libvirt

Libvirt is collection of software that provides a convenient way to manage virtual machines and other virtualization functionality, such as storage and network interface management. These software pieces include a long term stable C API, a daemon (libvirtd), and a command line utility (virsh). A primary goal of libvirt is to provide a single way to manage multiple different virtualization providers/hypervisors, such as the **KVM/QEMU**, **Xen**, **LXC**, **OpenVZ (http://openvz.org)** or **VirtualBox hypervisors** (**among others (http://libvirt.org/drivers.html)**).

Related articles

Category:Hypervisors

PCI passthrough via OVMF

Some of the major libvirt features are:

- **VM management**: Various domain lifecycle operations such as start, stop, pause, save, restore, and migrate. Hotplug operations for many device types including disk and network interfaces, memory, and CPUs.
- **Remote machine support**: All libvirt functionality is accessible on any machine running the libvirt daemon, including remote machines. A variety of network transports are supported for connecting remotely, with the simplest being SSH, which requires no extra explicit configuration.

- **Storage management**: Any host running the libvirt daemon can be used to manage various types of storage: create file images of various formats (qcow2, vmdk, raw, ...), mount NFS shares, enumerate existing LVM volume groups, create new LVM volume groups and logical volumes, partition raw disk devices, mount iSCSI shares, and much more.
- **Network interface management**: Any host running the libvirt daemon can be used to manage physical and logical network interfaces. Enumerate existing interfaces, as well as configure (and create) interfaces, bridges, vlans, and bond devices.
- **Virtual NAT and Route based networking**: Any host running the libvirt daemon can manage and create virtual networks. Libvirt virtual networks use firewall rules to act as a router, providing VMs transparent access to the host machines network.

Contents

Installation

Because of its daemon/client architecture, libvirt needs only be installed on the machine which will host the virtualized system. Note that the server and client can be the same physical machine.

Server

Install the **libvirt (https://www.archlinux.org/packages/?name=libvirt)** package, as well as at least one hypervisor:

- The **libvirt KVM/QEMU driver (http://libvirt.org/drvqemu.html)** is the primary *libvirt* driver and if **KVM is enabled**, fully virtualized, hardware accelerated guests will be available. See the **QEMU** article for more information.

- Other **supported hypervisors (http://libvirt.org/drivers.html)** include **LXC**, **VirtualBox** and **Xen**. See the respective articles for installation instructions. With respect to `libvirtd` installation note:
  - The **libvirt LXC driver (http://libvirt.org/drvlxc.html)** has no dependency on the **LXC** userspace tools provided by **lxc (https://www.archlinux.org/packages/?name=lxc)**, therefore there is no need to install the package if planning on using the driver.
  - **Xen** support is available, but not by default. You need to use the **ABS** to modify **libvirt (https://www.archlinux.org/packages/?name=libvirt)**'s **PKGBUILD**

and build it without the `--without-xen` option. As VirtualBox in turn has no planned stable support for Xen, you might as well replace it with `--without-vbox`.

For network connectivity, install:

- **ebtables (https://www.archlinux.org/packages/?name=ebtables) and dnsmasq (https://www.archlinux.org/packages/?name=dnsmasq)** for the **default (http://wiki.libvirt.org/page/VirtualNetworking#The_default_configuration)** NAT/DHCP networking.
- **bridge-utils (https://www.archlinux.org/packages/?name=bridge-utils)** for bridged networking.
- **openbsd-netcat (https://www.archlinux.org/packages/?name=openbsd-netcat)** for remote management over **SSH**.

Client

The client is the user interface that will be used to manage and access the virtual machines.

- *virsh* is a command line program for managing and configuring domains; it is included in the **libvirt (https://www.archlinux.org/packages/?name=libvirt)** package.
- **virt-manager (https://www.archlinux.org/packages/?name=virt-manager)** is a graphical user interface for managing virtual machines.
- **virt-viewer (https://www.archlinux.org/packages/?name=virt-viewer)** is a lightweight interface for interacting with the graphical display of virtualized guest OS.
- **gnome-boxes (https://www.archlinux.org/packages/?name=gnome-boxes)** is a simple GNOME 3 application to access remote or virtual systems.

- `virt-manager-qt5 (https://aur.archlinux.org/packages/virt-manager-qt5/)`[AUR]
- `libvirt-sandbox (https://aur.archlinux.org/packages/libvirt-sandbox/)`[AUR] is an application sandbox toolkit.

A list of libvirt-compatible software can be found **here (http://libvirt.org/apps.html)**.

Configuration

For ***system***-level administration (i.e. global settings and image-*volume* location), libvirt minimally requires **setting up authorization**, and **starting the daemon**.

Note: For user-***session*** administration, daemon setup and configuration is *not* required; authorization, however, is limited to local abilities; the front-end will launch a local instance of the **libvirtd** daemon.

Set up authentication

From **libvirt: Connection authentication (http://libvirt.org/auth.html#ACL_server_config)**:

> The libvirt daemon allows the administrator to choose the authentication mechanisms used for client connections on each network socket independently. This is primarily controlled via the libvirt daemon master config file in `/etc/libvirt/libvirtd.conf`. Each of the

libvirt sockets can have its authentication mechanism configured independently. There is currently a choice of `none`, `polkit` and `sasl`.

Because **libvirt (https://www.archlinux.org/packages/?name=libvirt)** pulls **polkit (https://www.archlinux.org/packages/?name=polkit)** as a dependency during installation, **polkit** is used as the default value for the `unix_sock_auth` parameter (**source (http://libvirt.org/auth.html#ACL_server_polkit)**). **File-based permissions** remain nevertheless available.

Using polkit

Note: A system reboot may be required before authenticating with `polkit` works correctly.

The *libvirt* daemon provides two **polkit actions** in `/usr/share/polkit-1/actions/org.libvirt.unix.policy`:

- `org.libvirt.unix.manage` for full management access (RW daemon socket), and
- `org.libvirt.unix.monitor` for monitoring only access (read-only socket).

The default policy for the RW daemon socket will require to authenticate as an admin. This is akin to **sudo** auth, but does not require that the client application ultimately run as root. Default policy will still allow any application to connect to the RO socket.

Arch defaults to consider anybody in the `wheel` group as an administrator: this is defined in `/etc/polkit-1/rules.d/50-default.rules` (see **Polkit#Administrator identities**). Therefore there is no need to create a new group and rule file **if your user is a member of the `wheel` group**: upon connection to the RW socket (e.g. via **virt-manager (https://www.archlinux.org/packages/?name=virt-manager)**) you will be prompted for your user's password.

Note: Prompting for a password relies on the presence of an **authentication agent** on the system. Console users may face an issue with the default `pkttyagent` agent which may or may not work properly.

Tip: If you want to configure passwordless authentication, see **Polkit#Bypass password prompt**.

As of libvirt 1.2.16 (commit:**[1] (http://libvirt.org/git/?p=libvirt.git;a=commit;h=e94979e901517af9fdde358d7b7c92cc055dd50c)**), members of the `libvirt` group have passwordless access to the RW daemon socket by default. The easiest way to ensure your user has access is to ensure the libvirt group exists and they are a member of it. If you wish to change the group authorized to access the RW daemon socket to be the kvm group, create the following file:

/etc/polkit-1/rules.d/50-libvirt.rules

```
/* Allow users in kvm group to manage the libvirt
daemon without authentication */
polkit.addRule(function(action, subject) {
    if (action.id == "org.libvirt.unix.manage" &&
        subject.isInGroup("kvm")) {
            return polkit.Result.YES;
```

```
    }
});
```

Then **add yourself** to the `kvm` group and relogin. Replace *kvm* with any group of your preference just make sure it exists and that your user is a member of it (see **Users and groups** for more information).

Do not forget to relogin for group changes to take effect.

Authenticate with file-based permissions

To define file-based permissions for users in the *libvirt* group to manage virtual machines, uncomment and define:

/etc/libvirt/libvirtd.conf

```
#unix_sock_group = "libvirt"
#unix_sock_ro_perms = "0777"  # set to 0770 to deny non-group libvirt users
#unix_sock_rw_perms = "0770"
#auth_unix_ro = "none"
#auth_unix_rw = "none"
```

While some guides mention changed permissions of certain libvirt directories to ease management, keep in mind permissions are lost on package update. To edit these system directories, root user is expected.

Daemon

Start both `libvirtd.service` and `virtlogd.service`. Optionally **enable** `libvirtd.service`. There is no need to enable `virtlogd.service`, since `libvirtd.service`, when enabled, also enables the `virtlogd.socket` and `virtlockd.socket` **units**.

Unencrypt TCP/IP sockets

Warning: This method is used to help remote domain, connection speed for trusted networks. This is the least secure connection method. This should *only* be used for testing or use over a secure, private, and trusted network. SASL is not enabled here, so all TCP traffic is *cleartext*. For real world use *always* enable SASL.

Edit `/etc/libvirt/libvirtd.conf`:

```
/etc/libvirt/libvirtd.conf
```

```
listen_tls = 0
listen_tcp = 1
auth_tcp="none"
```

It is also necessary to start the server in listening mode by editing `/etc/conf.d/libvirtd`:

```
/etc/conf.d/libvirtd
```

```
LIBVIRTD_ARGS="--listen"
```

Access virtual machines using their hostnames

For host access to guests on non-isolated, bridged networks, enable the `libvirt` NSS module provided by **libvirt (https://www.archlinux.org/packages/?name=libvirt)**.

Edit `/etc/nsswitch.conf`:

```
/etc/nsswitch.conf
```

```
hosts: files libvirt dns myhostname
```

Note: While commands such as `ping` and `ssh` should work with virtual machine hostnames, commands such as `host` and `nslookup` may fail or produce unexpected results because they rely on DNS. Use `getent hosts <vm-hostname>` instead.

Test

To test if libvirt is working properly on a *system* level:

```
$ virsh -c qemu:///system
```

To test if libvirt is working properly for a user-*session*:

```
$ virsh -c qemu:///session
```

Management

Libvirt management is done mostly with three tools: **virt-manager (https://www.archlinux.org/packages/?name=virt-manager)** (GUI), `virsh`, and `guestfish` (which is part of **libguestfs (https://aur.archlinux.org/packages/libguestfs/)**[AUR]).

virsh

The virsh program is for managing guest *domains* (virtual machines) and works well for scripting, virtualization administration. Though most virsh commands require root privileges to run due to the communication channels used to talk to the hypervisor, typical management, creation, and running of domains (like that done with VirtualBox) can be done as a regular user.

Virsh includes an interactive terminal that can be entered if no commands are passed (options are allowed though): `virsh`. The interactive terminal has support for tab completion.

From the command line:

```
$ virsh [option] <command> [argument]...
```

From the interactive terminal:

```
virsh # <command> [argument]...
```

Help is available:

```
$ virsh help [option*] or [group-keyword*]
```

Storage pools

A pool is a location where storage *volumes* can be kept. What libvirt defines as *volumes* others may define as "virtual disks" or "virtual machine images". Pool locations may be a directory, a network filesystem, or partition (this includes a **LVM**). Pools can be toggled active or inactive and allocated for space.

On the *system*-level, `/var/lib/libvirt/images/` will be activated by default; on a user-*session*, `virt-manager` creates `$HOME/VirtualMachines`.

Print active and inactive storage pools:

```
$ virsh pool-list --all
```

Create a new pool using virsh

If wanted to *add* a storage pool, here are examples of the command form, adding a directory, and adding a LVM volume:

```
$ virsh pool-define-as name type [source-host] [source-path] [source-dev] [source-name] [<target>] [--source-format format]
$ virsh pool-define-as poolname dir - - - - /home/username/.local/libvirt/images
$ virsh pool-define-as poolname fs - -  /dev/vg0/images - mntpoint
```

The above command defines the information for the pool, to build it:

```
$ virsh pool-build     poolname
$ virsh pool-start     poolname
$ virsh pool-autostart poolname
```

To remove it:

```
$ virsh pool-undefine  poolname
```

Tip: For LVM storage pools:

- It is a good practice to dedicate a volume group to the storage pool only.
- Choose a LVM volume group that differs from the pool name, otherwise when the storage pool is deleted the LVM group will be too.

Create a new pool using virt-manager

First, connect to a hypervisor (e.g. QEMU/KVM *system*, or user-*session*). Then, right-click on a connection and select *Details*; select the *Storage* tab, push the + button on the lower-left, and follow the wizard.

Storage volumes

Once the pool has been created, volumes can be created inside the pool. *If building a new domain (virtual machine), this step can be skipped as a volume can be created in the domain creation process.*

Create a new volume with virsh

Create volume, list volumes, resize, and delete:

```
$ virsh vol-create-as      poolname volumename 10GiB --format aw|bochs|raw|qcow|qcow2|vmdk
$ virsh vol-upload  --pool poolname volumename volumepath
$ virsh vol-list           poolname
$ virsh vol-resize  --pool poolname volumename 12GiB
$ virsh vol-delete  --pool poolname volumename
$ virsh vol-dumpxml --pool poolname volumename  # for details.
```

virt-manager backing store type bug

On newer versions of `virt-manager` you can now specify a backing store to use when creating a new disk. This is very useful, in that you can have new domains be based on base images saving you both time and disk space when provisioning new virtual systems. There is a bug (**https://bugzilla.redhat.com/show_bug.cgi?id=1235406**) in the current version of `virt-manager` which causes `virt-manager` to choose the wrong type of the backing image

in the case where the backing image is a `qcow2` type. In this case, it will errantly pick the backing type as `raw`. This will cause the new image to be unable to read from the backing store, and effectively remove the utility of having a backing store at all.

There is a workaround for this issue. `qemu-img` has long been able to do this operation directly. If you wish to have a backing store for your new domain before this bug is fixed, you may use the following command.

```
$ qemu-img create -f qcow2 -o backing_file=<path to backing image>,backing_fmt=qcow2 <disk name> <disk size>
```

Then you can use this image as the base for your new domain and it will use the backing store as a COW volume saving you time and disk space.

Domains

Virtual machines are called *domains*. If working from the command line, use `virsh` to list, create, pause, shutdown domains, etc. `virt-viewer` can be used to view domains started with `virsh`. Creation of domains is typically done either graphically with `virt-manager` or with `virt-install` (a command line program installed as part of the **virt-install (https://www.archlinux.org/packages/?name=virt-install)** package).

Creating a new domain typically involves using some installation media, such as an `.iso` from the storage pool or an optical drive.

Print active and inactive domains:

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

Note: SELinux has a built-in exemption for libvirt that allows volumes in `/var/lib/libvirt/images/` to be accessed. If using SELinux and there are issues with the volumes, ensure that volumes are in that directory, or ensure that other storage pools are correctly labeled.

Create a new domain using virt-install

For an extremely detailed domain (virtual machine) setup, it is easier to **#Create a new domain using virt-manager**. However, basics can easily be done with `virt-install` and still run quite well. Minimum specifications are `--name`, `--memory`, guest storage (`--disk`, `--filesystem`, or `--nodisks`), and an install method (generally an `.iso` or CD). See **`virt-install(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/virt-install.1)`** for more details and information about unlisted options.

Arch Linux install (two GiB, qcow2 format volume create; user-networking):

```
$ virt-install  \
  --name arch-linux_testing \
  --memory 1024             \
  --vcpus=2,maxvcpus=4      \
  --cpu host                \
  --cdrom $HOME/Downloads/arch-linux_install.iso \
  --disk size=2,format=qcow2  \
```

```
  --network user           \
  --virt-type kvm
```

Fedora testing (Xen hypervisor, non-default pool, do not originally view):

```
$ virt-install  \
  --connect xen:///     \
  --name fedora-testing \
  --memory 2048         \
  --vcpus=2             \
  --cpu=host            \
  --cdrom /tmp/fedora20_x84-64.iso      \
  --os-type=linux --os-variant=fedora20 \
  --disk pool=testing,size=4            \
  --network bridge=br0                  \
  --graphics=vnc                        \
  --noautoconsole
$ virt-viewer --connect xen:/// fedora-testing
```

Windows:

```
$ virt-install \
  --name=windows7           \
  --memory 2048             \
  --cdrom /dev/sr0          \
  --os-variant=win7         \
  --disk /mnt/storage/domains/windows7.qcow2,size=20GiB \
  --network network=vm-net  \
  --graphics spice
```

Tip: Run `osinfo-query --fields=name,short-id,version os` to get argument for `--os-variant`; this will help define some specifications for the domain. However, `--memory` and `--disk` will need to be entered; one can look within the appropriate `/usr/share/libosinfo/db/oses/os.xml` if needing these specifications. After installing, it will likely be preferable to install the **Spice Guest Tools (http://www.spice-space.org/downl**

oad.html) that include the VirtIO drivers (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Host_Configuration_and_Guest_Installation_Guide/form-Virtualization_Host_Configuration_and_Guest_Installation_Guide-Para_virtualized_drivers-Mounting_the_image_with_virt_manager.html). For a Windows VirtIO network driver there is also `virtio-win (https://aur.archlinux.org/packages/virtio-win/)`[AUR]. These drivers are referenced by a `<model type='virtio' />` in the guest's `.xml` configuration section for the device. A bit more information can also be found on the QEMU article.

Import existing volume:

```
$ virt-install  \
  --name demo  \
  --memory 512 \
  --disk /home/user/VMs/mydisk.img \
  --import
```

Create a new domain using virt-manager

First, connect to the hypervisor (e.g. QEMU/KVM *system* or user *session*), right click on a connection and select *New*, and follow the wizard.

- On the *fourth step*, de-selecting *Allocate entire disk now* will make setup quicker and can save disk space in the interum; *however*, it may cause volume fragmentation over time.
- On the *fifth step*, open *Advanced options* and make sure that *Virt Type* is set to *kvm* (this is usually the preferred method). If additional hardware setup is required, select the

Customize configuration before install option.

Manage a domain

Start a domain:

```
$ virsh start domain
$ virt-viewer --connect qemu:///session domain
```

Gracefully attempt to shutdown a domain; force off a domain:

```
$ virsh shutdown domain
$ virsh destroy  domain
```

Autostart domain on libvirtd start:

```
$ virsh autostart domain
$ virsh autostart domain --disable
```

Shutdown domain on host shutdown:

Running domains can be automatically suspended/shutdown at host shutdown using the `libvirt-guests.service` systemd service. This same service will resume/startup the suspended/shutdown domain automatically at host startup. Read `/etc/conf.d/libvirt-guests` for service options.

Edit a domain's XML configuration:

```
$ virsh edit domain
```

Note: Virtual Machines started directly by QEMU are not managable by libvirt tools.

Networks

A **decent overview of libvirt networking (https://jamielinux.com/docs/libvirt-networking-handbook/)**.

By default, when the `libvirtd` systemd service is started, a NAT bridge is created called *default* to allow external network connectivity (IPv4-only). For other network connectivity needs, four network types exist that can be created to connect a domain to:

- bridge — a virtual device; shares data directly with a physical interface. Use this if the host has *static* networking, it does not need to connect other domains, the domain requires full inbound and outbound trafficking, and the domain is running on a *system*-level. See **Network bridge** on how to add a bridge additional to the default one. After creation, it needs to be specified in the respective guest's `.xml` configuration file.
- network — a virtual network; has ability to share with other domains. Use a virtual network if the host has *dynamic* networking (e.g. NetworkManager), or using wireless.
- macvtap — connect directly to a host physical interface.
- user — local ability networking. Use this only for a user *session*.

`virsh` has the ability to create networking with numerous options for most users, however, it is easier to create network connectivity with a graphic user interface (like `virt-manager`), or to do so on **creation with virt-install**.

Note: libvirt handles DHCP and DNS with **dnsmasq (https://www.archlinux.org/packages/?name=dnsmasq)**, launching a separate instance for every virtual network. It also adds iptables rules for proper routing, and enables the `ip_forward` kernel parameter. This also means that having dnsmasq running on the host system is not necessary to support libvirt requirements (and could interfere with libvirt dnsmasq instances).

IPv6

When adding an IPv6 address through any of the configuration tools, you will likely receive the following error:

```
Check the host setup: enabling IPv6 forwarding with RA routes without accept_ra set to 2 is likely to cause routes loss. Interfaces to look at: eth0
```

Fix this by creating the following file (replace *`eth0`* with the name of your physical interface). Reboot your machine afterwards.

/etc/sysctl.d/libvirt-bridge.conf

```
net.ipv6.conf.eth0.accept_ra = 2
```

Snapshots

Snapshots take the disk, memory, and device state of a domain at a point-of-time, and save it for future use. They have many uses, from saving a "clean" copy of an OS image to saving a domain's state before a potentially destructive operation. Snapshots are identified with a unique name.

Snapshots are saved within the volume itself and the volume must be the format: qcow2 or raw. Snapshots use deltas so they have the potentiality to not take much space.

Create a snapshot

Once a snapshot is taken it is saved as a new block device and the original snapshot is taken offline. Snapshots can be chosen from and also merged into another (even without shutting down the domain).

Print a running domain's volumes (running domains can be printed with `virsh list`):

```
# virsh domblklist domain
```

```
Target     Source
------------------------------------------------
vda        /vms/domain.img
```

To see a volume's physical properties:

```
# qemu-img info /vms/domain.img
```

```
image: /vms/domain.img
file format: qcow2
virtual size: 50G (53687091200 bytes)
disk size: 2.1G
cluster_size: 65536
```

Create a disk-only snapshot (the option `--atomic` will prevent the volume from being modified if snapshot creation fails):

```
# virsh snapshot-create-as domain snapshot1 --disk-only --atomic
```

List snapshots:

```
# virsh snapshot-list domain
```

```
Name                 Creation Time             State
------------------------------------------------------------
snapshot1            2012-10-21 17:12:57 -0700 disk-snapshot
```

One can they copy the original image with `cp --sparse=true` or `rsync -S` and then merge the the original back into snapshot:

```
# virsh blockpull --domain domain --path /vms/domain.snapshot1
```

`domain.snapshot1` becomes a new volume. After this is done the original volume (`domain.img` and snapshot metadata can be deleted. The `virsh blockcommit` would work opposite to `blockpull` but it seems to be currently under development (including

`snapshot-revert feature`, scheduled to be released sometime next year.

Other management

Connect to non-default hypervisor:

```
$ virsh --connect xen:///
virsh # uri
xen:///
```

Connect to the QEMU hypervisor over SSH; and the same with logging:

```
$ virsh --connect qemu+ssh://username@host/system
$ LIBVIRT_DEBUG=1 virsh --connect qemu+ssh://username@host/system
```

Connect a graphic console over SSH:

```
$ virt-viewer  --connect qemu+ssh://username@host/system domain
$ virt-manager --connect qemu+ssh://username@host/system domain
```

Note: If you are having problems connecting to a remote RHEL server (or anything other than Arch, really), try the two workarounds mentioned in **FS#30748 (https://bugs.archlinux.org/task/30748)** and **FS#22068 (https://bugs.archlinux.org/task/22068)**.

Connect to the VirtualBox hypervisor (*VirtualBox support in libvirt is not stable yet and may cause libvirtd to crash*):

```
$ virsh --connect vbox:///system
```

Network configurations:

```
$ virsh -c qemu:///system net-list --all
$ virsh -c qemu:///system net-dumpxml default
```

Python connectivity code

The **libvirt-python (https://www.archlinux.org/packages/?name=libvirt-python)** package provides a **python2 (https://www.archlinux.org/packages/?name=python2)** API in `/usr/lib/python2.7/site-packages/libvirt.py`.

General examples are given in `/usr/share/doc/libvirt-python-your_libvirt_version/examples/`

Unofficial example using **qemu (https://www.archlinux.org/packages/?name=qemu)** and **openssh (https://www.archlinux.org/packages/?name=openssh)**:

```
#! /usr/bin/env python2
# -*- coding: utf-8 -*-
import socket
import sys
import libvirt
if (__name__ == "__main__"):
   conn = libvirt.open("qemu+ssh://xxx/system")
   print "Trying to find node on xxx"
   domains = conn.listDomainsID()
   for domainID in domains:
```

```
        domConnect = conn.lookupByID(domainID)
        if domConnect.name() == 'xxx-node':
            print "Found shared node on xxx with ID " + str(domainID)
            domServ = domConnect
            break
```

UEFI Support

Libvirt can support UEFI virtual machines through QEMU and **OVMF (https://github.com/tianocore/edk2)**.

Install the **`ovmf (https://www.archlinux.org/packages/?name=ovmf)`** package.

Add the following to `/etc/libvirt/qemu.conf`.

```
/etc/libvirt/qemu.conf
```

```
nvram = [
    "/usr/share/ovmf/x64/OVMF_CODE.fd:/usr/share/ovmf/x64/OVMF_VARS.fd"
]
```

Note: If you created a UEFI virtual machine before January 12th 2018, you must first update `/etc/libvirt/qemu.conf` like shown above and then run edit the machines' config (`virsh edit <domain>` and update the value of the `<loader>` element)

Restart `libvirtd`.

Now you are ready to create a UEFI virtual machine. Create a new virtual machine through `virt-manager (https://www.archlinux.org/packages/?name=virt-manager)`. When you get to the final page of the 'New VM' wizard, do the following:

- Click 'Customize before install', then select 'Finish'
- On the 'Overview' screen, Change the 'Firmware' field to select the 'UEFI x86_64' option.
- Click 'Begin Installation'
- The boot screen you'll see should use linuxefi commands to boot the installer, and you should be able to run efibootmgr inside that system, to verify that you're running an UEFI OS.

For more information about this, refer to **this fedora wiki page (https://fedoraproject.org/wiki/Using_UEFI_with_QEMU)**.

PulseAudio

The PulseAudio daemon normally runs under your regular user account, and will only accept connections from the same user. This can be a problem if QEMU is being run as root through **libvirt**. To run QEMU as a regular user, edit `/etc/libvirt/qemu.conf` and set the `user` option to your username.

```
user = "dave"
```

You will also need to tell QEMU to use the PulseAudio backend and identify the server to connect to. First add the qemu namespace to you domain.

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
```

Then add the following section to your domain configuration using `virsh edit`.

```
<qemu:commandline>
  <qemu:env name='QEMU_AUDIO_DRV' value='pa'/>
  <qemu:env name='QEMU_PA_SERVER' value='/run/user/1000/pulse/native'/>
</qemu:commandline>
```

`1000` is your user id. Change it if necessary.

See also

- **Official libvirt web site (http://libvirt.org/drvqemu.html)**
- **Red Hat Virtualization Deployment and Administration Guide (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Virtualization_Deployment_and_Administration_Guide/index.html)**
- **Red Hat Virtualization Tuning and Optimization Guide (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Virtualization_Tuning_and_Optimization_Guide/index.html)**
- **Slackware KVM and libvirt (http://docs.slackware.com/howtos:general_admin:kvm_libvirt)**

- **IBM KVM (http://www-01.ibm.com/support/knowledgecenter/linuxonibm/liaat/liaatkvm.htm)**
- **libvirt Networking Handbook (https://jamielinux.com/docs/libvirt-networking-handbook/)**

Retrieved from "https://wiki.archlinux.org/index.php?title=Libvirt&oldid=510505"
