目 录

1 libvirt **3**
1.1 libvirt 的介绍 . . . 3
1.2 libvirt 的安装 . . . 4
1.3 libvirt 和 libvirtd 的配置 . . . 5
1.3.1 libvirt 的配置文件 . . . 5
1.3.2 libvirtd 的配置 . . . 6
1.4 libvirt 域的 XML 配置文件 . . . 7
1.4.1 CPU 的配置 . . . 7
1.4.2 内存的配置 . . . 8
1.4.3 客户机启动的配置 . . . 8
1.4.4 网络的配置 . . . 9
1.4.5 存储的配置 . . . 10
1.4.6 其他配置简介 . . . 11

2 virsh **13**
2.1 virsh 常用命令 . . . 13
2.1.1 域管理的命令 . . . 13
2.1.2 宿主机和 Hypervisor 的管理命令 . . . 14
2.1.3 网络的管理命令 . . . 14
2.1.4 存储池和存储卷的管理命令 . . . 15
2.1.5 其他常用命令 . . . 15

3 创建一个虚拟机 **16**
3.1 制作虚拟机镜像 . . . 16
3.2 编写客户机配置文件 . . . 16
3.3 创建虚拟机 . . . 17

4 建立到 Hypervisor 的连接 **18**
4.1 使用本地 URI 连接 Hypervisor . . . 18
4.2 使用远程 URI 连接 Hypervisor . . . 18
4.3 连接到 Hypervisor 的例子 . . . 19

5 libvirt API **20**
5.1 libvirt API 的简介 20
5.1.1 连接 Hypervisor 的 API 20
5.1.2 域管理的 API 20
5.1.3 节点管理的 API 20
5.1.4 网络管理的 API 21
5.1.5 存储卷管理的 API 21
5.1.6 存储池管理的 API 21
5.2 使用 libvirt API 的例子 22

6 自动化安装 OpenStack **23**

1 libvirt

1.1 libvirt 的介绍

libvirt 是用于管理平台虚拟化技术的应用程序接口、守护进程和管理工具，它不仅提供了对虚拟化客户机的管理，也提供了对虚拟化网络和存储的管理。

在 libvirt 中有几个重要的概念，如下所示：

- Node 又叫做节点，是一个物理机器，上面可能运行着多个虚拟客户机。Hypervisor 和 Domain 都运行在节点之上。
- Hypervisor 又叫做虚拟机监控器，比如 KVM、Xen、VMware、Hyper-V 等，是虚拟化中的一个底层软件层，它可以虚拟化一个节点让其运行多个虚拟客户机。
- Domain 又叫做域，是在 Hypervisor 上运行的一个客户机操作系统实例。

libvirt 被用于管理节点上的各个域，其中的管理功能包括以下四个部分：

1. 域的管理。包括对域的生命周期的管理以及管理对多种设备类型的热拔插操作。
2. 远程节点的管理。libvirt 支持多种网络远程传输类型。只要物理节点上运行了 libvirtd 这个守护进程，远程的管理程序就可以连接到该节点进行管理操作。
3. 存储的管理。任何运行了 libvirtd 守护进程的主机，都可以通过 libvirt 来管理不同类型的存储。
4. 网络的管理。任何运行了 libvirtd 守护进程的主机，都可以通过 libvirt 来管理物理的和逻辑的网络接口。

libvirt 由三部分组成：

- 应用程序编程接口库，为其他虚拟机管理工具提供虚拟机管理的程序库支持。
- libvirtd 守护进程，负责执行对节点上的域的管理工作。
- virsh，是 libvirt 项目中默认的对虚拟机管理的一个命令行工具。

1.2 libvirt 的安装

安装 libvirt 的步骤如下：

1. 首先检查是否安装过 libvirt，命令如下所示：

```
which libvirtd
```

如果有安装过 libvirt，就应该先清除之前装过的 libvirt，命令如下所示：

```
// 仅在ubuntu16.04下试验过
sudo apt remove libvirt-bin
```

2. 下载 libvirt 的源代码，命令如下所示：

```
// 下载日期为2016.12.25，此时最新版本为2.5.0
wget http://libvirt.org/sources/libvirt-2.5.0.tar.xz
```

3. 配置 libvirt 前，需要安装一些工具。相关命令如下所示：

```
sudo apt-get install libpciaccess-dev
sudo apt-get install libxml++2.6-2v5
sudo apt-get install libxml++2.6-dev
sudo apt-get install libyajl-dev
sudo apt-get install libdevmapper-dev
sudo apt-get install libnl-3-dev
sudo apt-get install libnl-route-3-dev
```

4. 配置 libvirt，命令如下所示：

```
./configure
```

5. 编译 libvirt，命令如下所示：

```
make -j 4
```

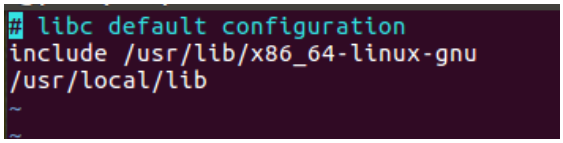
6. 安装 libvirt，命令如下所示：

```
sudo make install
```

7. 配置动态链接，命令如下所示：

```
sudo vi /etc/ld.so.conf.d/libc.conf
```

将文件内容写为下图中的内容：



然后再输入如下命令：

```
sudo ldconfig
```

8. 检查是否安装成功，命令如下所示：

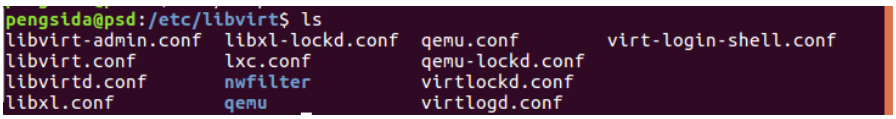
```
which libvirtd
libvirted ——version
virsh
```

如果安装成功，将得到如下图的结果：

1.3 libvirt 和 libvirtd 的配置

1.3.1 libvirt 的配置文件

libvirt 的相关配置文件都在/etc/libvirt/目录下，如下图所示：



下面介绍其中几个重要的配置文件和目录：

1. /etc/libvirt/libvirt.conf。这个文件用于配置一些常用的 libvirt 连接的别名，文件内容可以如下所示：

```
uri_aliases = [
    "remote = qemu+ssh://root@192.168.93.201/system",
]
```

文件中，将“remote”这个别名用于指代“qemu+ssh://root@192.168.93.201/system”这个 libvirt 连接。

2. /etc/libvirt/libvirtd.conf。这个文件是 libvirt 的守护进程 libvirtd 的配置文件。文件中使用“配置项 = 值”这样的配对格式来配置 libvirtd。

   例如，下面的几个配置项表示关闭 TLS 安全认证的连接、打开 TCP 连接、设置 TCP 监听的端口、TCP 连接不使用认证授权方式以及设置 UNIX domain socket 的保存目录。如下所示：

```
listen_tls = 0
listen_tcp = 1
tcp_port = "16666"
auth_tcp = "none"
unix_socket_dir = "/var/run/libvirt"
```

   需要注意的是，这个文件被修改后，需要让 libvirtd 重新加载配置文件才会生效。如果想要让 TCP、TLS 等连接生效，需要在启动 libvirtd 时加上“–listen”参数，命令如下所示：

```
libvirtd —listen
```

3. /etc/libvirt/qemu.conf。这个文件是 QEMU 驱动的配置文件。

4. /etc/libvirt/qemu/目录。在这个目录下存放着使用 QEMU 驱动的域的配置文件。

1.3.2 libvirtd 的配置

下面介绍以下几个 libvirtd 命令行的参数：

-d	表示让 libvirtd 作为守护进程在后台运行。
-f FILE	指定 libvirtd 的配置文件为 FILE。默认值为/etc/libvirt/libvirtd.conf
-l	开启配置文件中配置的 TCP/IP 连接。
-p FILE	将 libvirtd 进程的 PID 写入到 FILE 文件中。默认值为/var/run/libvirt.pid
-t SECONDS	设置对 libvirtd 连接的超时时间为 SECONDS 秒。
-v	libvirtd 运行时输出详细的输出信息。
–version	显示 libvirtd 程序的版本信息。

1.4 libvirt 域的 XML 配置文件

1.4.1 CPU 的配置

XML 配置文件中使用 vcpu 标签表示客户机中 vCPU 的个数，使用 features 标签表示为客户机打开或关闭 CPU 或其他硬件的特性。具体例子如下所示：

```
<domain>
    <!-- 设置客户机中vCPU的个数为2 -->
    <vcpu placement='static'>2</vcpu>
    <!-- 打开客户机的ACPI、APIC和PAE特性 -->
    <features>
        <acpi/>
        <apic/>
        <pae/>
    </features>
</domain>
```

libvirt 还提供了 cputune 标签来对 CPU 分配进行更多调节，具体例子如下所示：

```
<domain>
    <cputune>
        <vcpupin vcpu="0" cpuset="1"/>
        <vcpupin vcpu="1" cpuset="2,3"/>
        <vcpupin vcpu="2" cpuset="4"/>
        <vcpupin vcpu="3" cpuset="5"/>
        <emulatorpin cpuset="1-3">
        <iothreadpin iothread="1" cpuset="5,6"/>
        <iothreadpin iothread="2" cpuset="7,8"/>
        <shares>2048</shares>
        <period>1000000</period>
        <quota>-1<quota>
        <emulator_period>1000000</emulator_period>
        <emulator_quota>-1</emulator_quota>
        <iothread_period>1000000</iothread_period>
        <iothread_quota>-1<iothread_quota>
        <vcpushed vcpus='0-4,^3' scheduler='fifo' priority='1'/>
        <iothreadsched iothreads='2' scheduler='batch'/>
    </cputune>
</domain>
```

在此介绍以上例子中的标签：

cputune	在这个标签中可以设置 cpu 的参数
vcpupin	该标签表示了 vCPU 应该放置到宿主机哪个 CPU 上
emulatorpin	该标签表示了 qemu emulator 应该绑定到哪个宿主机 CPU 上
iothreadpin	该标签表示了 IOThreads 应该绑定到哪个宿主机 CPU 上
shares	该标签表示了客户机占用 CPU 时间的加权配额
period	该标签表示了 vCPU 的执行周期，在这个时间段内，vCPU 不允许执行超过 quota 的时间。这个值的范围是 [1000, 1000000]

quota	该标签表示了 vCPU 的最大允许执行时间。如果 quota 为负，说明 vCPU 可以执行无限长的时间。
emulator_period	该标签表示了 emulator 的执行周期。在这个时间段内，emulator 不能执行超过 emulator_quota 的时间。
emulator_quota	该标签表示了 emulator 的最大允许执行时间。如果 emulator_quota 为负，说明 emulator 可以执行无限长的时间。
iothread_period	该标签表示了 IOThreads 的执行周期。在这个时间段内，IOThreads 不能执行超过 iothread_quota 的时间。
iothread_quota	该标签表示了 IOThreads 的最大允许执行时间。如果 iothread_quota 为负，说明 IOThreads 可以执行无限长的时间。
vcpushed	该标签表示了 vcpu 调度算法的类型
iothreadsched	该标签表示了 IOThread 调度算法的类型

1.4.2 内存的配置

内存配置的例子如下所示：

```
<domain>
    <maxMemory slots='16'></maxMemory>
    <memory unit='KiB'>524288</memory>
    <currentMemory unit='KiB'>524288</currentMemory>
</domain>
```

在此介绍以上例子中的标签：

memory	该标签表示客户机启动时最大可使用的内存
maxMemory	该标签表示客户机运行时最大可使用的内存
currentMemory	该标签表示客户机实际被分配的内存

1.4.3 客户机启动的配置

首先看配置的例子：

```
<os>
    <type>hvm</type>
    <loader readonly='yes' secure='no' type='rom'>/usr/lib/xen/boot/hvmloader</
        loader>
    <nvram template='/usr/share/OVMF/OVMF_VARS.fd'>/var/lib/libvirt/nvram/
        guest_VARS.fd</nvram>
    <boot dev='hd'/> <boot dev='cdrom'/>
    <bootmenu enable='yes' timeout='3000'/>
    <smbios mode='sysinfo'/>
    <bios useserial='yes' rebootTimeout='0'/>
</os>
```

在此介绍以上例子中中的标签：

type	在虚拟机中启动的操作系统类型
loader	该标签指定了固件用于协助启动客户机
nvram	该标签指定了一个文件，用于存放客户机中不可修改的变量
boot	该标签指定了下一个要启动的设备
smbios	该标签指定了客户机 SMBIOS 信息可见的方式，有‘emulate’、‘host’、‘sysinfo’三种方式
bootmenu	该标签决定了是否在客户机启动时有启动菜单
bios	useserial 属性决定了用户能否在串口看到 BIOS 信息，rebootTimeout 属性决定了启动过程中客户机多久后重新启动

1.4.4 网络的配置

libvirt 中支持以下四种网络配置方式：

1. 桥接方式的网络配置
2. NAT 方式的虚拟网络配置
3. 用户模式网络的配置
4. 网卡设备直接分配

使用桥接方式的网络的配置如下所示：

```
<devices>
    <interface type='bridge'>
        <mac address='52:54:00:e9:e0:3b'/>
        <source bridge='br0'>
        <model type='virtio'>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
</devices>
```

在此介绍以上例子中使用的标签：

interface	属性 type='bridge' 表示使用桥接方式使客户机获得网络
max	属性 address 用于配置客户机网卡的 MAC 地址
source	属性 bridge='br0' 表示使用宿主机中的 br0 来建立网桥
model	属性 type='virtio' 表示在客户机中使用 virtio-net 驱动的网卡设备
address	属性 pci 用于配置该网卡在客户机中的 PCI 设备编号为 0000:00:03.0

使用 NAT 进行虚拟网络的配置的例子如下所示：

```
<devices>
    <interface type='network'>
        <mac address='52:54:00:32:7d:f6'/>
        <source network='default'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
</devices>
```

这里 <interface type='network'> 和 <source network='default'/> 表示使用 NAT 的方式。使用 NAT 必须保证宿主机中运行着 DHCP 和 DNS 服务器。

使用用户模式的网络配置如下所示：

```
<devices>
    <interface type='user'>
        <mac address="00:11:22:33:44:55"/>
    </interface>
</devices>
```

这里 <interface type='user'> 表示该客户机的网络接口是用户模式网络，是完全有 qemu-kvm 软件模拟的一个网络协议栈。

使用网卡设备直接分配的网络配置的例子如下：

```
<devices>
    <interface type='hostdev'>
        <source>
            <address type='pci' domain='0x0000' bus='0x08' slot='0x10' function='0
                x0'/>
        </source>
        <mac address='52:54:00:6d:90:02'/>
    </interface>
</devices>
```

这里 <interface type='hostdev'> 指定将网卡设备直接分配给客户机使用。以下代码直接将宿主机中的 PCI 0000:08:00.0 设备直接分配给客户机使用：

```
<source>
    <address type='pci' domain='0x0000' bus='0x08' slot='0x10' function='0x0'/>
</source>
```

1.4.5 存储的配置

首先来看一个关于客户机磁盘的配置例子：

```
<devices>
    <disk type='file' device='disk'>
        <driver name='qemu' type='raw' cache='none'/>
        <source file='/var/lib/libvirt/images/ubuntu1604.img'/>
```

```
        <target dev='vda' bus='virtio'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </disk>
</devices>
```

在此介绍上面例子中的标签：

disk	客户机磁盘配置的主标签，属性 type 表示磁盘使用哪种类型作为磁盘的来源，属性 device 表示让客户机如何使用该磁盘设备
driver	用于定义 Hypervisor 如何为该磁盘提供驱动，属性 name 制定了宿主机中使用的后端驱动名称，属性 type 指定了镜像文件的格式，属性 cache 表示在宿主机中打开该磁盘时使用的缓存方式
source	属性 file 指定了磁盘的来源
target	表示将磁盘暴露给客户时的总线类型和设备名称，属性 dev 制定了客户机中该磁盘设备的逻辑设备名称，属性 bus 指定了该磁盘设备被模拟挂载的总线类型
address	表示该磁盘设备在客户机中的 PCI 地址

1.4.6 其他配置简介

使用 domain 标签配置域，如下所示：

```
<domain type='kvm'>
    <!-- ... -->
</domain>
```

domain 标签中有两个属性：

type	用于表示 Hypervisor 的类型，可选值有 xen、kvm、qemu、lxc、kqemu 和 vmware
id	用于标识在 libvirt 中运行的客户机，如果不设置 id 属性，libvirt 会按顺序分配一个最小可用的 ID

还可以配置域的元数据，例子如下所示：

```
<domain type='xen' id='3'>
    <name>fv0</name>
    <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
    <title>A short description - title - of the domain</title>
    <description>Some human readable description</description>
    <metadata>
        <app1:foo xmlns:app1="http://app1.org/app1/">..</app1:foo>
```

```
        <app2:bar xmlns:app2="http://app1.org/app2/">..</app2:bar>
    </metadata>
</domain>
```

以上例子的标签介绍如下：

name	为客户机提供了名字
uuid	为客户机提供了 uuid
title	为域提供了简短的描述，不可以包换换行符
description	为客户机提供了描述
metadata	可以被应用程序用来存放自定义的元数据

可以使用 emulator 标签指定 QEMU 模拟器的绝对路径，如下所示：

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

可以使用 controller 标签来配置 PCI 控制器，相关例子如下所示：

```
<domain>
    <controller type='usb' index='0'>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
    </controller>
    <controller type='usb' index='0'>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
    </controller>
</domain>
```

在这个例子里指定了一个 USB 控制器和一个 IDE 控制器。

2 virsh

virsh 是完全在命令行文本模式下运行的用户态工具，用于管理虚拟化环境中的客户机和 Hypervisor。

2.1 virsh 常用命令

在 linux 系统中可以通过“man virsh”命令查看 virsh 的帮助文档。以下介绍一些 virsh 的常用命令。

2.1.1 域管理的命令

下面是域管理中常用的 virsh 命令：

list	获取当前节点上所有域的列表
domstate<ID or NAME or UUID>	获取一个域的运行状态
dominfo<ID>	获取一个域的基本信息
domid<NAME or UUID>	根据域的名称或 UUID 返回域的 ID 值
domname<ID or UUID>	根据域的 ID 或 UUID 返回域的名称
dommemstat<ID>	获取一个域的内存使用情况的统计信息
setmem<ID><mem-size>	设置一个域的内存大小
vcpuinfo<ID>	获取一个域的 vCPU 的基本信息
vcpupin<ID><vCPU><pCPU>	将一个域的 vCPU 绑定到某一个物理 CPU 上运行
setvcpus<ID><vCPU-num>	设置一个域的 vCPU 个数
vncdisplay<ID>	获取一个域的 VNC 连接 IP 地址和端口
create<dom.xml>	根据域的 XML 配置文件创建一个域
suspend<ID>	暂停一个域
resume<ID>	唤醒一个域
shutdown<ID>	让一个域执行关机操作
reboot<ID>	让一个域重启
reset<ID>	强制重启一个域，相当于按电源“reset”按钮
destroy<ID>	立即删除一个域
save<ID><file.img>	保存一个运行中的域的状态到一个文件中
restore<ID><file.img>	从一个被保存的文件中恢复一个域的运行
migrate<ID><dest_url>	将一个域迁移到另外一个目的地址
dumpxml<ID>	以 XML 格式转存出一个域的信息到标准输出中

attach-device<ID><device.xml>	向一个域添加 XML 文件中的设备
detach-device<ID><device.xml>	将 XML 文件中的设备从一个域中移除
console<ID>	连接到一个域的控制台

2.1.2 宿主机和 Hypervisor 的管理命令

下面是宿主机和 Hypervisor 管理中常用的 virsh 命令:

version	显示 libvirt 和 Hypervisor 的版本信息
sysinfo	以 XML 格式打印宿主机系统的信息
nodeinfo	显示该节点的基本信息
uri	显示当前连接的 URI
hostname	显示当前节点的主机名
capabilities	显示该节点宿主机和客户机的架构和特性
freecell	显示当前 MUMA 单元的可用空闲内存
nodememstats<cell>	显示该节点的内存单元使用情况的统计
connect<URI>	连接到 URI 指示的 Hypervisor
nodecpustats<cpu-num>	显示该节点的某个 CPU 使用情况
qemu-attach<pid>	根据 PID 添加一个 QEMU 进程 libvirt 中
qemu-monitor-command domain [–hmp] command	向域的 QEMU monitor 发送一个命令

2.1.3 网络的管理命令

下面是网络管理中常用的 virsh 命令:

iface-list	显示出物理主机的网络接口列表
iface-mac<if-name>	根据网络接口名称查询其对应的 MAC 地址
iface-name<MAC>	根据 MAC 地址查询其对应的网络接口名称
iface-edit<if-name-or-uuid>	编辑一个物理主机的网络接口的 XML 配置文件
iface-dumpxml<if-name-or-uuid>	以 XML 格式转存出一个网络接口的状态信息
iface-destroy<if-name-or-uuid>	关闭宿主机上一个物理网络接口
net-list	列出 libvirt 管理的虚拟网络
net-info<net-name-or-uuid>	根据名称查询一个虚拟网络的基本信息
net-uuid<net-name>	根据名称查询一个虚拟网络的 UUID

net-name<net-UUID>	根据 UUID 查询一个虚拟网络的名称
net-create<net.xml>	根据一个网络 XML 配置文件创建一个虚拟网络
net-edit<net-name-or-uuid>	编辑一个虚拟网络的 XML 配置文件
net-dumpxml<net-name-or-uuid>	转存出一个虚拟网络的 XML 格式的配置信息
net-destroy<net-name-or-uuid>	销毁一个虚拟网络

2.1.4 存储池和存储卷的管理命令

下面是存储池和存储卷管理中常用的 virsh 命令:

pool-list	显示出 libvirt 管理的存储池
pool-info<pool-name>	根据一个存储池名称查询其基本信息
pool-uuid<pool-name>	根据存储池名称查询其 UUID
pool-create<pool.xml>	根据 XML 配置文件来创建一个存储池
pool-edit<pool-name-or-uuid>	编辑一个存储池的 XML 配置文件
pool-destroy<pool-name-or-uuid>	关闭一个存储池
pool-delete<pool-name-or-uuid>	删除一个存储池
vol-list<pool-name-or-uuid>	查询一个存储池中存储卷的列表
vol-name<vol-key-or-path>	查询一个存储卷的名称
vol-path –pool <pool><vol-name-or-key>	查询一个存储卷的路径
vol-create<vol.xml>	根据 XML 配置文件来创建一个存储卷
vol-clone<vol-name-path><name>	克隆一个存储卷
vol-delete<vol-name-or-key-or-path>	删除一个存储卷

2.1.5 其他常用命令

以下是 virsh 其他方面的常用命令:

help	显示出 virsh 的命令帮助文档
pwd	打印出当前的工作目录
cd<your-dir>	改变当前工作目录
quit	退出 virsh
exit	退出 virsh

3 创建一个虚拟机

3.1 制作虚拟机镜像

命令如下所示：

```
qemu-img create -f qcow2 Ubuntu.qcow2 10G
```

3.2 编写客户机配置文件

根据第一节中的内容，可以编写 libvirt 域的配置文件，例子如下所示：

```
<domain type='kvm'>
    <name>Ubuntu</name>
    <memory>1048576</memory>
    <currentMemory>1048576</currentMemory>
    <vcpu>4<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>restart</on_reboot>
    <on_crash>destroy</on_crash>
    <devices>
        <emulator>/usr/local/bin/qemu-system-x86_64</emulator>
        <disk type='file' device='disk'>
            <driver name='qemu' type='qcow2'/>
            <source file='/home/pengsida/下载/Ubuntu.qcow2'/>
            <target dev='hda' bus='ide'/>
        </disk>
        <disk type='file' device='cdrom'>
            <source file='/home/pengsida/下载/ubuntu.iso'/>
            <target dev='hdb' bus='ide'/>
        </disk>
        <input type='mouse' bus='ps2'/>
        <graphics type='vnc' port='-1' autoport='yes' listen='0.0.0.0' keymap='en-
            us'/>
    </devices>
</domain>
```

3.3 创建虚拟机

首先需要保证 libvirtd 守护进程是启动的，否则会报错。启动 libvirtd 守护进程的命令如下：

```
libvirtd
```

然后还需要启动 virtlogd 服务，命令如下：

```
sudo systemctl start virtlogd
# 如果遇到Unit virlogd.service is masked，就输入如下命令：
# sudo systemctl unmask virtlogd.service
# 如果遇到Unit virlogd.socket is masked，就输入如下命令：
# sudo systemctl unmask virtlogd.socket
```

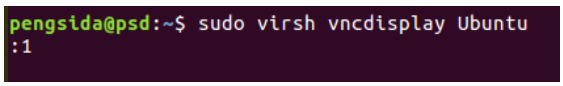
然后使用如下命令创建一个虚拟机：

```
# demo.xml是刚才创建的XML配置文件的文件名
sudo virsh create demo.xml
```

使用以下命令可以通过 vncviewer 查看虚拟机：

```
# Ubuntu是刚刚创建的域的名字
sudo virsh vncdisplay Ubuntu
```

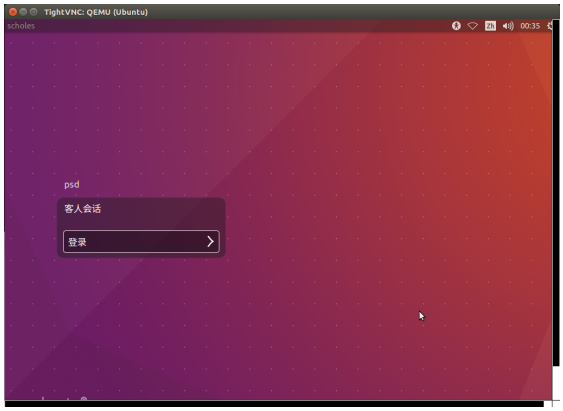
如下图所示：



注意到 terminal 输出“:1”，可以根据这个参数查看虚拟机，命令如下所示：

```
vncviewer :1
```

安装成功后，可以看到如下界面：



4 建立到 Hypervisor 的连接

要使用 libvirt API 进行虚拟化管理，就必须先建立到 Hypervisor 的连接。

在使用 virsh 工具时，可以使用“-c”参数来指定建立到某个 URI 上的连接，相关命令如下所示：

```
sudo virsh -c URI
# 如，virsh -c qemu:///system
# 又如，virsh -c qemu+ssh://root@192.168.158.31/system
```

4.1 使用本地 URI 连接 Hypervisor

使用本地 URI 可以连接本系统范围内的 Hypervisor，本地 URI 的一般格式如下：

```
driver[+transport]:///[path][?extral-param]
```

对其中的元素解释如下：

driver	连接 Hypervisor 的驱动名称
transport	选择该连接所使用的传输方式
path	连接到服务器端上的某个路径
?extral-param	用于添加一些额外的参数

本地连接 KVM 的 URI 的例子如下：

```
# 连接到本地的session实例，该连接仅能管理当前用户虚拟化资源
qemu:///session

# 以Unix domain socket的方式连接到本地的session实例，该连接仅能管理当前用户的虚拟化资源
qemu+unix:///session

# 连接到本地的system实例，该连接可以管理当前节点的所有虚拟化资源
qemu:///system

# 以Unix domain socket的方式连接到本地的system实例，该连接可以管理当前节点的所有虚拟化资源
qemu:///system
```

4.2 使用远程 URI 连接 Hypervisor

使用远程 URI 可以连接到网络上的 Hypervisor，远程 URI 的例子如下：

```
driver[+transport]://[user@][host][:port]/[path][?extral-param]
```

对其中的元素解释如下：

driver	连接 Hypervisor 的驱动名称
transport	选择该连接所使用的传输方式，取值可以是 ssh、tcp 和 libssh2
user	远程主机使用的用户名
host	远程主机的主机名或 IP 地址
port	连接远程主机的端口
path	连接到服务器端上的某个路径
?extral-param	用于添加一些额外的参数

远程连接 KVM 的 URI 的例子如下：

```
# 通过ssh通道连接到远程节点的system实例，以最大权限管理远程节点上的虚拟化资源
qemu+ssh://root@example.com/system

# 通过ssh通道连接到远程节点的使用user用户的session实例，仅能对user用户的虚拟化资源
    进行管理
qemu+ssh://user@example.com/session

# 通过加密的TLS连接到远程节点的system实例，以最大权限管理远程节点上的虚拟化资源
qemu://example.com/system

# 通过加密的TCP连接到远程节点的system实例，以最大权限管理远程节点上的虚拟化资源
qemu+tcp://example.com/system
```

4.3 连接到 Hypervisor 的例子

需要注意的是，如果要连接到某个节点上的 Hypervisor，需要保证那个节点上的 libvirtd 守护进程正在执行，否则会报错。

连接到当地 Hypervisor 的例子如下图所示：

5 libvirt API

5.1 libvirt API 的简介

libvirt API 可以分为 8 个部分，接下来在每小节列出常用的 API 函数。

5.1.1 连接 Hypervisor 的 API

连接 Hypervisor 相关的 API。有以下函数：

virConnectOpen	建立一个连接，返回值是一个 virConnectPtr 对象，该对象代表到 Hypervisor 的一个连接
virConnectOpenReadOnly	建立一个只读的连接
virConnectGetCapabilities	返回对 Hypervisor 和驱动的功能的描述的 XML 格式的字符串
virConnectListDomains	返回一列域标识符，它们代表该 Hypervisor 上的活动域

5.1.2 域管理的 API

域管理的 API。有如下函数：

根据域的 id 值到 conn 这个连接上去查找相应的域: virDomainPtrvirDomainLookupByID
根据域的名字去查找相应的域: virDomainLookupByName
根据域的 UUID 去查找相应的域: virDomainLookupByUUID
查询域的信息: virDomainGetHostname virDomainGetInfo virDomainGetVcpus virDomainGetVcpusFlags virDomainGetCPUStats
控制域的生命周期: virDomainCreate virDomainSuspend virDomainResume virDomainDestroy virDomainMigrate

5.1.3 节点管理的 API

节点管理的 API。有如下函数：

virNodeGetInfo: 获取节点的物理硬件信息
virNodeGetCPUStats: 获取节点上各个 CPU 的使用统计信息
virNodeGetFreeMemory: 获取节点上可用的空闲内存大小
virNodeSetMemoryParameters: 设置节点上的内存调度的参数
virNodeSuspendForDuration: 让节点暂停运行一段时间

5.1.4 网络管理的 API

网络管理的 API。有如下函数：

virNetworkGetName: 获取网络的名称
virNetworkGetBridgeName: 获取该网络中网桥的名称
virNetworkGetUUID: 获取网络的 UUID 标识
virNetworkGetXMLDesc: 获取网络的以 XML 格式的描述信息
virNetworkIsActive: 查询网络是否正在使用
virNetworkCreateXML: 根据提供的 XML 格式的字符串创建一个网络
virNetworkDestroy: 销毁一个网络
virNetworkFree: 回收一个网络
virNetworkUpdate: 根据 XML 格式的网络配置来更新一个已存在的网络
virInterfaceCreate: 创建一个网络接口
virInterfaceFree: 释放一个网络接口
virInterfaceDestroy: 销毁一个网络接口
virInterfaceGetName: 获取网络接口的名称
virInterfaceIsActive: 查询网络接口是否正在运行

5.1.5 存储卷管理的 API

存储卷管理的 API。有如下函数：

virStorageVolLookupByKey: 根据全局唯一的键值来获得一个存储卷的对象
virStorageVolLookupByName: 根据名称来获得一个存储卷的对象
virStorageVolLookupByPath: 根据节点上的路径来获取一个存储卷的对象
virStorageVolGetInfo: 查询某个存储卷的使用情况
virStorageVolGetPath: 获取存储卷的路径
virStorageVolGetConnect: 查询存储卷的连接
virStorageVolCreateXML: 根据 XML 配置文件来创建一个存储卷
virStorageVolFree: 释放存储卷的句柄
virStorageVolDelete: 删除一个存储卷
virStorageVolResize: 调整存储卷的大小

5.1.6 存储池管理的 API

存储池管理的 API。有如下函数：

virStoragePoolLookupByName: 根据存储池的名称来获取一个存储池对象
virStoragePoolLookupByVolume: 根据一个存储卷返回其对应的存储池对象

virStoragePoolCreateXML: 根据 XML 配置文件来创建一个存储池
virStoragePoolDefineXML: 根据 XML 配置文件静态地定义个存储池
virStoragePoolCreate: 激活一个存储池
virStoragePoolGetInfo: 获取存储池的信息
virStoragePoolGetName: 获取存储池的名称
virStoragePoolGetUUID: 获取存储池的 UUID 标识
virStoragePoolIsActive: 查询存储池是否处于使用状态
virStoragePoolFree: 释放存储池相关的内存
virStoragePoolDestroy: 用于销毁一个存储池
virStoragePoolDelete: 物理删除一个存储池资源

5.2 使用 libvirt API 的例子

下面用一个简单的例子介绍如何使用 libvirt API，例子如下：

```
#include<stdio.h>
#include<stdlib.h>
#include<libvirt/libvirt.h>

int main(int argc, char* argv[])
{
    virConnectPtr conn;
    conn = virConnectPtr("qemu:///system");
    if(conn == NULL)
    {
        fprintf(stderr, "Failed to open connection to qemu:///system");
        return 1;
    }
    else
        printf("Open connection successulfully");
    virConnectClose(conn);
    return 0;
}
```

其实这个例子不是重点，这里想重点说明的是，编译这个程序的时候，需要在后面加上“-lvirt”参数，如下所示：

```
# temp.c是刚刚那个例子的文件名字
cc temp.c -lvirt
```

编译通过以后，就可以像普通程序一样使用执行文件了。

6 自动化安装 OpenStack

这里使用 DevStack 脚本来搭建 OpenStack 开发环境，有以下两个步骤：

1. 下载 DevStack 的源代码，命令行如下：

```
git clone git://github.com/openstack-dev/devstack.git
```

2. 在 DevStack 文件夹下创建 local.conf 文件，命令如下所示：

```
sudo vi local.conf
```

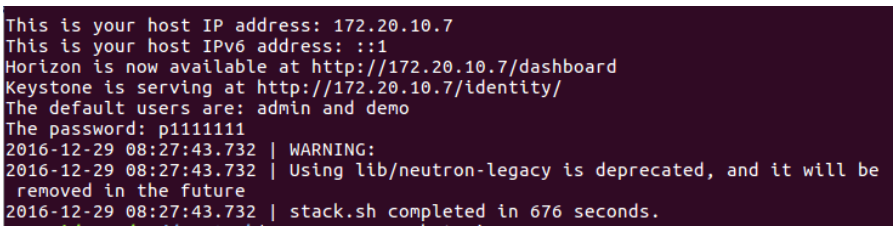
然后写入如下内容：

```
[[local|localrc]]
#NOVA
enable_service n-cell
```

3. 运行 stack.sh 脚本，命令行如下：

```
# 注意，DevStack脚本所处的路径不能包含中文
./stack.sh
```

需要注意的是，安装成功后，terminal 会输出很重要的信息，一定要记住，如下图所示：



里面有 dashboard 的登陆地址，dashboard 登陆的用户名和密码。如果丢失了这个信息，貌似是找不回来的。