpu has virtualization support enabled.

   ```
   # grep -e svm -e vmx /proc/cpuinfo
   ```

   The output should have one of the flags highlighted, either **svm** or **vmx**. If no output was produced, verify in your **BIOS** that virtualization is supported.

2. Get the **knk-resources.tar** file. The presenter will have the information as to how to obtain the file.

3. Install required packages:

   - On **CentOS7**:

     ```
     # yum install qemu-kvm libvirt virt-manager virt-install virt-viewer
     ```

   - On **OpenSUSE**:

     ```
     # zypper install qem-kvm libvirt virt-manager virt-viewer bridge-utils
     ```

   - On **Ubuntu**:

     ```
     # apt-get install qemu-kvm libvirt-bin virtinst \
                       virt-manager libosinfo-bin virt-viewer
     ```

4. Verify and optionally start and enable the **libvirtd** service.

```
# systemctl status libvirtd
# systemctl start libvirtd
# systemctl enable libvirtd
# systemctl status libvirtd
```

5. If using **Ubuntu** or **Centos** please log in with **GNOME Xorg** not **GNOME Wayland**.
6. If using **openSUSE** you may have to add **export LIBVIRT_DEFAULT_URI=qemu:///system** to `/etc/bash.bashrc.local`.

   This will allow a regular user to administer all the virtual machines.

   ```
   # echo "export LIBVIRT_DEFAULT_URI=qemu:///system"  >> /etc/bash.bashrc.local
   ```

7. If challenged for the administrator password using **virsh** commands similar to the following screenshot.

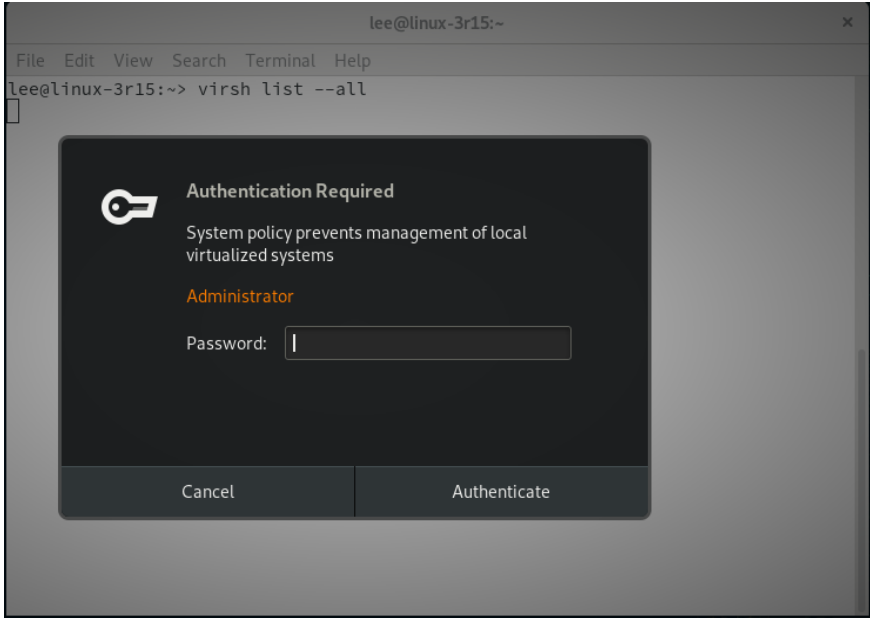

Figure 1.5: **libvirt authentication prompt**

Set the **unix_sock_group** variable in `/etc/libvirt/libvirtd.conf` to **libvirt**, it may be added to the bottom of the file like:

```
# echo 'unix_sock_group = "libvirt" ' >> /etc/libvirt/libvirtd.conf
```

Then add the additional supplemental group **libvirt** group to your user:

```
# usermod -a -G libvirt <username>
```

Exercise 1.2: Create and run a Virtual Machine based on an image

Often a virtual disk is used for distributing a pre-configured environment, for this exercise a pre-installed operating system image file is available in the **knk-resources.tar** file. The image file is called **tiny2.vmdk**.

This exercise will create a virtual machine using an **xml** file as input to **virsh**.

The parameters for the virtual machine are:

- VM name is **tiny2**
- 1 virtual CPU
- 512M memory
- the virtual disk is `/var/lib/libvirt/images/tiny2.vmdk`
- use **virt-viewer** to access the virtual machine

In this exercise a pre-installed disk `tiny2.vmdk` contains a operating system and an application to be made available. A **virsh define** command is to be used with a **xml** configuration file.

Solution 1.2

1. Create a directory to store the exercise resources in and extract the tarball.

   ```
   # mkdir  /var/tmp/knk
   # cd /var/tmp/knk
   # tar -xvf <download directory>/knk-resources.tar
   ```

2. Create a copy of the disk image and store it at `/var/lib/libvirt/images/tiny2.vmdk`

   ```
   # cp /var/tmp/knk/tiny2.vmdk  /var/lib/libvirt/images/
   ```

3. **Note:** As an added extra credit adventure, some distro's do not support writing to a **vmdk** image file. If **CentOS7** is being used for the host, the tiny2.vmdk must be converted to qcow2. The conversion is done with the **qemu-img** command.

   This will convert the file:

```
# cd /var/lib/libvirt/images/
# qemu-img convert -O qcow2 tiny2.vmdk tiny2.qcow2
```

You will have to remember to adjust the file name in the rest of the lab exercise.

4. Create an **xml** configuration file. (sample files are included in the `/var/tmp/knk/` directory)

   Note: There are three xml sample files in the resources directory, the names relate to the distro the host computer is running:

   - tiny2.xml-ubuntu
   - tiny2.xml-centos
   - tiny2.xml-suse

   Copy or rename the appopiate **xml** file to be **tiny2.xml**

   ```
   # cp tiny2.xml-ubuntu tiny2.xml
   ```

   The name and extension is not manditory but consistant naming helps with the exercise.

   Please feel free to compare the xml files as they illustrate minor changes in the distributions.

   An example of the configuratioin file is:

```
<domain type='kvm'>
<name>tiny2</name>
  <memory unit='KiB'>524288</memory>
  <os>
    <type arch='x86_64' machine='pc-i440fx-2.11'>hvm</type>
  </os>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/var/lib/libvirt/images/tiny2.qcow2'/>
      <target dev='hda' bus='ide'/>
      <address type='drive' controller='0' bus='0' target='0' unit='0'/>
```

```
    </disk>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <channel type='spicevmc'>
      <target type='virtio' name='com.redhat.spice.0'/>
      <address type='virtio-serial' controller='0' bus='0' port='1'/>
    </channel>
     <graphics type='spice' autoport='yes'>
      <listen type='address'/>
      <image compression='off'/>
    </graphics>
  </devices>
</domain>
```

5. define the virtual machine

```
# virsh define tiny2.xml
```

6. verify the configuration file

```
# virsh dumpxml tiny2
```

7. start the virtual machine

```
# virsh start tiny2
```

8. connect to the virtual machine

```
# virt-viewer tiny2
```

9. exit the virt-viewer and verify the state of the VM

```
# virsh list --all
```


10. try a graceful shutdown of the VM, success or failure will depend on the OS

```
# virsh shutdown tiny2
```

11. issue a forced shutdown of the VM if the graceful shutdown did not work

```
# virsh destroy tiny2
```

Exercise 1.3: Create a Virtual Machine using virt-install

In the previous exercise **virsh** was used to create the configuration using an xml file as input.

The object of this exercise is to simplify the install procedure by using the **virt-install** tool.

One feature of **virt-install** is the automatic configuration of a default network connection. This feature can be overridden if desired. In this exercise the default network connection will be used.

The parameters for the virtual machine are:

- name is **tiny3**
- 1 virtual CPU
- 512M memory
- the virtual disk is `/var/lib/libvirt/images/tiny3.vmdk`, a copy of the previously used **tiny2.vmdk**
- use a network connection called **default**
- use **virt-viewer** to access the virtual machine

Once the **VM** has been created and tested that it runs, compare the active configuration that **libvirt** has stored. The configuration created by **virt-install** will have many more default options enabled, including an active network connection NAT'd to the default adapter on the host computer.

Solution 1.3

1. Create a copy of **tiny2.vmdk** and place it in `/var/lib/libvirt/images/tiny3.vmdk`

   ```
   # cp /var/lib/libvirt/images/tiny2.vmdk /var/lib/libvirt/images/tiny3.vmdk
   ```

2. Confirm or create a user called **dnsmasq**, it is used by libvirt.

   ```
   # grep dnsmasq /etc/passwd
   ```

   if the user **dnsmasq** does not exist, create it.

   ```
   # useradd dnsmasq
   ```

3. Verify a virtual network called **default** exists and is set to autostart.

   ```
   # virsh net-list --all
    Name                 State      Autostart     Persistent
   ----------------------------------------------------------
   ```

   If the **default** network does not exist, use **virsh** to define the network from the xml file `default-net.xml`.
   The contents of the `default-net.xml` are:

   ```
   <network>
     <name>default</name>
     <uuid>96ea6db6-cfb9-48fa-ad6a-bd21ab81c0d3</uuid>
     <forward mode='nat'>
       <nat>
         <port start='1024' end='65535'/>
       </nat>
     </forward>
     <bridge name='virbr0' stp='on' delay='0'/>
     <mac address='52:54:00:95:cb:bd'/>
     <domain name='default'/>
     <ip address='192.168.122.1' netmask='255.255.255.0'>
       <dhcp>
         <range start='192.168.122.128' end='192.168.122.254'/>
   ```

```
    </dhcp>
  </ip>
</network>
```

Based on the default-net.xml file, the command to define the network is:

```
# virsh net-define default-net.xml
Network default defined from default-net.xml
```

Then set the network to autostart:

```
# virsh net-autostart default
Network default marked as autostarted
```

Verify the network looks good.

```
# virsh net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              inactive   yes           yes
```

Start the new network.

```
# virsh net-start default
```

4. Use the virt-install command to **import** the existing virtual machine's disk into a new configuration.

```
# virt-install \
              --name tiny3 \
              --memory 512 \
              --disk /var/lib/libvirt/images/tiny3.vmdk \
              --os-type linux --os-variant generic  \
              --import
```

5. Examine the active configurations stored by **libvirt**.

```
# virsh dumpxml tiny3 > /tmp/tiny3.xml.dump
# virsh dumpxml tiny2 > /tmp/tiny2.xml.dump
```

Use commands like diff and ls to see the difference in the xml.dump files.

Exercise 1.4: Create a VM and install from a CD-rom image

This exercise will demonstrate installation of an operating system from a CD-rom image using **virt-install**. The CD-rom image is available in resources tar file installed earlier. Since installation of operating systems can take a long time and require access to download updates or repository data a small self contained OS was chosen for this exercise.

Note: During the initial start up, a default of a text interface has been added. This avoids some interesting issues with the mouse during and after installation of Tiny Core Linux. This change has bee added specifically for the confrence environment. If a graphics interface is desire, the **TinyCore-Plus** installation media is recomended.

The parameters for the virtual machine are:

- name is **tinycore**
- 1 virtual CPU
- 512M memory
- 100M new virtual disk image
- the virtual disk is `/var/lib/libvirt/images/tinycore.qcow2`
- the virtual cd is `/var/tmp/knk/k-n-k-3.iso`
- use **virt-viewer** to access the virtual machine

1. Copy the CD-rom image to `/var/lib/libvirt/images` directory.

   ```
   # cp /var/tmp/knk/k-n-k-3.iso /var/lib/libvirt/images/k-n-k-3.iso
   ```

2. Boot from the CD-rom using **virt-install** using the options specified below:

```
$ virt-install  --name tinycore --memory 512 --vcpus 1 \
                --cdrom /var/lib/libvirt/images/k-n-k-3.iso  \
                --disk /var/lib/libvirt/images/tinycore.qcow2,size=0.1 \
                --os-type linux --os-variant generic --boot useserial=on
```

A **virt-viewer** window should pop open.

3. When the initial Core Plus screen appears, press the **enter** key.

   After the boot messages finish, the user friendly Tiny Core command prompt will be visible.

4. To launch the installer at the prompt type:

   ```
   $ sudo tc-install.sh
   ```

5. Please fill in the installation choices as listed below:

   - Core Installation

     ```
     c for cdrom, enter to continue
     ```

   - Install type

     ```
     f for frugal, enter to continue
     ```

   - Select Target for Installation

     ```
     1 for whole disk
     ```

   - Bootloader

     ```
     y for yes for bootloader
     ```

   - Install Extensions

     ```
     TCE/CDE Directory is left blank, press enter
     ```

   - Disk Formatting Options

     ```
     Select 3 for ext4
     ```

   - Boot Options

     ```
     No additional boot options, press enter
     ```

- Last chance before destroying a data on sda

  ```
  yes, start the install
  ```

The installation should complete in a few seconds.

6. When the installation is complete type in:

   ```
   sudo reboot
   ```

Exercise 1.5: Create a Virtual Machine and install with a text console

Note: This is an optional lab not intended for completion at the conference but rather a take home exercise. You will need to download an install DVD. This example uses a **Debian 9.5.0** DVD as its source and is approximately 3.4G. The images can be found at:

`https://cdimage.debian.org/debian-cd/current/amd64/iso-dvd/`

Note: This installation **will** access the Internet to communicate with the **Debian** repository servers. To avoid the outbound connection and speed up the install, shutdown the network access on the host system before starting the installation.

The parameters for the virtual machine are:

- name is **test-loc**
- 1 virtual CPU
- 512 memory
- 1.5G disk
- the virtual disk is going to be manually created, the suggested location is `/var/lib/libvirt/images/deb9.qcow2`
- the image used as the DVD can reside anyplace there is space
- use **virt-viewer** to access the virtual machine

- the install and finished installation will be text only

Solution 1.5

1. Create a disk image with a size of **1.5GiB** and a type of **qcow2**.

```
# qemu-img create -f qcow2 /var/lib/libvirt/images/deb9.qcow2 1.5g
```

2. Use **virt-install** to create the **VM** and start the install sequence. Answer the **OS** installation questions for a minimal configuration.

```
virt-install \
       --name deb9 \
       --memory 2048 \
       --disk /var/lib/libvirt/images/deb9.qcow2 --vcpus 2 \
       --os-type linux \
       --os-variant debian9 \
       --boot useserial=on  \
       --cdrom /var/ftp/pub/iso/debian-9.5.0-amd64-DVD-1.iso
```

3. When the installation is completed, a **virt-viewer** window should open up with the login prompt for the new **VM**.

Exercise 1.6: Challenge exercise using virt-manager, a KVM management GUI

In the progression of configuration tools for KVM from **xml** files to **virt-install** utility and now to feature packed GUI, **virt-manager**. Everything done in our exercises can be done with **virt-manager**. If the configuration was done using libvirt (all of ours were) then you should be able to manage all of the previously created VM's.

The challenge in this exercise is to repeat the exercise steps using **virt-manager**.