目 录

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
```

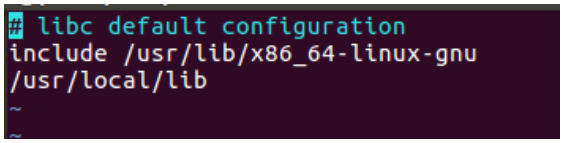
6. 安装 libvirt，命令如下所示：

```
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：

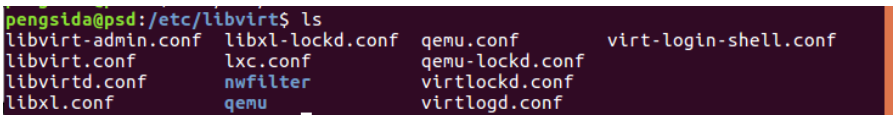
```
which libvirtd
libvirted ——version
virsh
```

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

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
```

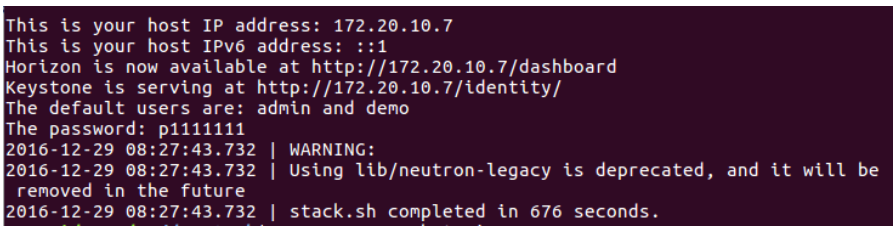
然后写入如下内容：

```
[[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。