

学习目标

快速入门

学习目标：

- 了解 虚拟化定义和目的
- 说出 虚拟化常见分类
- 说出 虚拟化常见实现方式

虚拟化基础

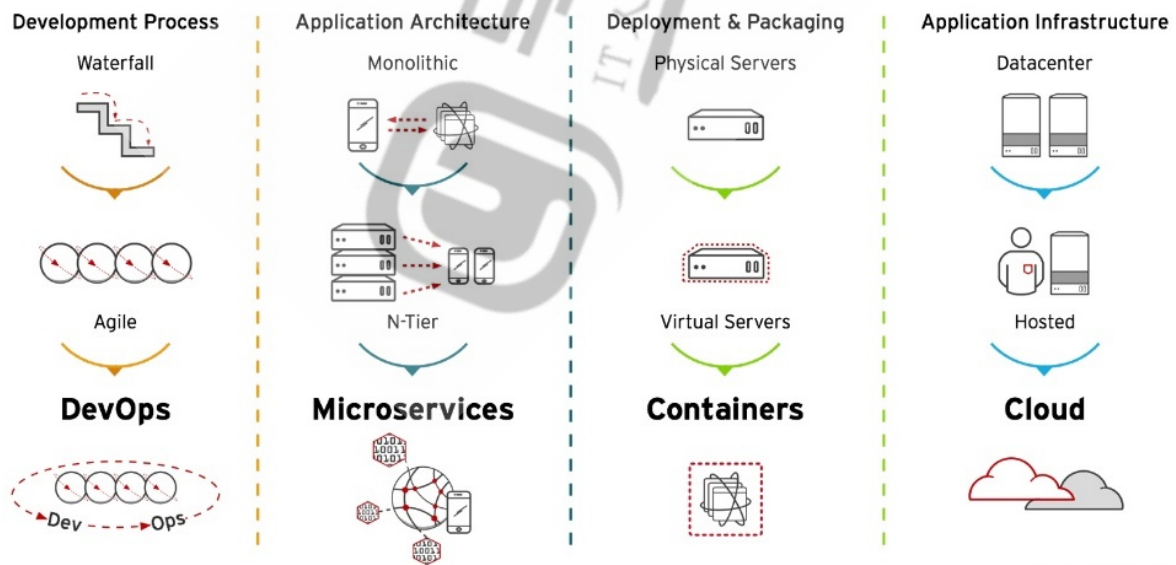
虚拟化基础

学习目标：

这一节，我们从 基础知识、虚拟分类、虚拟方式、小结 四个方面来学习。

基础知识

技术发展趋势



虚拟化是什么

虚拟化技术：

在现有物理主机基础上创造多个彼此独立的具有完整功能主机的技术。

将底层的计算机资源，抽象为彼此互相隔离的多组计算平台，每个计算平台都包含计算机的五大部件：**cpu**(控制器+寄存器)、**mem**(存储)、**IO**(输入+输出)，

我的理解

简单来说就是：

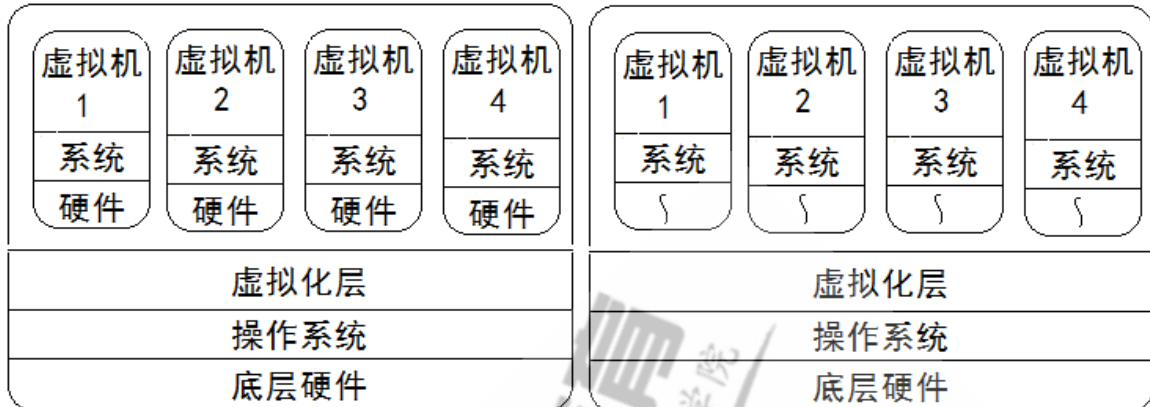
本来没有，但是通过某种特殊的手段，让你以为有，而且确信不已。
这些手段就是虚拟化技术。

虚拟化目标

在时间上和空间上突破我们工作的限制，提升工作效率。

时间上：多种工作在一段时间内同时进行

空间上：在一台物理主机上，虚拟出来多台主机，多台主机共同做一件事情。



虚拟化发展

起始：

60年代ibm公司，成本巨大的大型机，企业负担不起业务多环境，虚拟化技术实现多环境。

没落：

80年代pc机，pc机很便宜，可以解决虚拟化多环境的问题。

风云再起：

随着x86主机硬件性能的增强，各企业业务对资源的需求量增加，自03年以来，虚拟化技术风云归来，现在越来越火

虚拟分类

虚拟化技术(系统级别)

模拟：

使用模拟器技术，模拟出来一套完整的主机功能，虚拟出来的主机平台可以跟宿主机平台不一样。

全虚拟化：

使用软件方式，虚拟出一个功能完整，底层平台与宿主机一致的主机，虚拟机自以为是一个真实的主机

常见的技术：kvm(2008年发布)

最早出现在1966年

半虚拟化：

结合软件方式，虚拟出一个部分功能，底层平台与宿主机一致的主机，虚拟机知道自己是一个傀儡。

常见的技术：xen(2003年发布)

硬件辅助虚拟化：

借助主机硬件的功能，虚拟出一个完全功能的主机。他是全虚拟化技术的一个特殊表现形式

常见技术：VT-x/EPT或者AMD-v/RVI

最早出现在1972年

代表软件

开源的虚拟化技术非常多，著名的就是Xen和KVM。

xen是2003年发布，当时全球虚拟化/公有云，基本都是xen的天下。

2006年，为了对抗Xen，intel，红帽，IBM，联手搞出一个KVM。并与2008年发布，自从2010年kvm添加到RHEL6.0内核中，kvm终于成为了主流。

经过了5年的PK，特别是硬件辅助虚拟化技术的应用，xen的性能优势被一点点的超越，目前现在已经是KVM独大。AWS，阿里云，已经全面转向KVM。

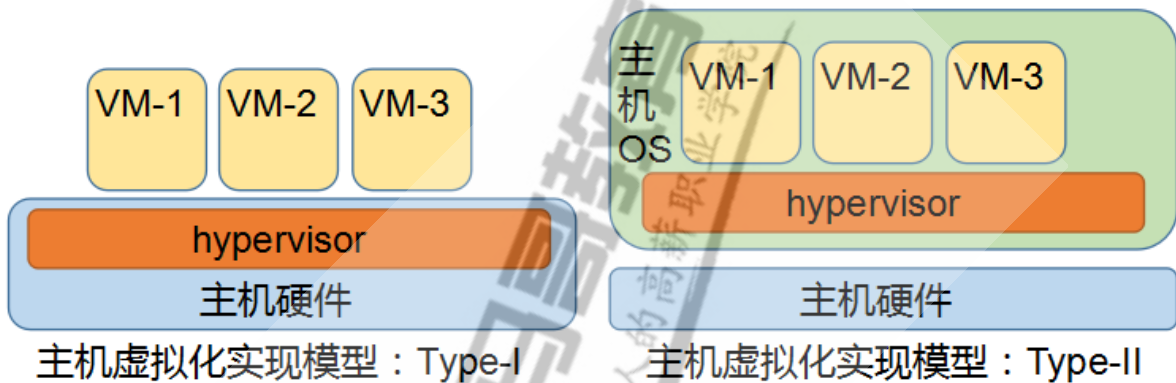
虚拟化技术(软件级别)

容器：

通过技术手段，将软件应用的运行状态保留下拉，

虚拟方式

主机的两种虚拟化实现方式：



实现特点

type-I:

实现虚拟化的方面更彻底，更可靠。

常见软件：xen，vmware ESX/ESXi

type-II:

借助于宿主机的软件来实现各种虚拟机管理

常见软件：kvm，vmware workstation，virtualbox

小结

kvm简介

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

官方介绍

Kernel Virtual Machine

KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, `kvm.ko`, that provides the core virtualization infrastructure and a processor specific module, `kvm-intel.ko` or `kvm-amd.ko`.

Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc.

KVM is open source software. The kernel component of KVM is included in mainline Linux, as of 2.6.20. The userspace component of KVM is included in mainline QEMU, as of 1.3.

Blogs from people active in KVM-related virtualization development are syndicated at <http://planet.virt-tools.org/>

关键点:

全虚拟化的解决方案

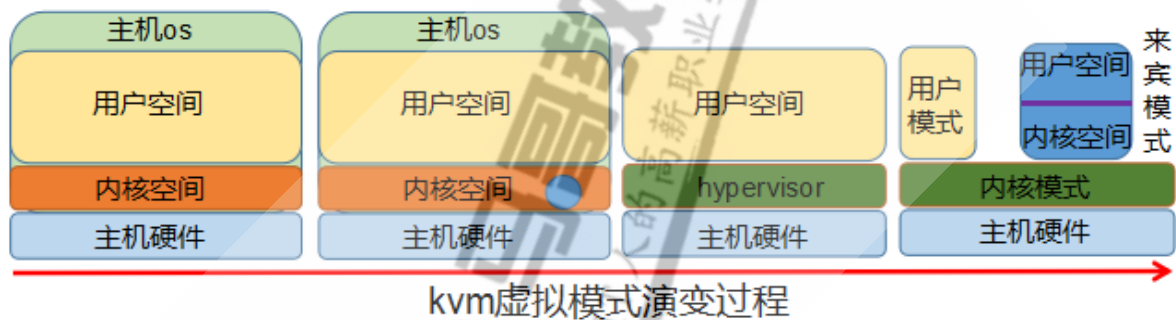
开源的，被内嵌到linux内核了

官方地址：

https://www.linux-kvm.org/page/Main_Page

原理解析

原理介绍



kvm依赖于HVM技术，只能运行在支持硬件虚拟化的CPU上，也就是说CPU支持VT-x或AMD-V，并且只支持x86_64bits系统。只要基于硬件的操作系统内核一旦装载了kvm模块，那么kvm模块就和linux内核组合成了hypervisor，变成了type-1模型的虚拟机。

kvm类似于寄生虫，linux内核一旦被寄宿，就没有自主权，跟着kvm一起变成hypervisor的一部分。原来的用户空间就变成了hypervisor的管理控制台运行位置，而kvm以字符设备/dev/kvm的形式作为kvm整个的调用管理接口。

kvm创建的虚拟机就是一个进程，ps查看，kill杀死

特点介绍

优势:

- 支持 CPU 和 memory 超载 (Overcommit)
- 支持 半虚拟化I/O (virtio)
- 支持 热插拔 (cpu, 块设备、网络设备等)
- 支持 多存储管理
- 支持 实时迁移 (Live Migration)
- 支持 PCI 设备直接分配和 单根I/O 虚拟化 (SR-IOV)
- 支持 内核同页合并 (KSM)
- 支持 NUMA (Non-Uniform Memory Access, 非一致存储访问结构)

关键点:

- Cpu 支持硬件虚拟化
- CPU 不建议超过当前核心数
- 迁移时候，基础环境要一致

小结

部署KVM

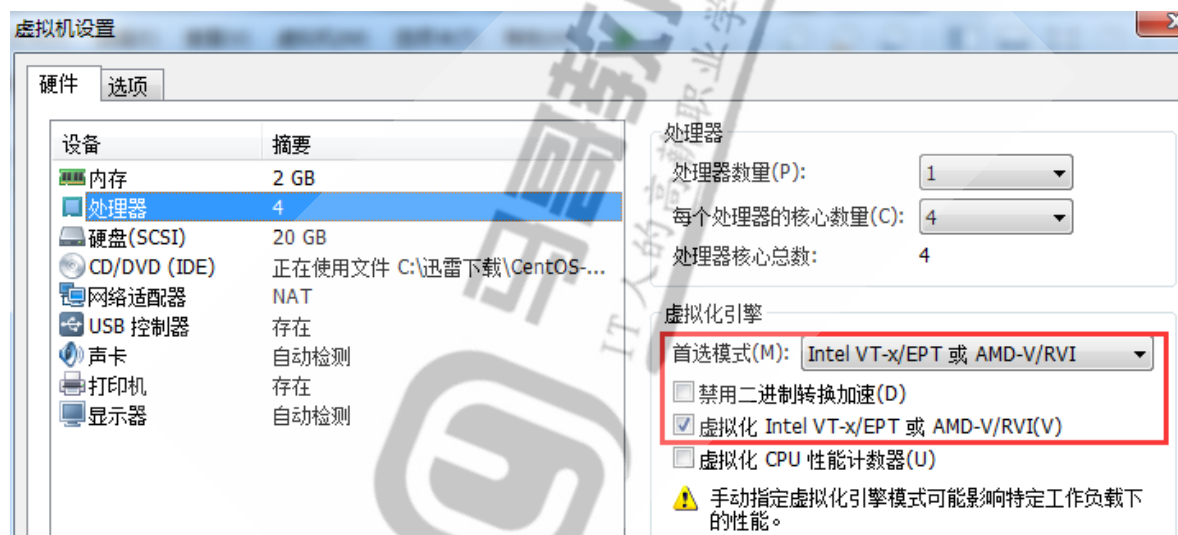
学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

前提

- 1 有一台创建好的虚拟机
- 2 因为我们实验kvm，所以要指定虚拟化引擎，不然的话，无法做虚拟化实验，效果如下



检查效果

```
# 确保CPU支持HVM
egrep '(vmx|svm)' /proc/cpuinfo

# 支持模块
modprobe kvm && modprobe kvm-intel
lsmod | egrep 'kvm|Module'
modinfo kvm
modinfo kvm-intel

# 验证
ls /dev/kvm
```

注意：

如果有信息，就表明我们的虚拟机支持虚拟化。

软件安装

qemu-kvm: 用户态管理kvm,网卡、声卡、PCI设备等都需要qemu来管理。

libvirt: 是管理kvm的工具

virt-install: 安装虚拟机用的

简单实践

安装kvm

```
apt install -y qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils  
virtinst virt-manager virt-viewer
```

工具简介

qemu-kvm	为 KVM 管理程序提供硬件模拟的软件程序
libvirt-daemon-system	将 libvirt 守护程序作为系统服务运行的配置文件
libvirt-clients	用来管理虚拟化平台的软件
bridge-utils	用来配置网络桥接的命令行工具
virtinst	用来创建虚拟机的命令行工具
virt-manager	提供一个易用的图形界面，并且通过libvirt 支持用于管理虚拟机的命令行工具
virt-viewer	查看虚拟机的控制端工具

启动服务

检查服务状态
`systemctl is-active libvirtd`

启动服务效果
`systemctl start libvirtd`
`systemctl enable libvirtd`

检查效果

```
ifconfig  
ip route  
ps aux | grep -v grep | grep dns
```

目录结构

虚拟机管理目录 /etc/libvirt

小结

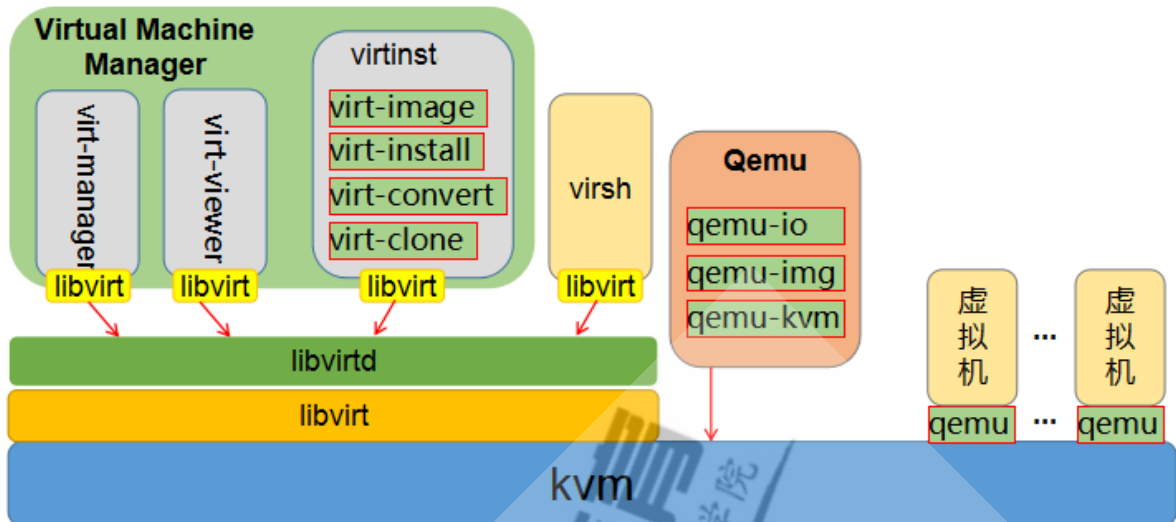
常见命令

学习目标：

这一节，我们从 基础知识、小结 两个方面来学习。

基础知识

工具体系



体系解析

虚拟机的管理有两套管理工具：

qemu: linux内核支持的专用kvm管理工具

qemu-kvm 结合虚拟化软件qemu管理kvm虚拟机的集成工具集
qemu-img 虚拟磁盘的管理工具

libvirt: 通用的虚拟化技术管理工具

virt-install 虚拟机专用创建工具
virsh 操作虚拟环境的一个命令行管理工具，支持远程管理操作
virt-manager 管理虚拟机的图形化的桌面用户接口

我们的学习过程中创建虚拟机选用**virt-install**，管理虚拟机使用**virsh**，磁盘用**qemu-img**。

配置结构

安装完毕kvm之后，会形成一个专用的目录结构

```
# ls /etc/libvirt/
hooks                libxl-sanlock.conf  qemu-sanlock.conf
libvirt-admin.conf   lxc.conf            secrets
libvirt.conf         nwfilter            storage
libvirtd.conf        qemu                virtlockd.conf
libxl.conf           qemu.conf           virtlogd.conf
libxl-lockd.conf     qemu-lockd.conf     virt-login-shell.conf
```

注意：

qemu目录是我们创建虚拟机后，保存虚拟机配置文件的目录

小结

核心知识

本节学习目标：

- 应用 虚拟机的多种核心技术，熟练进行虚拟机的各种操作

基础操作

初始化

学习目标：

这一节，我们从 基础知识、简单实践、系统安装、简单操作、小结 三个方面来学习。

基础知识

需求：要创建虚拟机，就需要给他提供一个虚拟的磁盘，我们就在/opt目录下创建一个10G大小的raw格式的虚拟磁盘CentOS-8-x86_64.raw

查看子命令帮助：

```
qemu-img create -f 磁盘格式 -o ?
```

命令格式：qemu-img create -f 磁盘格式 磁盘名称 磁盘大小

```
qemu-img create -f raw /opt/CentOS-8-x86_64.raw 10G
```

执行效果：

```
# ls /opt/  
CentOS-8-x86_64.raw
```

简单实践

使用virt-install命令，基于我们提供的系统镜像和虚拟磁盘来创建一个虚拟机，另外在创建虚拟机之前，提前打开vnc客户端，在创建虚拟机的时候，通过vnc来连接虚拟机，对其进行配置和查看进度。

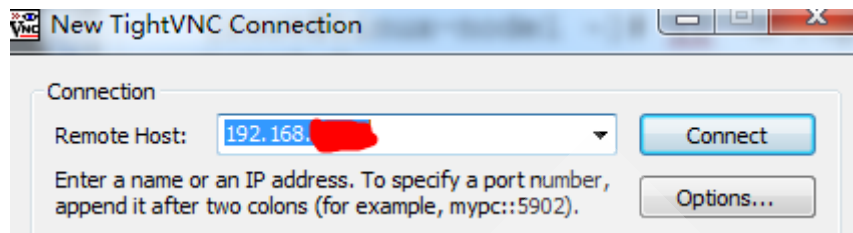
命令格式：virt-install 启动参数

```
virt-install --virt-type kvm --name CentOS-8-x86_64 --memory 1024 --  
cdrom=/tmp/CentOS-8.4.2105-x86_64-boot.iso --disk=/opt/CentOS-8-x86_64.raw --  
network network=default --graphics vnc,listen=0.0.0.0 --noautoconsole
```

参数解析

<code>--virt-type kvm</code>	指定创建的虚拟机基于的虚拟化技术
<code>--name CentOS-8-x86_64</code>	指定创建虚拟机的名字，推荐"简洁"
<code>--ram 1024</code>	指定创建虚拟机的内存大小
<code>--cdrom=XX.iso</code>	指定创建虚拟机使用的系统镜像
<code>--disk path=XX.raw</code>	指定创建虚拟机使用的虚拟磁盘
<code>--network network=default</code>	指定创建虚拟机使用的网络类型
<code>--graphics vnc,listen=0.0.0.0</code>	指定访问虚拟机使用的客户端配置信息
<code>--noautoconsole</code>	禁用配置的客户端自动连接虚拟机

通过vnc客户端连接到安装界面,kvm虚拟机的默认端口从5900开始，输入虚拟机的宿主机ip地址192.168.X.X，然后点击“connect”效果如下：



注意：

要在创建虚拟机命令执行前打开终端界面
退出全屏的快捷键：Ctrl+Alt+Shift+F

系统安装

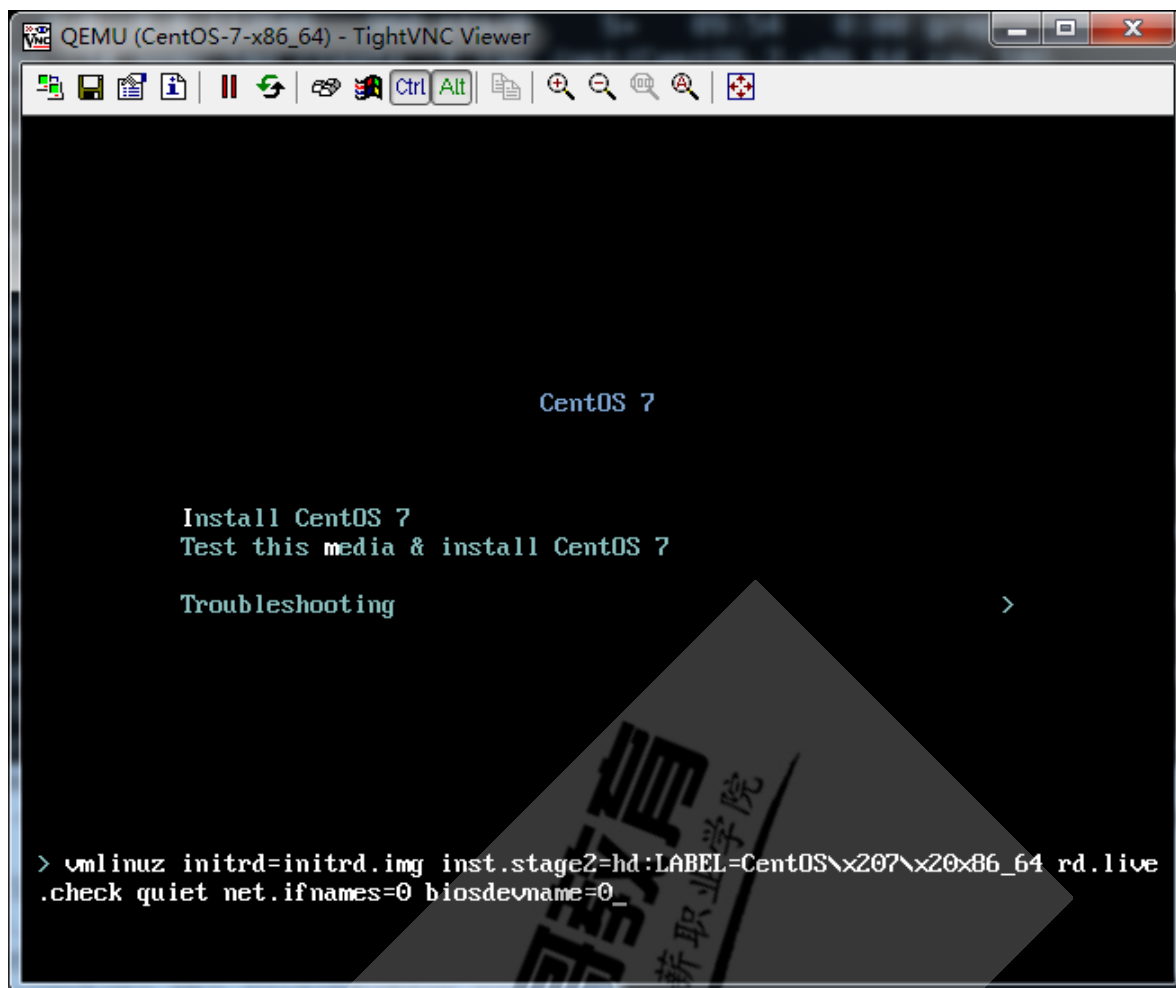
网卡配置

选择安装Centos条目

光标移动到“Install CentOS7 ...”，

通过给内核传递参数，配置网卡

按Tab键，编辑配置，在末尾追加 `net.ifnames=0 biosdevname=0`，编辑完毕后，直接按Enter

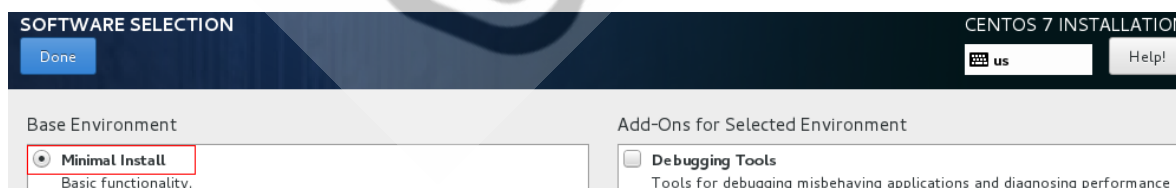


接下来跟安装普通系统的步骤一致，毕竟是测试，所以注意以下几项：

网卡打开：



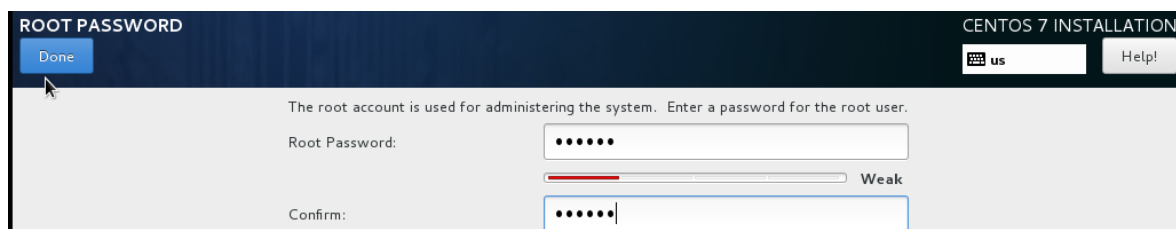
安装软件：最小化



检查安装源：(默认)



设置最简单密码：123456



等待系统安装完毕后，点击reboot，重启虚拟机

简单操作

vnc中虚拟机是不会重启的，需要在宿主机系统中，使用命令开启虚拟机

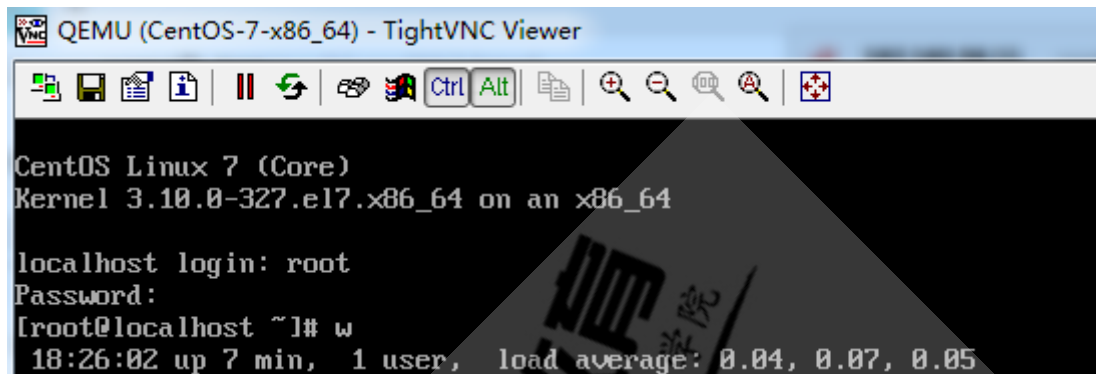
命令格式：virsh start 虚拟机名称

```
virsh start CentOS-8-x86_64
```

注意：

每次关闭虚拟机后，再次开启虚拟机，虚拟机的id都会自动的变化

vnc连接登录



查看ip地址

```
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 52:54:00:72:25:64 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.24/24 brd 192.168.122.255 scope global dynamic eth0
        valid_lft 2682sec preferred_lft 2682sec
    inet6 fe80::5054:ff:fe72:2564/64 scope link
        valid_lft forever preferred_lft forever
```

查看路由信息

```
[root@localhost ~]# ip route
default via 192.168.122.1 dev eth0 proto static metric 100
192.168.122.0/24 dev eth0 proto kernel scope link src 192.168.122.24 metric 100
```

可以看到：操作方式是跟正常操作linux系统的方式是一致的。

宿主机检查

宿主机检查网络

`ifconfig`

效果:

虚拟机启动后, 当前系统中, 多出来了一个网卡vnet0, 这个vnet网卡没有ipv4的地址

宿主机检查进程

`ps aux | grep CentOS`

效果:

虚拟机, 本质上就是在宿主机系统上的一个kvm进程, 管理虚拟机跟管理进程基本一样

注意:

我的libvirtd服务即使停止了, 它不会影响虚拟机的正常运行

但是就不能使用virsh来管理虚拟机了

配置解析

虚拟机配置

```
root@python-auto:/etc/libvirt/qemu# tree /etc/libvirt/qemu
/etc/libvirt/qemu
├── CentOS-8-x86_64.xml
├── networks
│   ├── autostart
│   │   └── default.xml -> /etc/libvirt/qemu/networks/default.xml
│   └── default.xml
2 directories, 3 files
```

可以看到:

我们创建的虚拟机配置文件都在 `/etc/libvirt/qemu` 目录下

这个目录下的networks目录是存放的虚拟机的网络配置, 文件名称是default.xml

CentOS-8-x86_64.xml是我们定义的虚拟机的配置文件, 原则上, 这个文件只能用virsh edit进行编辑

虚拟机配置文件

```
# cat /etc/libvirt/qemu/CentOS-8-x86_64.xml
<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made using:
    virsh edit CentOS-8-x86_64      提示我们只能用virsh edit进行编辑这个文件
or other application using the libvirt API.
-->

<domain type='kvm'>                                # domain完全指虚拟机系统。type一项指明了使用的是哪种虚
拟化技术
    下面几项是虚拟机的全局配置信息:
    虚拟机名称和uuid信息
    <name>CentOS-8-x86_64</name>                        # 名字不允许包含空格
    <uuid>405f68cb-d55a-4b18-a916-f213afed2e57</uuid>
    内存信息
    <memory unit='KiB'>1048576</memory>
    <currentMemory unit='KiB'>1048576</currentMemory>
    cpu信息
    <vcpu placement='static'>1</vcpu>
    <os>
    <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
```

```

# type 表示全虚拟化还是半虚拟化, hvm表示全虚拟化
<boot dev='hd' />          # 表示虚拟机从哪里开机
# boot 怎么启动的, 如"fd"表示从文件启动, "hd"从硬盘启动, "cdrom"从光驱启动 和 "network"从
网络启动
#可以重复多行, 指定不同的值, 作为一个启动设备列表。
#The dev attribute takes one of the values "fd", "hd", "cdrom" or "network"
</os>
<features>                  # 表示硬件资源(处理器)特性
  <acpi/>
  <apic/>
</features>
<cpu mode='custom' match='exact'>
  <model fallback='allow'>SandyBridge</model>
</cpu>

```

时钟信息

```

<clock offset='utc'>        # utc表示时间模式, localtime表示使用本地时间
  <timer name='rtc' tickpolicy='catchup' />
  <timer name='pit' tickpolicy='delay' />
  <timer name='hpet' present='no' />
</clock>

```

突发事件处理机制

```

<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>

```

注意:

突发事件处理定义了当发生poweroff时, 直接destroy虚拟机。当虚拟机reboot、crash的时候, 会自动采用重启操作, 还可以自定义

虚拟机管理配置(内存和磁盘)

```

<pm>
  <suspend-to-mem enabled='no' />
  <suspend-to-disk enabled='no' />
</pm>

```

虚拟机设备信息

注意:

在虚拟化技术或者云计算中, 都使用image一词来表示虚拟磁盘

```

<devices>                  #在<device></device>中的都是虚拟外设
  模拟器信息
  <emulator>/usr/libexec/qemu-kvm</emulator>

```

虚拟机磁盘信息

```

  <disk type='file' device='disk'>
    <driver name='qemu' type='raw' />
    <source file='/opt/CentOS-8-x86_64.raw' />
    <target dev='vda' bus='virtio' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06'
function='0x0' />
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' type='raw' />
    <target dev='hda' bus='ide' />
    <readonly />
    <address type='drive' controller='0' bus='0' target='0' unit='0' />
  </disk>

```

虚拟机控制器配置信息

```

  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x7' />
  </controller>

```

```

...
<controller type='virtio-serial' index='0'>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x04'
function='0x0' />
</controller>
虚拟机网络接口配置信息
  <interface type='network'>
    <mac address='52:54:00:72:25:64' />
    <source network='default' />
    <model type='virtio' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0' />
  </interface>
# 虚拟机的网卡的mac地址，前面的52:54:00地址是固定的，后面的三项是随机的。

虚拟机串口配置信息，可以不用更改
  <serial type='pty'>
    <target port='0' />
  </serial>
虚拟机终端配置信息
  <console type='pty'>
    <target type='serial' port='0' />
  </console>
虚拟机渠道配置信息
  <channel type='unix'>
    <target type='virtio' name='org.qemu.guest_agent.0' />
    <address type='virtio-serial' controller='0' bus='0' port='1' />
  </channel>
虚拟机输入输出设备信息
  <input type='tablet' bus='usb'>
    <address type='usb' bus='0' port='1' />
  </input>
  <input type='mouse' bus='ps2' />
  <input type='keyboard' bus='ps2' />
# vnc方式登录，端口号自动分配 可以通过virsh vncdisplay来查询[vncdisplay domainId]

虚拟机图形设备信息
  <graphics type='vnc' port='-1' autoport='yes' listen='0.0.0.0'>
    <listen type='address' address='0.0.0.0' />
  </graphics>
虚拟机声音设备信息
  <video>
    <model type='cirrus' vram='16384' heads='1' primary='yes' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
function='0x0' />
  </video>
虚拟机存储信息
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
function='0x0' />
  </memballoon>
</devices>
</domain>

# 虚拟机网络文件

```

注意：配置文件中展示的所有信息，都可以在ps aux 中的进程中——找到对应的配置项

网络配置文件

```
# cat /etc/libvirt/qemu/networks/default.xml
<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made using:
    virsh net-edit default
or other application using the libvirt API.
-->

<network>
  <name>default</name>
  <uuid>beae6f6b-1e04-4085-b41f-3c33196f49ca</uuid>
  <forward mode='nat' />
  <bridge name='virbr0' stp='on' delay='0' />
  <mac address='52:54:00:13:5f:fa' />
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254' />
    </dhcp>
  </ip>
</network>
```

小结

检查、创建

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

检查

命令解析

<code>virsh list</code>	列出所有活动的虚拟机
<code>virsh list --all</code>	列出所有活动和不活动的虚拟机
<code>virsh list --transient</code>	列出所有临时的虚拟机
<code>virsh list --persistent</code>	列出所有永久的虚拟机

示例效果

查看所有运行中的虚拟机

```
# virsh list
```

Id	Name	State

查看所有的虚拟机，无论运行与否

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

Id	Name	State

-	CentOS-8-x86_64	shut off
---	-----------------	----------

注意：

虚拟机的每次重启，虚拟机的id都会自动的变化，因为虚拟机的运行本质上是操作系统的一个进程而已

创建

简介

创建虚拟机的方式除了初始方法，还有一种经常使用的方法--使用配置文件，而使用配置文件的话，会有两种创建的方式：

临时创建

- 借助于现成的配置文件和之前创建的虚拟机数据，创建一个新虚拟机，他会自动清除
- 步骤：配置文件-创建虚拟机-查看
- 命令：virsh create file_name.xml

正规创建

- 借助于现成的配置文件在虚拟机管理程序中注册一个新的虚拟机，他不会自动清除
- 步骤：配置文件-注册虚拟机-开启虚拟机-查看
- 命令：virsh define file_name.xml

关键点：

尽量拷贝一个完整的虚拟机配置文件，然后修改配置文件中的名字，这个配置文件名称和配置文件中的name属性要一致，且名字唯一

临时创建

拷贝文件

```
cd /etc/libvirt/qemu
cp CentOS-8-x86_64.xml CentOS-8-x86_64-1.xml
```

修改配置文件

```
# vim CentOS-8-x86_64-1.xml

...
9  <name>CentOS-8-x86_64-1</name>
10 <uuid>db9e17ea-6e9f-4251-bb04-47985d442ce6</uuid>
...
```

注意：

主要修改的地方就是：虚拟机的名字，uuid

虚拟磁盘不需要修改，如果要修改的话，一个新建的虚拟磁盘里面因为没有boot文件，就会报错网卡什么的尽量不要更改，更改后即使可以，但是启动不成功

创建并启动虚拟机

```
virsh create /etc/libvirt/qemu/CentOS-8-x86_64-1.xml
```

查看效果


```
virsh list --all
```

可以看到:

新建的虚拟机已经成功了, 而且启动的非常快

永久创建

配置文件

操作同上

注册虚拟机

```
virsh define /etc/libvirt/qemu/CentOS-8-x86_64-2.xml
```

查看效果

```
virsh list --all
```

结果显示:

默认创建的虚拟机是关闭着的

开启虚拟机

```
virsh start CentOS-8-x86_64-2
```

注意:

由于多个虚拟机使用同一个磁盘文件, 所以, 同一时间点内, 只允许启动1个虚拟机

查看

```
virsh list --all
```

登录

当我们虚拟机过多的时候, 如果想用vnc来连接的话, 就不够了, 但我还想用vnc怎么办?

思路:

执行 'virsh domdisplay 虚拟机名称' 来获取虚拟机的地址信息

示例

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

Id	Name	State
4	CentOS-8-x86_64-2	running
-	CentOS-8-x86_64	shut off

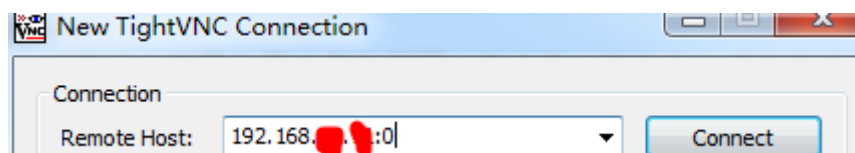
```
# virsh domdisplay CentOS-8-x86_64
error: Domain is not running
```

```
# virsh domdisplay CentOS-8-x86_64-2
vnc://localhost:0
```

注意:

:后面的数字n代表虚拟机启动的端口号, 特指vnc端口号5900+n

vnc查看连接方式:



需求

虽然我们可以使用vnc等方式连接到虚拟机中进行各种操作，但是，对于有些简单操作，使用vnc比较麻烦，所以我们这里学一种轻巧的登录虚拟机的方法：**console**

注意：

默认情况下，新安装的虚拟机，是无法通过**virsh console** 命令连入虚拟机中的，这时我们需要开启虚拟机的**console**功能。

```
使用vnc登录虚拟机，添加ttys0的许可，然后重启虚拟机
grubby --update-kernel=ALL --args="console=ttys0"
reboot
```

思路

命令帮助：

```
virsh console --help
```

命令格式：

```
virsh console 虚拟机名称
```

简单实践

在宿主机终端登录虚拟机

```
virsh list
```

```
virsh console CentOS-8-x86_64
```

输入用户名和密码后验证

```
ip addr
```

注意：

按 **ctrl+] 组合键**退出**virsh console**

小结

挂起、恢复

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

需要对虚拟机进行临时性的 暂停状态 和 恢复状态

思路

挂起:

查看帮助: `virsh suspend --help`

命令格式: `virsh suspend vmhost_name`

恢复:

查看帮助: `virsh resume --help`

命令格式: `virsh resume vmhost_name`

简单实践

挂起实践

查看效果

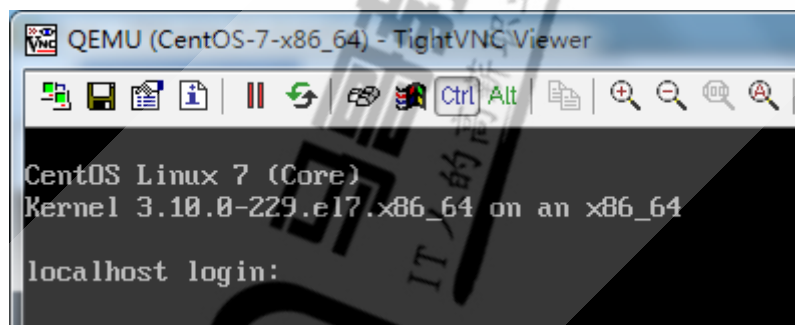
```
virsh list --all
```

挂起虚拟机

```
virsh suspend CentOS-8-x86_64
```

查看效果

```
virsh list --all
```



效果显示: vnc界面卡死了, 但是虚拟机并没有关机

恢复实践

查看效果

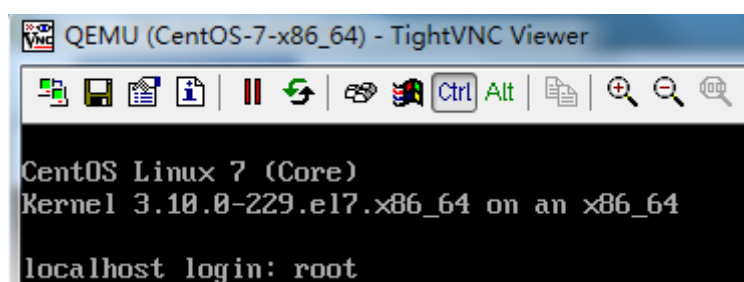
```
virsh list --all
```

挂起虚拟机

```
virsh resume CentOS-8-x86_64
```

查看效果

```
virsh list --all
```



可以看到：vnc界面里面恢复了，而且可以进行操作

小结

关闭、删除

学习目标：

这一节，我们从 关闭实践、删除实践、小结 三个方面来学习。

关闭实践

需求

虚拟机运行时间太长了，向休息一下，那就关闭咯

方法

`virsh shutdown 虚拟机名称`

注意：

`shutdown`方法依赖于`acpid`服务，所以要保证我们的操作系统有`acpid`服务

对于`ubuntu`来说，它会自动安装相关软件并运行该服务

对于`centos`来说，它需要手工安装并启动

```
yum install acpid -y
systemctl start acpid
systemctl enable acpid
```

简单实践

关闭虚拟机

```
virsh shutdown CentOS-8-x86_64
```

检查

```
virsh list --all
```

注意：

使用`shutdown`关闭的时候，一定要避免资源冲突，特别是基于同一个配置文件创建出来的虚拟机。

如果出现这种情况的话，`shutdown`不管用，使用`kill`方式关闭或者强制删除方法，示例如下：

关闭虚拟机

```
ps aux | grep CentOS
kill 6589
```

检查

```
virsh list --all
```

删除实践

简介

删除虚拟机有两种方法：正常删除和强制删除

正常删除

不影响虚拟机的当前运行状态的删除。

- 删除其配置文件，`virsh`状态表中存在，关闭虚拟机后，状态表中亦删除

命令格式：`virsh undefine` 虚拟机名称

强制删除

将一个运行中的虚拟机强制关闭。

- `virsh`状态表中直接删除，配置文件不受影响。

命令格式：`virsh destroy` 虚拟机名称

正常删除

删除虚拟机

```
virsh list --all
virsh undefine CentOS-8-x86_64
virsh list --all
ls /etc/libvirt/qemu
```

关闭虚拟机

```
virsh shutdown CentOS-8-x86_64
virsh list --all
```

强制删除

强制关闭虚拟机

```
virsh list --all
virsh destroy CentOS-8-x86_64-1
```

检查效果

```
virsh list --all
ls /etc/libvirt/qemu
```

小结

开机自启动

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

默认情况下，创建好的kvm虚拟机不会随宿主机开启而启动，特殊的工作场景中，为了简便工作，我们会将其设置成开机自启动功能。

命令

开启自启动命令格式：

帮助: `virsh autostart --help`

开启: `virsh autostart 虚拟机名称`

关闭: `virsh autostart --disable 虚拟机名称`

注意：

开机自启动的操作，仅限于被注册的虚拟机，临时虚拟机没有该功能。

简单实践

开启自启动

检查配置文件目录效果

```
# ls /etc/libvirt/qemu
CentOS-8-x86_64.xml  networks
```

设置开机自启动

```
virsh autostart CentOS-8-x86_64
```

检查效果

```
# tree /etc/libvirt/qemu
/etc/libvirt/qemu
├── autostart # 多了一个autostart目录
│   └── CentOS-8-x86_64.xml -> /etc/libvirt/qemu/CentOS-8-x86_64.xml
├── CentOS-8-x86_64.xml
└── networks
    ...

3 directories, 4 files
```

方法二：

```
virsh list --autostart --all
```

取消开机自启动

取消开机自启动

```
virsh autostart --disable CentOS-8-x86_64
```

检查效果

```
# tree /etc/libvirt/qemu
/etc/libvirt/qemu
├── autostart
├── CentOS-8-x86_64.xml
└── networks
    ...

3 directories, 3 files
```

方法二:

```
virsh list --no-autostart --all
```

小结

备份、编辑

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

手工修改虚拟机的配置，限制太多。学习虚拟机配置文件的专用的操作功能。

基本操作

备份

作用：文件内容的版本保存

格式：`virsh dumpxml 虚拟主机名 > backup_vmhost.file`

注意：备份文件如果想在本地使用的话，需要vim修改一下主机名和uuid，才可以

编辑

作用：文件内容的编辑动作

格式：`virsh edit 虚拟主机名`

注意：备份的虚拟机文件如果想要使用virsh edit编辑的话，必须要保证该文件已经注册到虚拟机管理设备中了

简单实践

备份实践

查看效果

```
cd /etc/libvirt/qemu
ls
```

备份文件

```
virsh dumpxml CentOS-8-x86_64 > CentOS-backup.xml
```

检查效果

```
ls
diff CentOS-8-x86_64.xml CentOS-backup.xml
```

注意：

备份文件如果想在本地使用的话，需要vim修改一下主机名和uuid，才可以

修改配置文件

```
# vim CentOS-backup.xml
...
 9   <name>CentOS-8-x86_64-2</name>
10   <uuid>db9e17ea-6e9f-4251-bb04-47985d442ce7</uuid>
...
```

注意：仅仅修改name和uuid即可

创建虚拟机

```
virsh define CentOS-backup.xml
```

查看效果

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

Id	Name	State
-	CentOS-8-x86_64-2	shut off
...		

可以看到：

创建的虚拟机默认情况下是关闭状态的。

编辑虚拟机

```
# virsh edit CentOS-8-x86_64-2
```

原内容：

```
<graphics type='vnc' port='-1' autoport='yes' listen='0.0.0.0'>
  <listen type='address' address='0.0.0.0' />
</graphics>
```

修改后内容：

```
<graphics type='vnc' port='8888' listen='0.0.0.0' passwd='654321'>
  <listen type='address' address='0.0.0.0' />
</graphics>
```

保存文件后，开启虚拟机

```
virsh start CentOS-8-x86_64-2
```

注意：

上面测试的信息是给 vnc 连接时候登录用的，不是给 virsh console 连接用的，因为virsh console 不走vnc

小结

存储管理

存储池基础

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

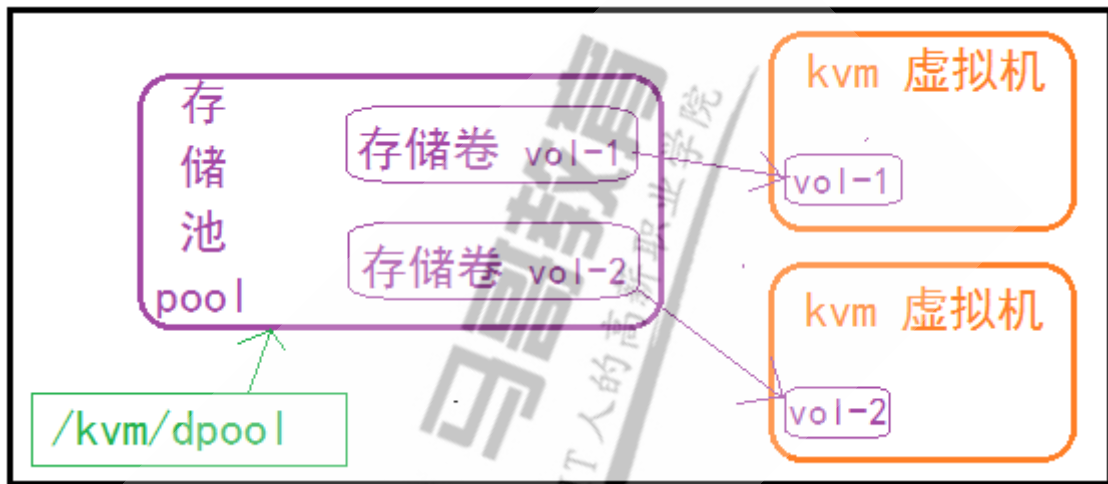
为了使虚拟机获得更强大的后端存储能力，**libvirt** 提供了对各种各样的存储介质支持，包括本地文件系统，网络文件系统，**iSCSI**，**LVM** 等多种后端存储系统。在虚拟机中主要以存储池(**pool**)与存储卷(**volume**)的形式存在。

libvirt 中的存储管理独立于虚拟机管理。也就是存储池和存储卷的操作独立于虚拟机的操作存在，即使是没有虚拟机的场景。工作中我们可以当虚拟机需要存储资源时再进行灵活分配。

存储池

KVM平台以存储池的形式对存储进行统一管理。它可以理解作为一种映射关系，即将宿主机上的存储空间形成可被**KVM**使用的逻辑存储池，以方便虚拟主机存储资源的统一管理。

宿主机的存储空间，可以是本地的、网络的等**kvm**支持的任意形式。



默认**KVM**将这些存储池的配置文件存放于 `/etc/libvirt/storage`，因为我在创建虚拟机的时候，涉及到两个目录，所以这两个目录在虚拟机创建的时候，自动有存在了。

通过**virsh**中**pool**命令能够查看、创建、激活、注册、删除存储池

```
virsh --help | grep pool
```

基本操作

信息查看：

存储列表: `virsh pool-list`

具体信息: `virsh pool-info pool_name`

创建存储池

常见两种创建方法，基于文件夹和文件系统，我们主要介绍基于文件夹的方式。

定义永久池+创建池: `virsh pool-define-as + virsh pool-build`

创建临时池+启动池: `virsh pool-create-as`

命令格式一: `virsh pool-define-as vmware_pool --type dir --target /kvm/images`

注释:

`vmware_pool`

定义的存储池名称

`--type dir`

我们的类型是一个dir

`--target /kvm/images` 存储池的在宿主机上的目录是/kvm/images

命令格式二: `virsh pool-create-as --name vmware_pool --type dir --target /kvm/images`

注释: 直接创建并启动一个存储池

<code>vmware_pool</code>	定义的存储池名称
<code>--type dir</code>	我们的类型是一个dir
<code>--target /kvm/images</code>	存储池的在宿主机上的目录是/kvm/images

其他的创建命令

基于分区创建存储池:

`virsh pool-define-as <存储池名> fs --source-dev <分区名称> --target <挂载目录>`

基于磁盘创建存储池:

`virsh pool-define-as <存储池名> disk --source-dev <磁盘名称> --source-format gpt --target <挂载目录>`

基于lvm创建存储池:

`virsh pool-define-as <存储池名> logical --source-name <vg名称> --target <挂载目录>`

基于iscsi创建存储池:

`virsh pool-define-as <存储池名> iscsi --source-host <存储主机ip> --source-dev <存储目标> --target <挂载目录>`

基于nfs创建存储池:

`virsh pool-define-as <存储池名> netfs --source-host <nfs主机ip> --source-path <nfs共享目录> --target <挂载目录>`

简单实践

永久存储池

创建目录

`mkdir /kvm/images -p`

创建存储池

`virsh pool-define-as magedu_dpool --type dir --target /kvm/images`

`virsh pool-build magedu_dpool`

检查效果

`virsh pool-list --persistent`

`ls /etc/libvirt/storage/`

可以看到:

`pool-define-as`创建的存储池没有启动,在存储目录中有定义文件

临时存储池

创建目录

```
mkdir /magedu/images -p
```

创建存储池

```
virsh pool-create-as magedu_cpool --type dir --target /magedu/images
```

检查效果

```
virsh pool-list --transient  
ls /etc/libvirt/storage/
```

可以看到:

pool-create-as创建的存储池启动成功了,在存储目录中没有定义文件

小结

启动、关闭、取消

学习目标:

这一节,我们从 基础知识、简单实践、小结 三个方面来学习。

启动实践

需求

启动存储池

语法

启动命令:

```
virsh pool-start 存储池名称
```

开机自启动:

```
virsh pool-autostart 存储池名称
```

实践: 刚才创建的 magedu_dpool存储池没有启动,现在我们启动它,使它生效。

启动创建的 magedu_dpool 存储池

```
virsh pool-start magedu_dpool
```

检查效果

```
virsh pool-list
```

可以看到:

这个时候,启用仅仅是让存储池生效了,但是还没有自动运行,如果要自动运行的话,还需要另一个命令

实践: 设置为开机自启动

设置刚才启用的存储池自动运行。

```
virsh pool-autostart magedu_dpool
```

检查效果

```
virsh pool-list
```

注意：

只有定义好的存储池，才可以设置为开机自启动，临时的存储池没有该功能。

关闭实践

需求

存储池不用了，关闭一下

语法

关闭命令：

```
virsh pool-destroy 存储池
```

实践：关闭存储池

关闭存储池

```
virsh pool-destroy magedu_dpool
```

```
virsh pool-destroy magedu_cpool
```

检查效果

```
virsh pool-list
```

取消实践

需求

存储池不需要的时候，为了节省空间，可以取消(移除)存储池。

取消存储池主要有两部分组成：删除存储目录、删除存储配置文件

注意：

这两步都要求我们的存储对象在**virsh pool-list**的列表中

语法

删除存储目录：

```
virsh pool-delete 存储池
```

删除存储配置文件：

```
virsh pool-undefine 存储池
```

实践：取消存储池

取消存储池目录

```
virsh pool-delete magedu_dpool
```

检查

```
ls /kvm
```

检查效果

```
virsh pool-list
```

删除存储池配置文件

```
virsh pool-undefine magedu_dpoo1
```

检查

```
ls /etc/libvirt/storage/
```

检查

```
virsh pool-list
```

小结

存储卷基础

学习目标：

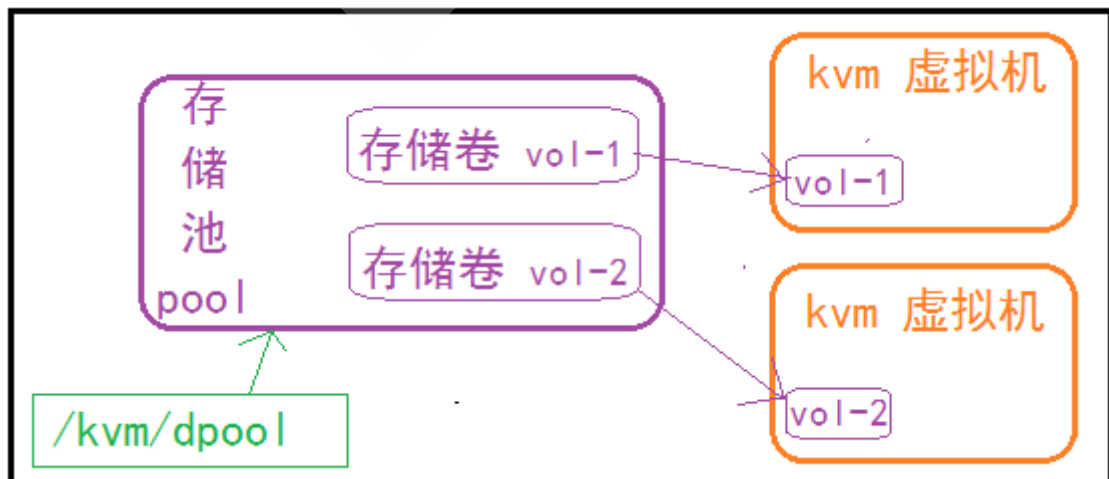
这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

有了存储池，代表我们有存储数据的基本前提了，但是还不能正常使用，需要借助于存储卷才可以。

简介



存储卷是一种可以分配给虚拟机使用的存储设备。在虚拟机中与一个挂载点对应，而物理上可以是一个虚拟机磁盘文件或一个真实的磁盘分区。

它一般存放在一个专用的存储池中来使用。

存储卷是结合存储池来使用的，所以我们就先来做存储卷的准备工作：创建存储池

```
virsh pool-define-as magedu_pool --type dir --target /kvm/images
virsh pool-start magedu_pool
```

命令解析

通过virsh中vol命令能够查看、创建、激活、注册、删除存储卷

```
virsh --help | grep vol
```

创建存储卷主要有两种方式：

创建新的：virsh vol-create-as

查看存储卷信息

查看列表：virsh vol-list 存储池名称

查看属性：virsh vol-info <存储卷> <存储池>

查看配置：virsh vol-dumpxml <存储卷> <存储池>

新建命令

命令格式：

```
virsh vol-create-as --pool magedu_pool --name magedu.img --capacity 2G --
allocation 1G --format raw
```

注释：

virsh vol-create-as	命令
--pool magedu_pool	指定存储池
--name magedu.img	支持创建的磁盘卷
--capacity 10G	存储卷的容量
--allocation 1G	存储卷的初始分配大小
--format qcow2	存储卷的格式，默认格式是raw

简单实践

新建示例

创建存储卷

```
virsh vol-create-as --pool magedu_pool --name magedu.img --capacity 2G --
allocation 1G --format raw
```

查看位置

```
ll /kvm/images
```

存储池信息查看

查看存储池的存储卷列表

```
virsh vol-list magedu_pool
```

查看存储卷信息

```
virsh vol-info magedu.img magedu_pool
```

查看存储卷配置信息

```
virsh vol-dumpxml magedu.img magedu_pool
```

小结

挂载、卸载

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

虚拟机在需要更多存储空间或者精简存储空间的时候，就遇到了存储卷的 挂载和卸载动作

方法指令

挂载动作：

```
virsh attach-disk <虚拟机> <存储卷> <挂载设备>
```

注意：挂载设备可以是vdx或者sdx，x是从b开始的字母

卸载动作

```
virsh detach-disk <虚拟机> <挂载设备>
```

注意：挂载格式可以是vdx或者sdx

简单实践

挂载实践

挂载操作

```
virsh list --all
```

```
virsh attach-disk CentOS-8-x86_64 /kvm/images/magedu.img vda
```

检查效果

```
virsh console CentOS-8-x86_64
```

```
fdisk -l
```

使用磁盘

```
mkfs.ext4 /dev/vdb
```

```
mkdir /magedu
```

```
mount /dev/vdb /magedu/  
df -h  
dd if=/dev/zero of=/magedu/test.1.txt bs=1M count=200  
ll /magedu/
```

卸载实践

卸载目录

```
umount /magedu/  
Ctrl + ]
```

卸载磁盘

```
virsh detach-disk CentOS-8-x86_64 vda  
virsh console CentOS-8-x86_64  
fdisk -l
```

检查磁盘效果

```
virsh vol-info magedu.img magedu_pool
```

小结

数据实践

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

在我们使用数据卷操作的时候，不可避免的涉及到 数据导出、数据清除、数据导入、容量变动 等操作。

语法格式

数据导出

```
virsh vol-download <存储卷名称> <导出文件名> --pool <存储池>
```

数据清除

```
virsh vol-wipe <存储卷名称> --pool <存储池>
```

数据导入

```
virsh vol-upload <存储卷名称> <导出文件名> --pool <存储池>
```

容量变动

```
virsh vol-resize <已存在的存储卷名称> <新的容量大小> --pool <存储池>
```

注意：容量的变动主要针对的是磁盘容量，而不是磁盘里面存储的数据

简单实践

导出实践

导出卷中数据

```
virsh vol-download magedu.img file.txt --pool magedu_pool  
ll file.txt
```

结果显示:

将一个磁盘文件的所有信息都备份出来了, 容量大小与数据卷大小一致

清除实践

清除卷中数据

```
virsh vol-info magedu.img magedu_pool  
virsh vol-wipe magedu.img --pool magedu_pool  
virsh vol-info magedu.img magedu_pool
```

导入实践

导入卷中数据

```
virsh vol-info magedu.img magedu_pool  
virsh vol-upload magedu.img file.txt --pool magedu_pool  
virsh vol-info magedu.img magedu_pool
```

注意:

如果没有数据卷, 则无法导入数据

容量变动

查看当前卷信息

```
virsh vol-info magedu.img magedu_pool
```

调整卷容量后查看卷信息

```
virsh vol-resize magedu.img 3G --pool magedu_pool  
virsh vol-info magedu.img magedu_pool  
virsh vol-resize magedu.img 5G --pool magedu_pool  
virsh vol-info magedu.img magedu_pool
```

默认情况下, 容量不支持扩容

```
# virsh vol-resize magedu.img 1G --pool magedu_pool  
error: Failed to change size of volume 'magedu.img' to 1G  
error: invalid argument: Can't shrink capacity below current capacity unless  
shrink flag explicitly specified
```

小结

克隆、删除

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

当我们定制好应用数据卷之后，可能需要快速的复制一个，这就用到了数据卷的克隆操作；当我们数据卷的使命结束后，我们就要删除数据卷了。

语法解析

克隆存储卷

```
virsh vol-clone <已存在的存储卷名称> <新的存储卷名称> --pool <存储池>
```

删除存储卷

```
virsh vol-delete <已存在的存储卷> <存储池>
```

简单实践

克隆实践

克隆一个新的存储卷

```
virsh vol-clone magedu.img magedu-2.img magedu_pool
```

查看存储卷列表

```
virsh vol-list magedu_pool
```

```
ls /kvm/images/
```

查看两个存储卷信息

```
virsh vol-info magedu.img magedu_pool
```

```
virsh vol-info magedu-2.img magedu_pool
```

删除实践

删除存储卷

```
virsh vol-delete magedu-2.img magedu_pool
```

查看存储卷列表

```
virsh vol-list magedu_pool
```

小结

网络管理

网络基础

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

网络模型

kvm类型的虚拟机，默认情况下使用的是nat网络，同一网络模型下创建的多虚拟机之间可以彼此通信，这种网络模型与外界网络进行通信依赖于nat服务，nat服务正常的话，该模型的网络是通过数据包的转换来达到通信的效果的，在网络流量比较大的场景下，该模式的网络模型会成为虚拟机的瓶颈。

生产中我们一般采用桥接的网络模型，这种模型是借助于宿主机的网卡设备，直接与外网进行通信，效果很好。



对于kvm来说，它有两种网络：永久网络和临时网络

配置流程

前提：保证宿主机网络正常，网络服务使用network
首先：生成新的桥接设备
其次：关联桥设备和物理网卡设备
然后：清空物理网卡地址，启用桥设备地址，设置网络网关为桥设备地址
最后：虚拟机使用桥设备

简单实践

基本配置

kvm的网络配置目录

```
# tree /etc/libvirt/qemu/networks/  
/etc/libvirt/qemu/networks/  
├── autostart  
│   └── default.xml -> ../default.xml  
└── default.xml
```

1 directory, 2 files

查看桥接网络

```
[root@controller ~]# ifconfig vnet0  
vnet0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
        inet6 fe80::fc54:ff:fe8d:a117 prefixlen 64 scopeid 0x20<link>
```

```
ether fe:54:00:fd:a1:17 txqueuelen 1000 (Ethernet)
...
```

查看所有网络相关命令

```
virsh -h | grep net-
```

网络查看

查看网络列表

```
virsh net-list
```

```
virsh net-list --all
```

查看网络信息

```
virsh net-info default
```

查看网络配置

```
virsh net-dumpxml default
```

小结

网络操作

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

如果我们不想用默认的网络的话，可以自己定义一个专用的网络。

命令解析

创建网络

临时: `virsh net-create <xml配置文件>`

永久: `virsh net-define <xml配置文件>`

关闭网络

```
virsh net-destroy <网络名称>
```

开启网络

```
virsh net-start <网络名称>
```

删除网络

```
virsh net-undefine <网络名称>
```

简单实践

创建临时网络

设置配置文件

```
# cp default.xml magedu-linshi.xml
# vim magedu-linshi.xml
<network>
  <name>magedu-linshi</name>
  <uuid>62acc0bb-6d0f-43e8-9100-a44a26651f22</uuid>
  <forward mode='nat' />
  <bridge name='virbr1' stp='on' delay='0' />
  <mac address='52:54:00:74:58:0a' />
  <ip address='192.168.123.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.123.2' end='192.168.123.254' />
    </dhcp>
  </ip>
</network>
```

创建临时网络

```
virsh net-list --all
virsh net-create magedu-linshi.xml
```

查看效果

```
virsh net-list --all
virsh net-list --transient
ifconfig
```

创建临时网络

设置配置文件

```
# cp default.xml magedu-yongjiu.xml
# vim magedu-yongjiu.xml
<network>
  <name>magedu-yongjiu</name>
  <uuid>62acc0bb-6d9f-44e8-9100-a44a26651f22</uuid>
  <forward mode='nat' />
  <bridge name='virbr2' stp='on' delay='0' />
  <mac address='52:54:00:74:57:1a' />
  <ip address='192.168.124.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.124.2' end='192.168.124.254' />
    </dhcp>
  </ip>
</network>
```

创建永久网络

```
virsh net-list --all
virsh net-define magedu-yongjiu.xml
virsh net-list --all
virsh net-start magedu-yongjiu
```

查看效果

```
virsh net-list --all
```

```
virsh net-list --persistent  
ifconfig
```

关闭实践

```
停止网络  
virsh net-destroy default  
virsh net-list --all
```

开启实践

```
开启网络  
virsh net-start default  
virsh net-list --all
```

删除实践

删除网络需要有两步：取消注册+关闭网络

```
关闭网络  
virsh net-undefine default  
virsh net-list --all
```

```
删除网络  
ls /etc/libvirt/qemu/networks  
virsh net-destroy default  
ifconfig
```

小结

自定义网络

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

准备工作

基本信息查看

```
命令帮助  
brctl --help  
  
查看网络基本信息  
ip addr  
route -n
```

准备工作

```
安装桥操作工具: bridge-utils
apt install bridge-utils -y
apt-file list bridge-utils

dns解析
# grep namese /etc/resolv.conf
nameserver 192.168.8.2
```

桥接设备

基本实践

```
创建桥接网卡设备
# brctl addbr br0
# brctl stp br0 on

删除物理网卡的ip地址
# ifconfig ens33 0 up

关联桥接设备和物理网卡设备
# brctl addif br0 ens33

给桥接网卡配置ip地址并启动
# ifconfig br0 192.168.8.14/24 up

给桥接网卡配置路由
# route add default gw 192.168.8.2
注意:
```

操作br0的另一种方法是 `ip set dev br0 up/down`

如果将命令分开执行的话，则会在关联桥接网卡和物理网卡的时候造成断网现象，所以建议将这几条命令同时执行或者使用脚本执行命令

方法二脚本变种：

```
#!/bin/bash
brctl addbr br0
brctl stp br0 on
ifconfig ens33 0 up
brctl addif br0 ens33
ifconfig br0 192.168.8.14/24 up
route add default gw 192.168.8.2
```

使用设备

应用方法

```
查看虚拟机的网卡默认配置
virsh edit CentOS-8-x86_64

修改虚拟机的网卡配置
# virsh edit CentOS-8-x86_64
```

```

...
69     <interface type='bridge'>
70         <mac address='52:54:00:15:a5:13' />
71         <source bridge='br0' />
72         <model type='virtio' />
73         <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0' />
74     </interface>
...

```

注意:

修改了三处地方:

type后面的network修改为了bridge

source后面的network修改为了bridge,network后面的default修改为了br0

重启虚拟机

```
virsh shutdown CentOS-8-x86_64
```

```
virsh start CentOS-8-x86_64
```

通过vnc连接到虚拟机

因为默认的虚拟机使用的是nat模型,而net模型有专用的dnsmasq软件提供dns服务,我们现在使用桥接模型,所以我们需要自己来设置虚拟机的网卡信息。

查看虚拟机网络信息

```
ip add
```

编辑网卡配置

```
[root@localhost ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
```

```
...

```

```
BOOTPROTO=static
```

```
...

```

```
DEVICE=eth0
```

```
ONBOOT=yes
```

```
IPADDR=192.168.8.16
```

```
NETMASK=255.255.255.0
```

```
GATEWAY=192.168.8.2
```

```
[root@localhost ~]# cat /etc/resolv.conf
```

```
# Generated by NetworkManager
```

```
nameserver 192.168.8.2
```

重启网络 或者 重启虚拟机

注意: centos8环境下重启网络的命令有些繁琐,不能使用 `systemctl restart network`

```
nmcli c reload
```

```
nmcli c up eth0
```

```
nmcli d reapply eth0
```

```
nmcli d connect eth0
```

宿主机连接虚拟机

```
ssh 192.168.8.16
```

问题解析

常见报错:

错误: 开始域 magedu-web 失败

错误: 不支持的配置: Unable to find security driver for model selinux

解决办法:

将虚拟机中的"`<seclabel type='dynamic' model='selinux' relabel='yes'/>`" 删除即可

小结

镜像管理

磁盘基础

学习目标:

这一节, 我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

磁盘模式

根据存储数据方式的不同可以分为两种格式, 一种是稀疏模式, 一种是全镜像模式, 全镜像模式无法做快照, IO层面是有qemu模拟的, CPU和内存是有KVM实现的

raw:

指定多大就创建多大, 直接占用指定大小的空间:

老牌的格式了, 性能上来说的话还是不错的。由于原生的裸格式, 不支持snapshot。在虚拟机迁移方面还是有很大的限制。

qcow2:

是openstack默认也是比较推荐的格式, 将差异保存在一个文件, 文件比较小而且做快照也比较小, 空间是动态增长的。

现在比较主流的一种虚拟化镜像格式, 支持快照功能。

raw 格式

基础镜像

变化内容

确认效果

查看虚拟机支持的磁盘格式种类

`qemu-img --help`

支持的种类

Supported formats: vvfat vpc vmdk vhdx vdi ssh sheepdog rbd raw host_cdrom host_floppy host_device file qed qcow2 qcow parallels nbd iscsi gluster dmg tftp ftps ftp https http cloop bochs blkverify blkdebug

简单实践

基本语法

创建磁盘

```
qemu-img create [-q] [-f fmt] [-o options] filename [size]
```

查看某个具体磁盘格式的帮助信息：

```
qemu-img create -f qcow2 -o ? file.qcow
```

注意：在“-o”选项中各个选项用逗号来分隔

查看磁盘

```
qemu-img info 磁盘名称
```

磁盘转换

```
qemu-img convert -f 原格式 旧磁盘名称 -o 转换后格式 新磁盘名称
```

创建实践：创建一个3G大小的raw格式的虚拟磁盘CentOS-8-x86_64.raw

基本动作执行命令

```
qemu-img create -f raw /opt/CentOS-8-x86_64.raw 3G
```

创建一个基于前后端模式的qcow2的磁盘文件

```
qemu-img create -f qcow2 -o backing_file=/kvm/images/magedu.img test-1.qcow2 3G
```

查看实践

```
ll /opt/
```

```
qemu-img info /opt/CentOS-8-x86_64.raw
```

转换实践

raw磁盘转换qcow2

```
cd /opt/
```

```
qemu-img convert -f raw CentOS-8-x86_64.raw -o qcow2 CentOS-8-x86_64.qcow2
```

检查效果

```
ll -h /opt/
```

```
qemu-img info CentOS-8-x86_64.qcow2
```

刚才我们生成了一个新的磁盘，现在虚拟机使用新的磁盘

```
# virsh shutdown CentOS-8-x86_64
```

```
# virsh edit CentOS-8-x86_64
```

```
37     <disk type='file' device='disk'>
```

```
38     <driver name='qemu' type='qcow2' />
```

```
39     <source file='/opt/CentOS-8-x86_64.qcow2' />
```

启动虚拟机

```
virsh start CentOS-7-x86_64
```

注意：由于是一个新的磁盘，所以系统在启动的时候，会导致失败，效果如下

```
SeaBIOS (version 1.13.0-1ubuntu1.1)
Machine UUID 5df5bdf4-b414-43b8-8337-4059dca2b2cc

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+3FF8C800+3FECC800 CA00

Booting from Hard Disk...
Boot failed: not a bootable disk

No bootable device.
```

小结

克隆

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

在某些特殊场景下，我们需要批量安装多台虚拟机，目前我们学习的虚拟机安装的方式是逐一的，效率太低了，有什么方法能解决这个痛点呢？ -- 虚拟机克隆

实现方式

kvm虚拟机的克隆分为两种情况。

- (1) KVM主机本机虚拟机直接克隆。
- (2) 通过复制配置文件与磁盘文件的虚拟机复制克隆(适用于异机的静态迁移)。

准备环境

查看虚拟主机

```
virsh list --all
```

查看配置文件

```
ll /etc/libvirt/qemu/CentOS-8-x86_64.xml
```

还原虚拟机的网络配置配置信息

```
<interface type='network'>
  <mac address='52:54:00:74:57:0a' />
  <source network='default' />
  <model type='virtio' />
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0' />
</interface>
```

查看磁盘文件

```
ll /opt/
```

简单实践

命令解析

镜像克隆

```
virt-clone -o <被克隆的虚拟机名称> -n <生成的虚拟机名称> -f <新的磁盘文件>
virt-clone -o <被克隆的虚拟机名称> --auto-clone
```

镜像实践

关闭虚拟机

```
virsh shutdown CentOS-8-x86_64
```

克隆镜像

```
virt-clone -o CentOS-8-x86_64 -n CentOS-8-clone -f /opt/CentOS-8-clone.img
```

查看虚拟机相关文件:

```
virsh list --all
ls /opt/
ls /etc/libvirt/qemu
```

查看文件区别

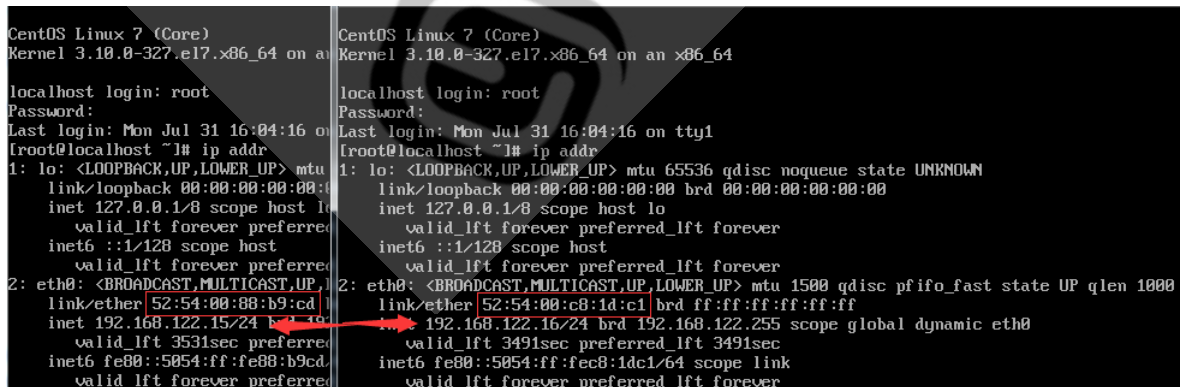
```
diff CentOS-8-clone.xml CentOS-8-x86_64.xml
```

可以看到:

我们克隆的虚拟机创建了新的磁盘文件/配置文件/虚拟机
克隆的虚拟机和源虚拟机的配置文件主要改变的方在: name/uuid/source/mac 这四方面

启动虚拟机查看效果

```
virsh start CentOS-8-clone
virsh start CentOS-8-x86_64
```



```
CentOS Linux 7 (Core)
Kernel 3.10.0-327.el7.x86_64 on an x86_64

localhost login: root
Password:
Last login: Mon Jul 31 16:04:16 on tty1
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host localhost
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host 
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 52:54:00:88:b9:cd brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.15/24 brd 192.168.122.255 scope global dynamic eth0
        valid_lft 3531sec preferred_lft 3491sec
    inet6 fe80::5054:ff:fe88:b9cd/64 scope link
        valid_lft forever preferred_lft forever

CentOS Linux 7 (Core)
Kernel 3.10.0-327.el7.x86_64 on an x86_64

localhost login: root
Password:
Last login: Mon Jul 31 16:04:16 on tty1
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host localhost
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host 
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 52:54:00:c8:1d:c1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.16/24 brd 192.168.122.255 scope global dynamic eth0
        valid_lft 3491sec preferred_lft 3491sec
    inet6 fe80::5054:ff:fec8:1dc1/64 scope link
        valid_lft forever preferred_lft forever
```

小结

资源管理

cpu简介

学习目标:

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

在虚拟机不关机的情况下，对CPU进行调整操作叫热调整。它是在Redhat7.0以后才出现的功能，所以要使用这功能，那必须要求KVM宿主机和虚拟机规格必须一致，而且在7.0+版本。

目前cpu热调整命令，只能增加，不能减少，想要减少的话，可以进入虚拟机中，关闭某个使用的CPU即可。

公有云上的虚拟机为什么不支持cpu的热调整

- 1、因为客户创建的虚拟机类型不一致，不满足热添加和热删除的前提条件，所以为了管理的方便就统一不允许了
- 2、因为创建虚拟机会有个调度的问题，创建的虚拟机的命令，会将创建的虚拟机调度到某一台宿主机上，具体调度到哪台宿主机上，没有办法保证，由于受宿主机的配置影响，所以热添加和热删除的功能没有办法保证

常见命令

热调整

```
virsh setvcpus <虚拟机名称> <cpu个数> --live
```

注意：

默认后面会有 `--live` 属性，标识热调整，默认不会更改后端的配置

如果需要更改后端的配置，需要使用 `--config`

cpu绑定

```
virsh vcpupin <虚拟机名称> <虚拟机CPU号> <宿主CPU号>
```

cpu重置

```
virsh shutdown <虚拟机名称>
```

```
virsh start <虚拟机名称>
```

准备工作

查看宿主机的cpu个数

```
root@python-auto:~# lscpu
```

```
架构: x86_64
CPU 运行模式: 32-bit, 64-bit
字节序: Little Endian
Address sizes: 42 bits physical, 48 bits virtual
CPU: 4
在线 CPU 列表: 0,1
离线 CPU 列表: 2,3
...
```

可以看到：

我们的宿主机的cpu个数为4个，目前在线的是 0、1 两个，离线的是 2，3两个

宿主机查看所有的cpu状态

```
# cat /sys/devices/system/cpu/cpu1/online
```

```
1
```

```
# cat /sys/devices/system/cpu/cpu3/online
```

```
0
```

注意：

cpu0里面有可能没有online，这不影响，如果非要说的话，可以手工添加个

可以看到:

宿主机的2个cpu都是处于工作状态中

查看虚拟机当前的cpu状态

```
# virsh edit CentOS-8-x86_64
...
6    <vcpu placement='static'>1</vcpu>
```

调整配置, 让cpu支持动态调整功能

关闭虚拟机

```
virsh shutdown CentOS-8-x86_64
```

编辑虚拟机配置

```
# vim CentOS-8-x86_64.xml 或者 virsh edit CentOS-8-x86_64
```

原内容

```
<vcpu placement='static'>1</vcpu>
```

修改后

```
<vcpu placement='static' current='1'>4</vcpu>
```

注意:

意思就是, 当前的cpu可以动态调整, 当前的cpu个数为1, 可调整的最大个数为4
经过测试, 我的虚拟机可以支持255个, 但是最大只能设置253个。

如果操作异常, 有可能导致以下异常情况

```
error: operation failed: Failed to query numad for the advisory nodeset
```

启动虚拟机后查看效果

```
virsh start CentOS-8-clone
virsh dominfo CentOS-8-clone
virsh vcpuinfo CentOS-8-clone
```

简单实践

动态调整

增加cpu后检查

```
virsh setvcpus CentOS-8-x86_64 4
virsh vcpuinfo CentOS-8-x86_64
```

删减cpu后检查

```
virsh setvcpus CentOS-8-x86_64 2
virsh vcpuinfo CentOS-8-x86_64
grep vcpu /etc/libvirt/qemu/CentOS-8-x86_64.xml
```

调整cpu的时候, 同步修改配置

```
virsh setvcpus CentOS-8-x86_64 3
virsh vcpuinfo CentOS-8-x86_64
grep vcpu /etc/libvirt/qemu/CentOS-8-x86_64.xml
```

重置cpu

普通重置

```
virsh shutdown CentOS-8-x86_64
```

```
virsh start CentOS-8-x86_64
```

同步重置

```
virsh setvcpus CentOS-8-x86_64 1 --config
```

```
virsh vcpuinfo CentOS-8-x86_64
```

```
grep vcpu /etc/libvirt/qemu/CentOS-8-x86_64.xml
```

cpu绑定

虚拟机内部CPU号绑定

```
virsh vcpupin CentOS-8-x86_64 0 1 # 将kvm的0号cpu绑定在宿主机的1号cpu上
```

```
virsh vcpupin CentOS-8-x86_64 1 0 # 将kvm的1号cpu绑定在宿主机的0号cpu上
```

查看效果

```
virsh vcpuinfo CentOS-7-x86_64
```

小结

内存实践

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

需要根据实际情况，对kvm进行内存资源调整

注意：

不推荐频繁切换，否则导致虚拟机启动异常

基本语法

方法1

```
virsh qemu-monitor-command <虚拟机名称> --hmp --cmd 命令
```

命令解析：

查看所有信息 info

查看内存 info balloon

调整内存 balloon 定制容量

特点：

只管调整，不管系统是否重启成功

方法2

```
virsh setmem <虚拟机名称> 定制容量
```

注意：

该方法上课时候不推荐，因为不是及时生效

因为它涉及到虚拟机资源的重新加载(重启主机)，如果内存过小，导致系统无法启动。

再一个，频繁执行次数过多导致虚拟机卡死

超限使用

原则上，宿主机的资源是不允许超限使用的，比如，我要在 100G的电脑上，存储 200G的资源。这是不可能的。

根据我们之前对虚拟磁盘的原理解析，我们知道，分配的资源和使用资源是有区别的，实际使用的远远小于分配的资源。所以我们在分配内存的时候，可以对内存进行超配的使用

基本信息查看

查看配置文件

```
# virsh dumpxml CentOS-8-x86_64 | grep -i memo
<memory unit='KiB'>1048576</memory>
<currentMemory unit='KiB'>1048576</currentMemory>
```

查看虚拟机内存信息

```
# virsh dominfo CentOS-8-x86_64 | grep memory
Max memory:      1048576 KiB
Used memory:      1048576 KiB
```

简单实践

调整内存

查看内存信息

```
virsh qemu-monitor-command CentOS-8-x86_64 --hmp --cmd info balloon
```

调整内存信息

```
virsh qemu-monitor-command CentOS-8-x86_64 --hmp --cmd balloon 512
```

确认内存信息

```
virsh qemu-monitor-command CentOS-8-x86_64 --hmp --cmd info balloon
virsh dumpxml CentOS-8-x86_64 | grep -i memo
virsh dominfo CentOS-8-x86_64 | grep memory
```

结果显示：

主机的内存信息和配置信息都同步更改了

超限实践

调整内存的限制

```
# virsh edit CentOS-8-x86_64
```

...

```
4 <memory unit='KiB'>3170304</memory>
```

```
5 <currentMemory unit='KiB'>1048576</currentMemory>
```

说明：

调整了内存的最大值为3G，当前的内存大小为1G

动态调大内存

```
virsh qemu-monitor-command CentOS-8-x86_64 --hmp --cmd balloon 2048
```

查看当前内存大小


```
virsh qemu-monitor-command CentOS-8-x86_64 --hmp --cmd info balloon
动态调大内存
virsh qemu-monitor-command CentOS-8-x86_64 --hmp --cmd balloon 3000
查看当前内存大小
virsh qemu-monitor-command CentOS-8-x86_64 --hmp --cmd info balloon
```

```
[root@localhost ~]# free -m
              total        used         free       shared    buff/cache   available
Mem:           2750          139          2417           8         193        2459
Swap:          1023           0          1023
```

小结

磁盘实践

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

需求

虚拟机在操作过程中，会遇到各种数据容量操作，我们常见的操作就是
初始化时候，磁盘容量不合适，我们可以进行调整
数据容量不足，我们就增加一块数据磁盘
数据容量过大，我们就删减一块数据磁盘

常见语法

磁盘基本信息

```
virsh domblklist --details 虚拟磁盘名称
```

磁盘容量调整

```
qemu-img resize 虚拟磁盘名称 +5G
```

挂载磁盘

```
virsh attach-disk <虚拟机名称> --source <新磁盘名> --target <磁盘类型名> --persistent
```

注意：

磁盘类型名一般是 `sda~` 或者 `vda~`

这里的 `vda` 不一定是磁盘内部的信息，而是kvm管理的名称

卸载磁盘

```
virsh detach-disk <虚拟机名称> <磁盘类型名>
```

简单实践

磁盘容量调整

关闭虚拟机

```
virsh shutdown CentOS-8-x86_64
```

查看当前磁盘状态

```
qemu-img info /opt/CentOS-8-x86_64.raw
```

调整磁盘大小

```
qemu-img resize /opt/CentOS-8-x86_64.raw +1G
```

查看磁盘状态

```
qemu-img info /opt/CentOS-8-x86_64.raw
```

再次调整磁盘大小

```
qemu-img resize /opt/CentOS-8-x86_64.raw +2G
```

查看磁盘状态

```
qemu-img info /opt/CentOS-8-x86_64.raw
```

重启虚拟机

```
virsh start CentOS-8-x86_64
```

进入虚拟机查看磁盘大小

```
# fdisk -l
```

根据上面实验得知:

动态调整磁盘容量大小调整成功

磁盘基本操作

创建新磁盘

```
qemu-img create -f raw /opt/storage.qcow2 2G
```

增加新磁盘到虚拟机

```
virsh attach-disk CentOS-8-x86_64 --source /opt/storage.qcow2 --target vdb --persistent
```

```
virsh domblklist --details CentOS-8-x86_64
```

登录到虚拟机中检查效果

```
fdisk -l
```

删除虚拟机的某磁盘

```
virsh detach-disk CentOS-8-x86_64 --target vdb --persistent
```

登录到虚拟机中检查效果

```
fdisk -l
```

新增磁盘实践

关闭虚拟机

```
virsh shutdown CentOS-8-x86_64
```

新增一块磁盘

```
qemu-img create -f raw /opt/image.raw 2G
```

查看效果

```
ll /opt/
```

编辑配置文件

```
[root@linux-node1 ~]# virsh edit CentOS-8-x86_64
```

```
...
```

```
38     <disk type='file' device='disk'>
39         <driver name='qemu' type='raw'/>
40         <source file='/opt/image.raw'/>
41         <target dev='vda' bus='virtio'/>
42     </disk>
```

...

注意:

添加 38-42行的内容

和之前的disk配置缺少了<address type='pci'... 的内容。 -- 该部分内容会自动增加

启动虚拟机

```
virsh start CentOS-8-x86_64
```

连接进入虚拟机确认新增磁盘已识别

```
# fdisk -l
```

格式化磁盘

```
# mkfs.ext4 /dev/vdb
```

新建一个目录来挂载新硬盘

```
mkdir /data
```

```
mount /dev/vdb /data/
```

```
mount | grep vdb
```

```
umount /data
```

开机自动挂载

```
blkid /dev/vdb
```

```
echo 'UUID=fd53695a-4a6f-4735-8439-f02707ec7a6e /data ext4 defaults 1 2'>>
/etc/fstab
```

```
mount -a
```

```
mount | grep vdb
```

```
df -h | egrep 'Files|vd'
```

可以看到:

新增磁盘的方法调整虚拟机的磁盘容量，在虚拟机中想要看到效果，必须重启虚拟机

小结

拓展思考

虚拟化原理

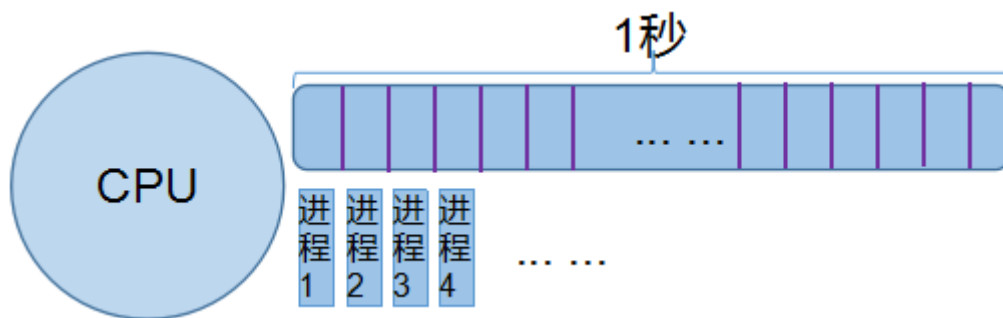
cpu虚拟化

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

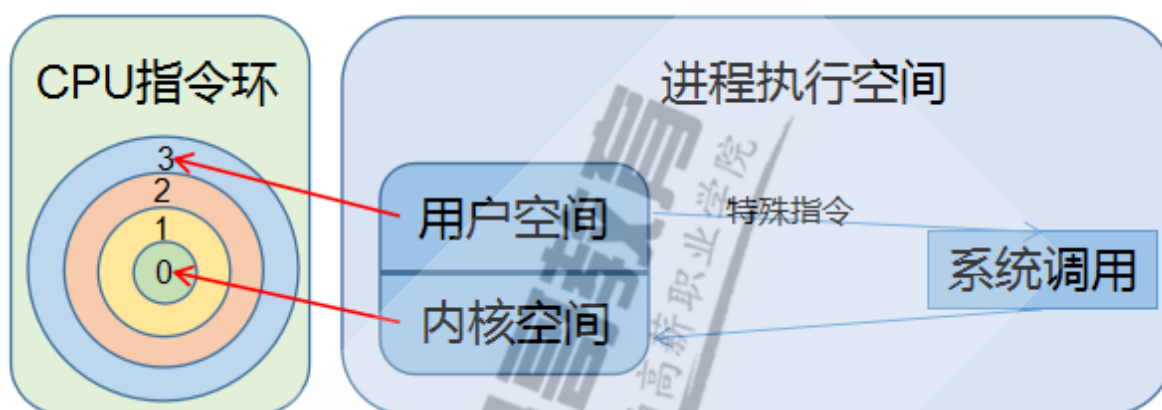
基础知识

时间分片



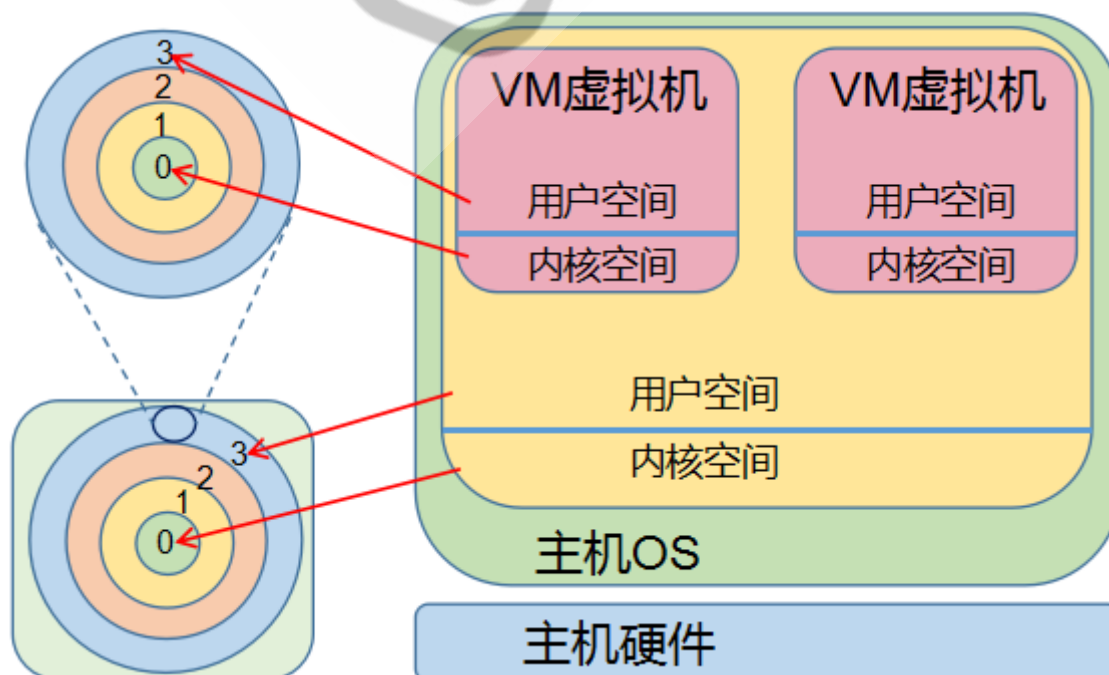
操作系统安装后，CPU默认情况下就是被虚拟化了的，只不过是基于"时间片"的方式，所以虚拟化技术也需要基于"分时技术"来实现，只不过比进程的虚拟化实现麻烦很多。原因就在于工作中的任务都是由进程来实现的。

系统调用



为了协调多任务，操作系统被分成了两段：
内核空间：接近硬件，主要执行特权指令
用户空间：执行普通的指令
用户空间的执行想要执行某些特权指令，就需要进行"系统调用"

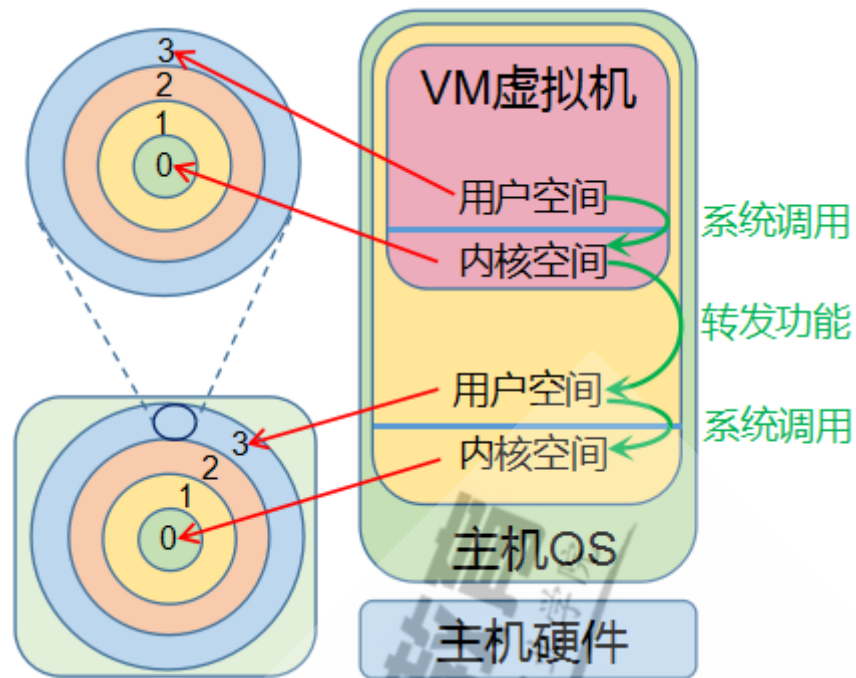
虚拟机内部的cpu指令



简单实践

实现虚拟机操作宿主机的资源的方式很多，主要是通过以下几种方式来实现：模拟、完全虚拟化、硬件辅助虚拟化、半虚拟化

模拟



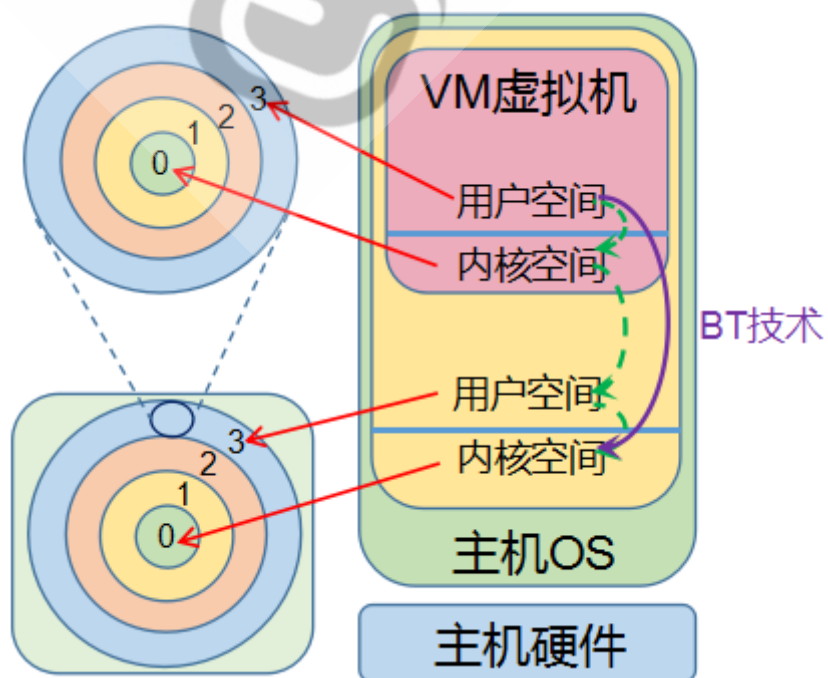
关机场景：

使用宿主机的进程来模拟虚拟机的CPU，那么虚拟机的操作指令都转发给宿主机的进程来执行，最终关闭宿主机的进程来模拟虚拟机的电源功能。

缺点：

虚拟机大量进程间的转换导致宿主机的cpu切片频繁发生，导致性能很不好。

完全虚拟化



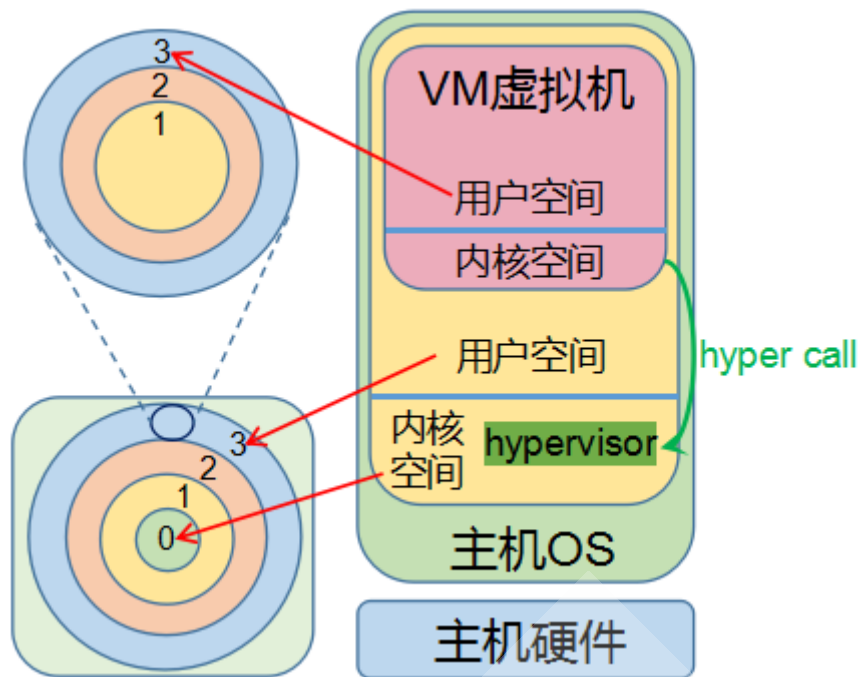
因为虚拟机指令转发给宿主主机进程太消耗性能了，所以就出来了一种新的技术：**BT(二进制转换)**将虚拟机对特权指令的调用，在调用的那一刻直接翻译成对宿主主机的特权指令的调用，

比分时技术稍好一些，但性能依然很差，达到物理性能的80%。
某些用户空间指令没有必要翻译，可直接执行在主机的ring3上，

The diagram illustrates the relationship between a VM virtual machine, Host OS, and Host Hardware. It shows three nested layers: VM virtual machine (top), Host OS (middle), and Host Hardware (bottom). The VM virtual machine contains User Space and Kernel Space. The Host OS contains User Space, Kernel Space, and Host Hardware. The diagram shows how the VM's Kernel Space maps to the Host OS's Kernel Space.

据说：达到物理主机性能的85%。

半虚拟化



宿主机中接收虚拟机指令的组件叫hypervisor，它直接管理宿主机的CPU和mem硬件，不包括io硬件。

半虚拟化：修改操作系统的内核，让虚拟机的内核知道自己是在虚拟环境中，不能执行特权指令。虚拟主机发现要执行一个特权指令，那么虚拟主机内核向宿主机发起hyper call，交由宿主机的hypervisor组件去执行相应的硬件。

据说：达到物理主机性能的90~95%

小结

内存虚拟化

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

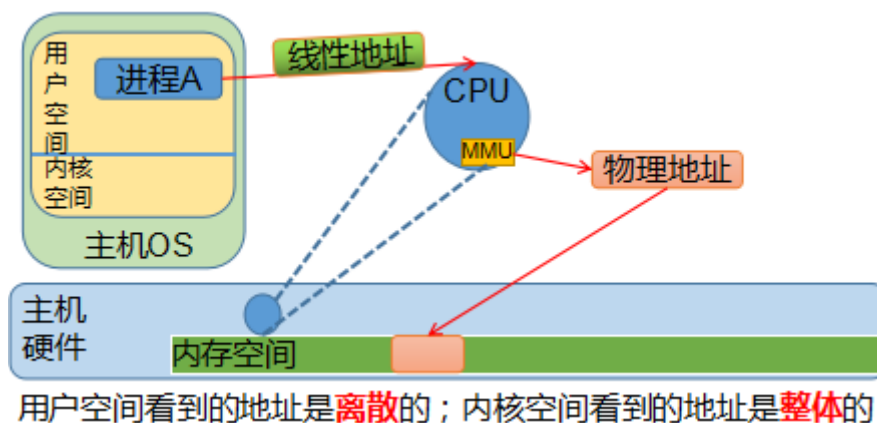
内存虚拟化

内存本身就是虚拟化的。

因为进程看到的是线性地址空间，内核看到的是物理地址空间。

没有虚拟化的时候，主机内核以为自己操作的是整个内存空间，而在虚拟化场景中，每个虚拟机内核都认为为自己操作的是整个内存空间，这是不现实的。

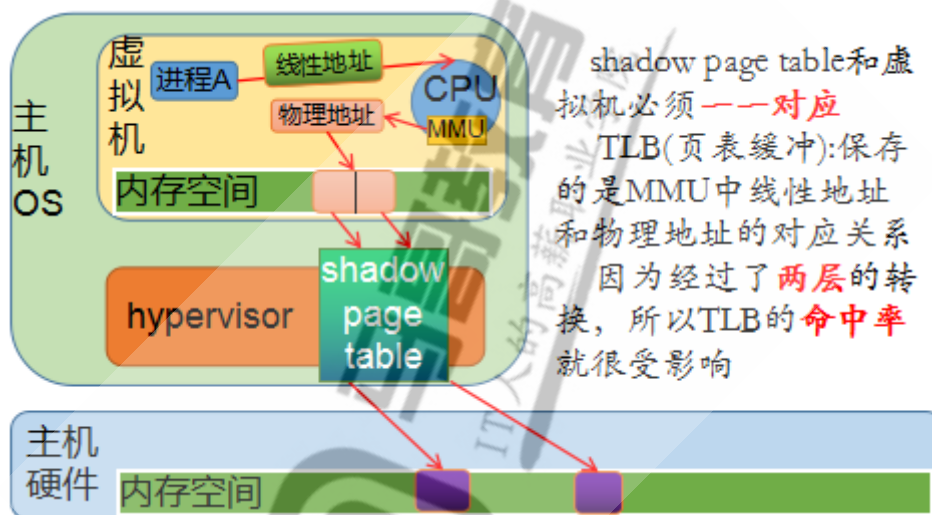
基本原理



简单实践

实现虚拟机操作宿主机的资源的方式很多，主要是通过以下几种方式来实现：模拟、mmu虚拟化、TLB虚拟化

模拟



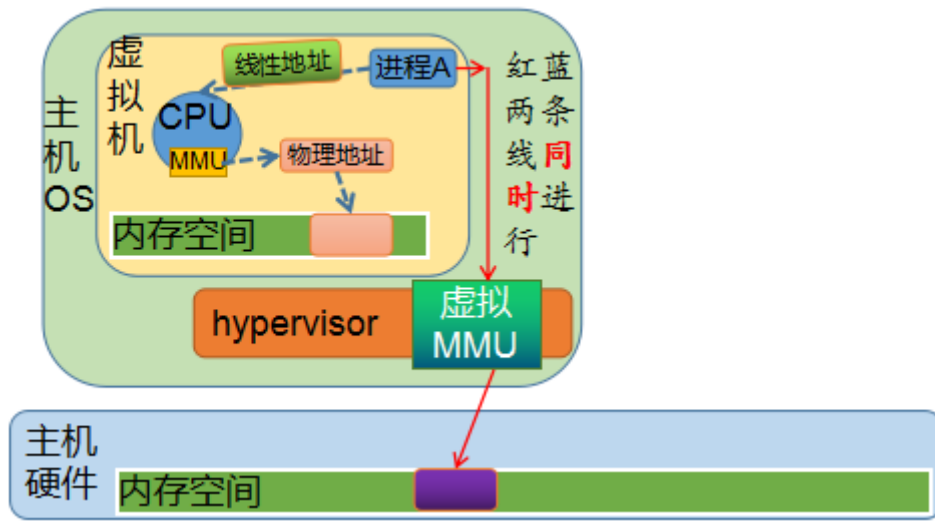
虚拟机进程经过CPU的MMU的转换到虚拟机的内存中找到了相应的地址，但是这个地址不是真正的宿主机的内存地址，所以再次经过shadow page table的技术，将虚拟的内存地址转换成真正的物理地址，找到对应的存储元素。

缺点：

shadow page table 必须和虚拟机一一对应使用

TLB缓存相应的虚拟机A地址转换信息，但是当切换虚拟机B使用的时候，TLB内容就失效了。

mmu虚拟化



借助于cpu的硬件虚拟化作用，在虚拟机进行线性地址转换的同时，通过虚拟MMU将其转换成宿主机内存的物理地址，提升了内存的性能。

缺点：

TLB的命中率还是比较低

TLB虚拟化

虚拟主机 线性地址 物理地址

在方法二的基础上，引入一个TLB虚拟化功能，在缓存信息的时候，增加一个标签的功能，类似数据库表字段。

小结

IO虚拟化

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

常见io设备

对服务器来说的必备IO设备：

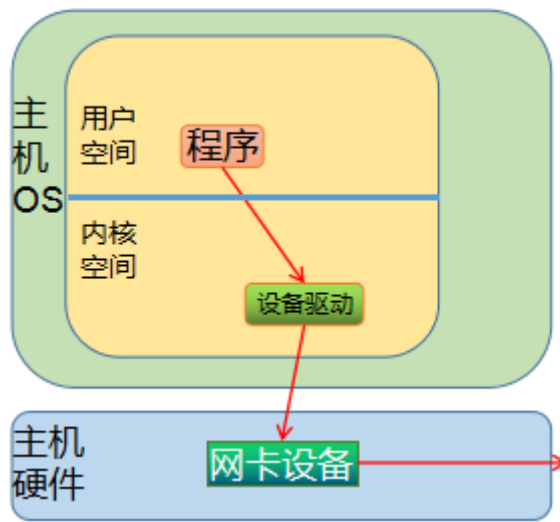
外存设备： 硬盘、光盘、U盘

显示设备： VGA

网络设备： 网卡

键盘鼠标： ps/2, usb

io原理(以网卡为例)

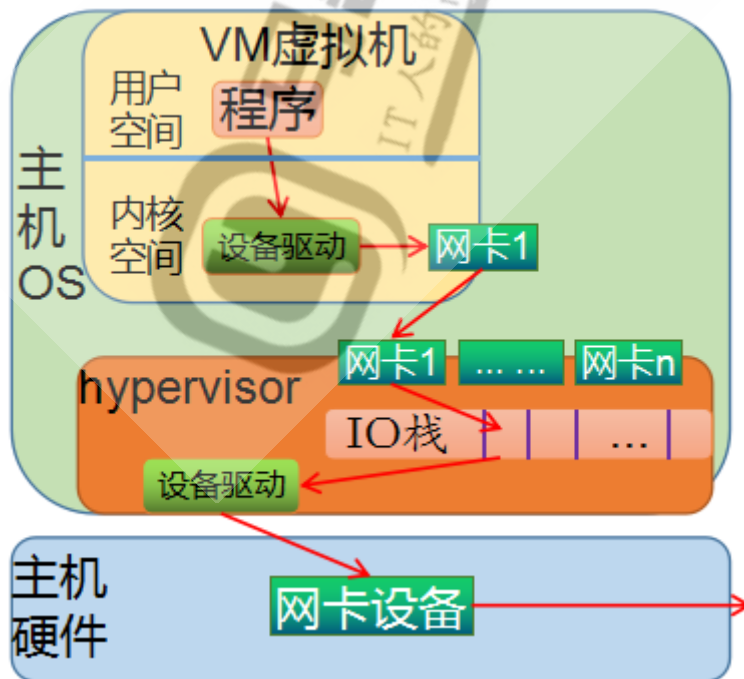


主机用户空间中的一个程序，发出一个信息报文，通过系统调用，由内核调用该设备的驱动程序，由主机硬件中的网卡设备将信息报文输出

简单实践

实现虚拟机操作宿主机的资源的方式很多，主要是通过以下几种方式来实现：模拟、半虚拟化、透传、硬件虚拟化

模拟



完全使用软件来模拟真实的常见的硬件使用场景。

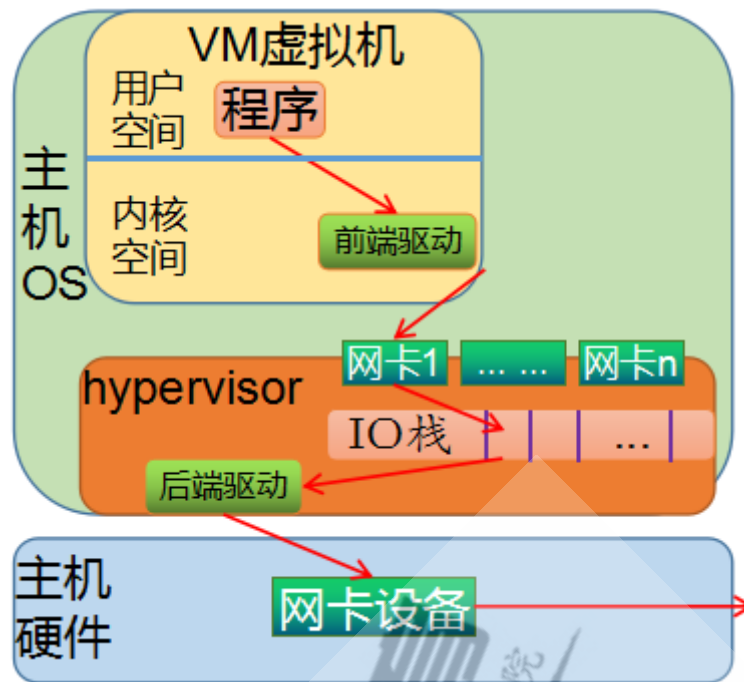
虚拟主机用户空间中的一个程序，发出一个信息报文，通过系统调用，由内核调用该设备的驱动程序，由主机硬件中的网卡设备将信息报文输出，该信息交给宿主机的hypervisor上对应的网卡设备，然后把该网卡接收到的信息暂存到IO栈中，然后调用宿主机的网卡设备驱动，经宿主机网卡将该信息报文发出去。

缺点：

虚拟机上的设备驱动以及虚拟网卡发送信息所做的工作和hypervisor是重复的。

物理硬件的性能发挥60%左右

半虚拟化



借助硬件虚拟化功能，将设备驱动分成两部分，前端驱动和后端驱动，前端驱动是虚拟机专用的，后端驱动是统一使用的。

虚拟主机用户空间中的一个程序，发出一个信息报文，通过系统调用，内核不做任何处理，直接由该设备的虚拟驱动程序--前端驱动，该信息交给宿主机的hypervisor上对应的网卡设备，然后把该网卡接收到的信息暂存到IO栈中，然后调用网卡驱动--后端驱动，经宿主机网卡将该信息报文发出去。这样就省略了虚拟机中重复调用驱动发送信息的动作。

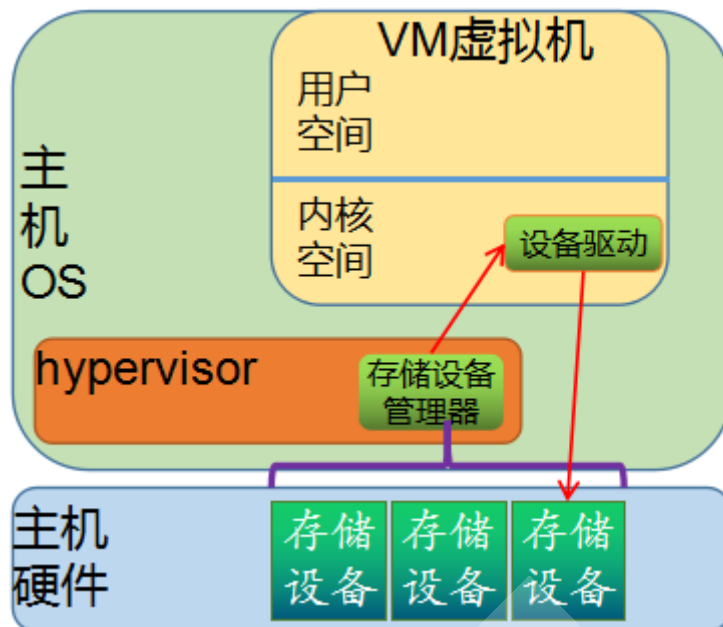
可以使用物理硬件95%的性能

缺点：

前后端驱动的方法只适用于硬盘和网卡设备，

显卡设备的虚拟，一般使用frame buffer机制来实现。目前显卡的虚拟，可以在Centos7上使用半虚拟化方式来实现。

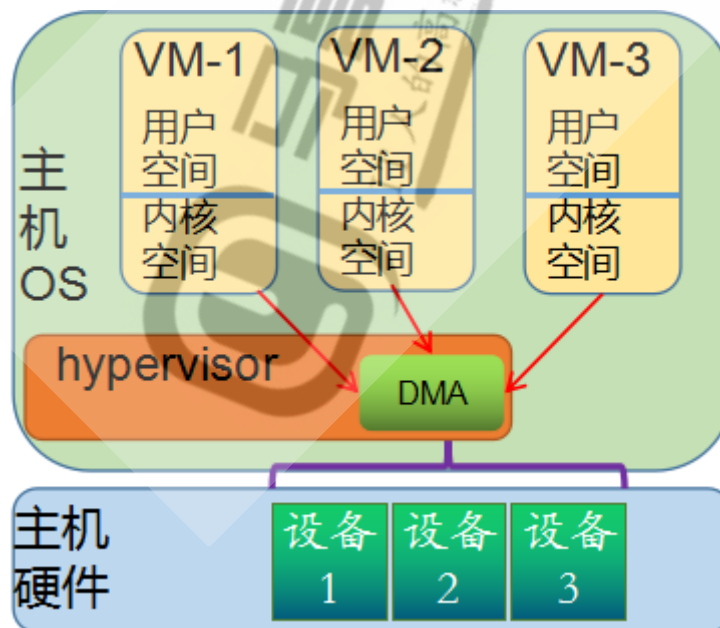
透传



宿主主机上有多块盘，每个虚拟机直接使用一块专用盘，像这种场景方法就叫IO透传。前提是我们的硬件必须支持响应的透传技术。

宿主主机上的hypervisor管理所有的存储设备，然后分配给VM虚拟机一个专用的存储设备，虚拟机使用的时候，直接通过内核空间中的设备驱动，操作该分配的存储设备。

硬件虚拟化



传统的x86架构的设备有一个集中式的管理设备叫DMA(直接内存访问)，它是一种加速io访问性能的方式，在它内部有一个IOMMU，它可以进行IO寄存器和IO设备端口的自动映射工作。

如果在宿主机用DMA，所有的虚拟机都连接一个DMA，所以虚拟机io设备的调用必须在IOMMU级别上实现隔离，VT-d就是基于北桥的硬件辅助的虚拟化技术，提高了IO设备的可靠性、灵活性和性能。

小结

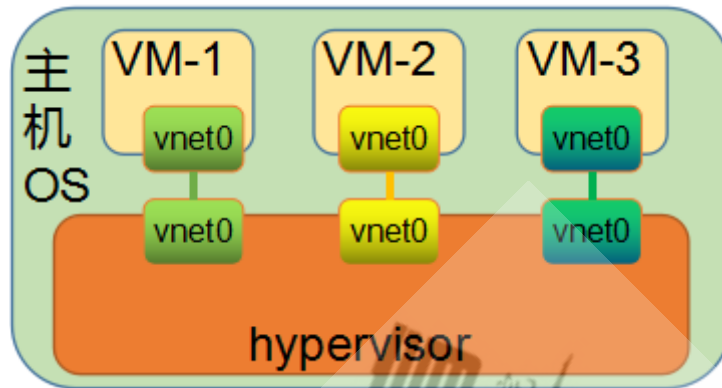
网络虚拟化

学习目标：

这一节，我们从 虚拟原理、网络模式、小结 三个方面来学习。

虚拟原理

虚拟网络设备创建的原理：



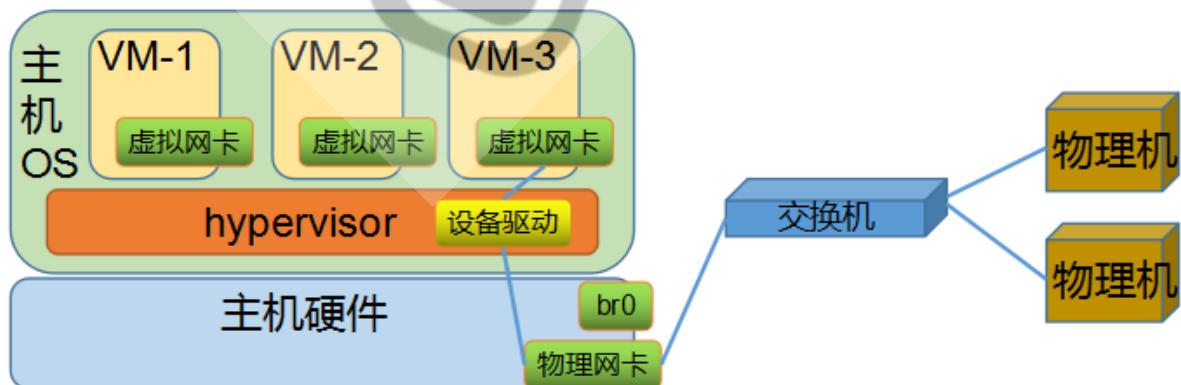
虚拟网络设备是由软件实现的，它们是成对出现的：即虚拟机和hypervisor会生成一对虚拟网卡设备，vm通过vnet0和绑定在hypervisor上的vnet0进行通信，反之一样。

该虚拟网络设备主要有两种方式：TUN(基本不用)与TAP；TAP模拟以太网设备，操作第二层数据包。TUN模拟网络层设备，操作第三层数据包。

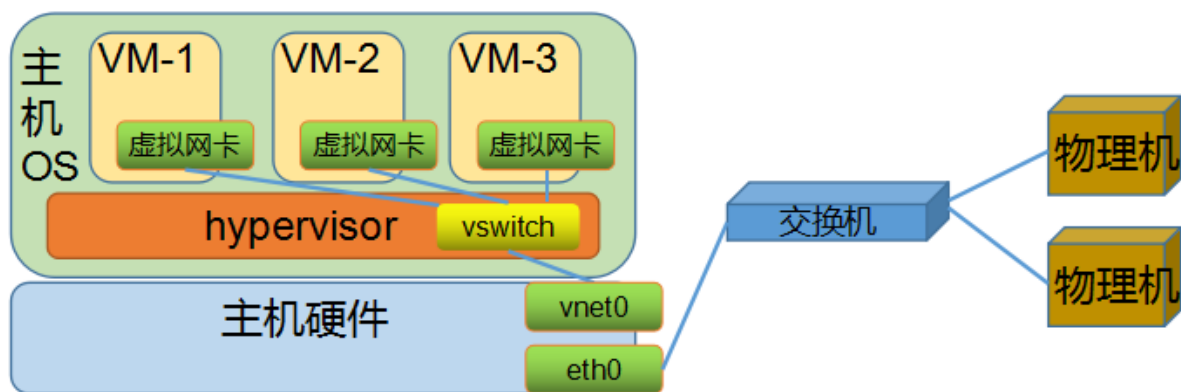
网络模式

网络的虚拟化，一般是由四种模型来实现的：桥接模型、路由模型(仅主机模型)、net模型、隔离模型

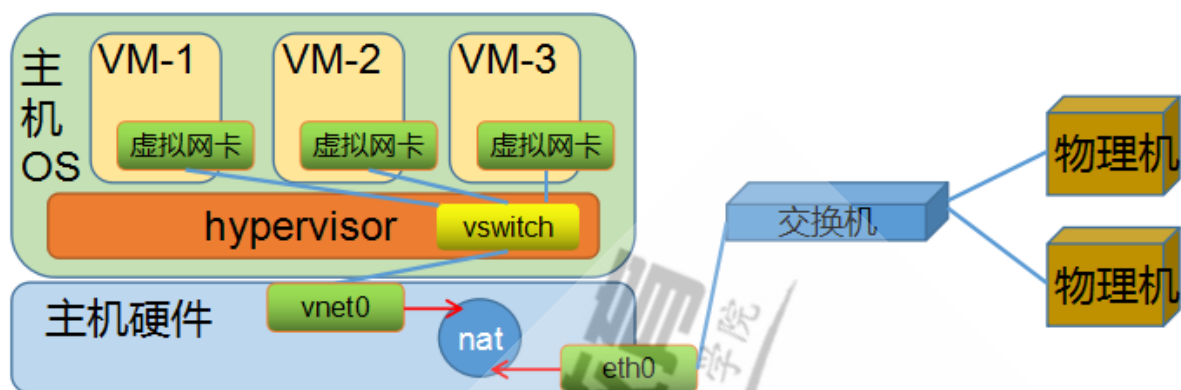
桥接模型



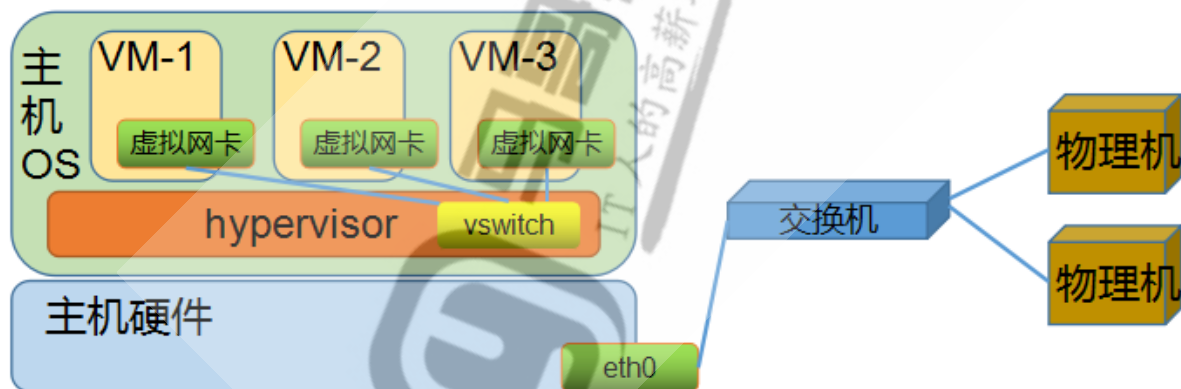
路由模型(仅主机模型)



net模型



隔离模型



小结

其他

批量管理

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

kvm虚拟机如何批量管理

专用工具:

wok+kimchi 搭配实现一个kvm的web界面管理平台。

wok基于cherry.py的web框架, 可以通过一些插件来进行扩展, 例如: 虚拟化管理、主机管理、系统管理。

Kimchi是一个基于HTML5的KVM管理工具, 是wok的一个插件, 通过Kimchi可以更方便的管理KVM

github地址: <https://github.com/kimchi-project>

ovirt 一个开源的虚拟化管理平台, 是redhat 虚拟化管理平台RHEV的开源版本。

官方地址: <https://www.ovirt.org/>

Proxmox 一款套开源的虚拟化管理软件, 用户可通过网页的方式来管理服务器上使用 kvm等虚拟机。

官方地址: <https://www.proxmox.com/en/>

平台工具:

OpenStack、CloudStack、EucaIptus、OpenNebula、等等

基本安装

依赖环境

```
sudo apt install -y python3-configobj python3-xml python3-magic python3-paramiko python3-ldap spice-html5 novnc qemu-kvm python3-libvirt python3-parted python3-guestfs python3-pil python3-cherry.py3 libvirt0 libvirt-daemon-system libvirt-clients nfs-common sosreport open-iscsi libguestfs-tools libnl-route-3-dev python3-cheetah gcc make autoconf automake git python3-pip python3-requests python3-mock gettext pkgconf xsltproc python3-dev pep8 pyflakes python3-yaml
```

确认效果

```
kvm-ok
```

更新python库

```
sudo apt-get -y remove python3-cherry.py3
sudo pip3 install cherry.py ethtool python-pam psutil jsonschema
```

安装wok

```
# 编译安装软件
cd /tmp
wget https://github.com/kimchi-project/wok/archive/3.0.0.tar.gz
tar -zxvf 3.0.0.tar.gz
cd wok-3.0.0

sudo ./autogen.sh --system
sudo make
sudo make install
sudo make deb
sudo apt install ./wok-3.0.0-0.ubuntu.noarch.deb

# 检查效果
systemctl status wokd.service
netstat -tnulp | grep 8010
```

安装 kimchi

```
# 编译安装软件
cd /tmp
git clone https://github.com/kimchi-project/kimchi kimchi
cd kimchi

sudo ./autogen.sh --system
sudo make
sudo make install
sudo make deb
sudo apt install ./kimchi-3.0.0-13.git36ed74be.noarch.deb

# 检查效果
systemctl restart wokd.service
netstat -tnulp | grep 8010
```

配置apache

```
# 安装软件
apt install apache2

# 加载模块
sudo touch /etc/wok/plugins.d/__pycache__.conf
sudo a2enmod ssl
sudo a2enmod rewrite
a2enmod proxy
a2enmod proxy_http
a2enmod proxy_connect
systemctl restart apache2
systemctl reload apache2

# 设定专用配置
cat <<- __EOF__ | sudo tee -a /etc/apache2/sites-available/kimchi.conf
Listen 8001
Listen 8000

<VirtualHost *:8001>
    ErrorLog /var/log/apache2/error_log
    TransferLog /var/log/apache2/access_log

    Timeout 600
    LimitRequestBody 4294967296

    SSLEngine on
    SSLCertificateFile /etc/wok/wok-cert.pem
    SSLCertificateKeyFile /etc/wok/wok-key.pem
    SSLOpenSSLConfCmd DHParameters /etc/wok/dhparams.pem
    SSLSessionCacheTimeout 600

    RewriteEngine On
    RewriteCond %{HTTP:Upgrade} =websocket [NC]
    RewriteRule /(.*) ws://localhost:64667/$1 [P,L]

    ProxyPass /websocketify http://127.0.0.1:64667/websocketify
    ProxyPassReverse /websocketify http://127.0.0.1:64667/websocketify
```



```

ProxyPass / http://127.0.0.1:8010/
ProxyPassReverse / http://127.0.0.1:8010/
</VirtualHost>

<VirtualHost *:8000>
    ErrorLog /var/log/apache2/error_log
    TransferLog /var/log/apache2/access_log

    RewriteEngine On
    RewriteRule ^/(.*)$ https://%{SERVER_NAME}:8001/$1 [R]
</VirtualHost>
__EOF__

# kimchi配置加载到apache
cd /etc/apache2/sites-enabled/
sudo ln -s ../sites-available/kimchi.conf kimchi.conf

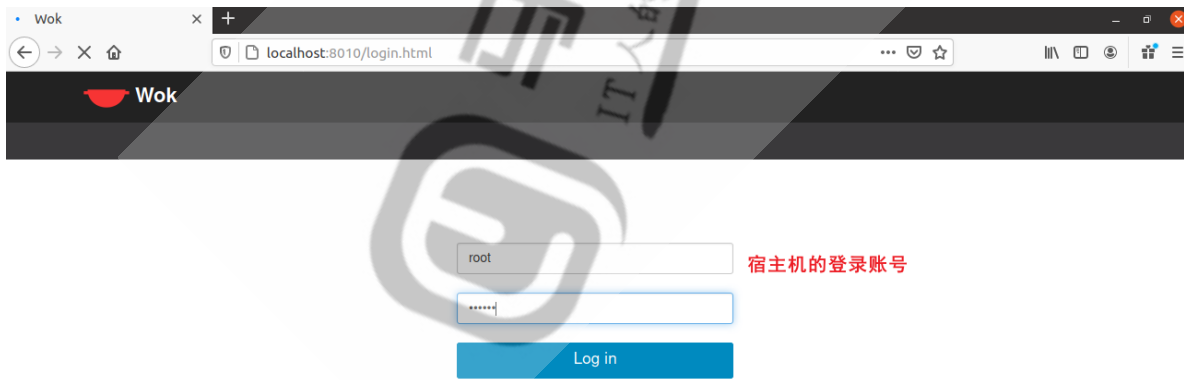
# apache重启
sudo systemctl restart apache2.service
sudo systemctl status apache2.service

# 查看日志效果
sudo tail -f /var/log/wok/wok-*.log

```

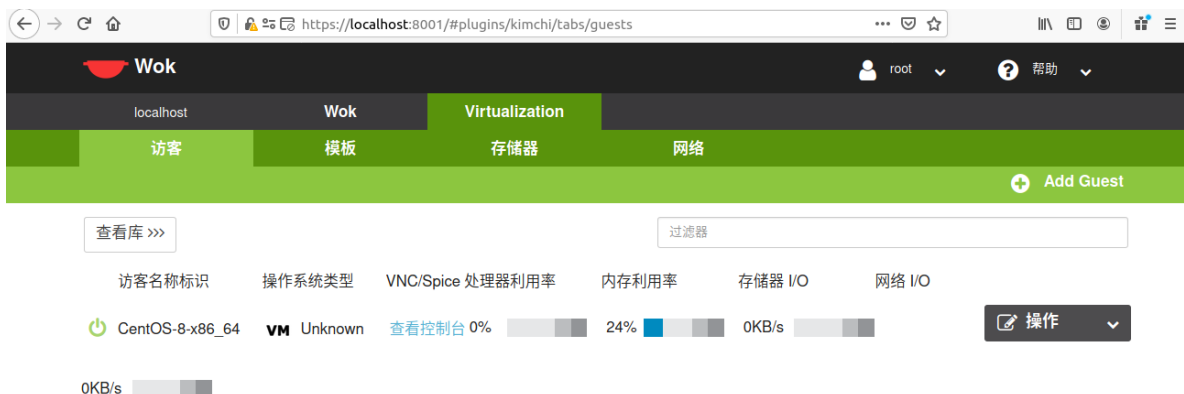
简单实践

浏览器访问 127.0.0.1:8001，自动代理到后台的 wok业务上面



注意：这里用的登录信息是 当前系统的登录账号和密码登信息。

登录后查看效果



小结

