虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化，是指通过虚拟化技术将一台计算机虚拟为多台逻辑计算机。在一台计算机上同时运行多个逻辑计算机，每个逻辑计算机可运行不同的操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化使用软件的方法重新定义划分 IT 资源，可以实现 IT 资源的动态分配、灵活调度、跨域共享，提高 IT 资源利用率，使 IT 资源能够真正成为社会基础设施，服务于各行各业中灵活多变的应用需求

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

灵活多变的应用需求有哪些？

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化种类：

1) 完全虚拟化 --- 最流行的虚拟化方法使用名为 hypervisor 的一种软件，在虚拟服务器和底层硬件之间建立一个抽象层。 VMware 和微软的 VirtualPC 是代表该方法的两个商用产品，而基于核心的虚拟机 (KVM) 是面向 Linux 系统的开源产品 hypervisor 可以捕获 CPU 指令，为指令访问硬件控制器和外设充当中介。因而，完全虚拟化技术几乎能让任何一款操作系统不用改动就能安装到虚拟服务器上，而它们不知道自己运行在虚拟化环境下。主要缺点是， hypervisor 给处理器带来开销

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化种类：
2) 准虚拟化 --- 完全虚拟化是处理器密集型技术，因为它要求 hypervisor 管理各个虚拟服务器，并让它们彼此独立。减轻这种负担的一种方法就是，改动客户端操作系统，让它以为自己运行在虚拟环境下，能够与 hypervisor 协同工作。这种方法就叫准虚拟化 (para-virtualization)
Xen 是开源准虚拟化技术的一个例子。操作系统作为虚拟服务器在 Xen hypervisor 上运行之前，它必须在核心层面进行某些改变。因此， Xen 适用于 BSD 、 Linux 、 Solaris 及其他开源操作系统，但不适合对像 Windows 这些专有的操作系统进行虚拟化处理，因为它们无法改动。
准虚拟化技术的优点是性能高。经过准虚拟化处理的服务器可与 hypervisor 协同工作，其响应能力几乎不亚于未经过虚拟化处理的服务器。准虚拟化与完全虚拟化相比优点明显，以至于微软和 VMware 都在开发这项技术，以完善各自的产品

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化种类：

1) 系统虚拟 --- 就操作系统层的虚拟化而言，没有独立的 hypervisor 层。相反，主机操作系统本身就负责在多个虚拟服务器之间分配硬件资源，并且让这些服务器彼此独立。一个明显的区别是，如果使用操作系统层虚拟化，所有虚拟服务器必须运行同一操作系统 (不过每个实例有各自的应用程序和用户账户)。

虽然操作系统层虚拟化的灵活性比较差，但本机速度性能比较高。此外，由于架构在所有虚拟服务器上使用单一、标准的操作系统，管理起来比异构环境要容易。

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化种类：

1) 桌面虚拟化 --- 服务器虚拟化主要针对服务器而言，而虚拟化最接近用户的还是要算的上桌面虚拟化了，桌面虚拟化主要功能是将分散的桌面环境集中保存并管理起来，包括桌面环境的集中下发，集中更新，集中管理。桌面虚拟化使得桌面管理变得简单，不用每台终端单独进行维护，每台终端进行更新。终端数据可以集中存储在中心机房里，安全性相对传统桌面应用要高很多。桌面虚拟化可以使得一个人拥有多个桌面环境，也可以把一个桌面环境供多人使用

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化种类：

1) 硬件助力软件 --- 不像大型机， PC 的硬件在设计时并没有考虑到虚拟化，而就在不久前，它还是完全由软件来承担这项重任。随着 AMD 和英特尔推出了最新一代的 x86 处理器，头一回在 CPU 层面添加了支持虚拟化的功能

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

你当前的 CPU 是否支持 VT 技术？
当不确定你当前 CPU 是否支持 VT 技术时
1. 可以在 windows 下使用 cpu-z 软件来进行测试
2. 可以在 Linux 下查看 CPU 的相信信息来确定

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

CPU 虚拟化给我们带来了哪些好处？
CPU 的虚拟化技术可以将单 CPU 模拟多 CPU 并行，允许一个平台同时运行多个操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化技术与多任务超线程的技术的区别？
虚拟化技术与多任务以及超线程技术是完全不同的。多任务是指在一个操作系统中多个程序同时并行运行，而在虚拟化技术中，则可以同时运行多个操作系统，而且每一个操作系统中都有多个程序运行，每一个操作系统都运行在一个虚拟的 CPU 或者是虚拟主机上；而超线程技术只是单 CPU 模拟双 CPU 来平衡程序运行性能，这两个模拟出来的 CPU 是不能分离的，只能协同工作

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化的发展史：

1. 上世纪 60 年代，虚拟化技术最初应用在大型机上，在未来的 20 年，虚拟化技术发展越发成熟，但随着小型机和 x86 的流行，大型机在新兴的服务器市场已失去了影响力
2. 由于处理器架构不同，应用在大型机的虚拟化技术却不能为小型机和 x86 所用
3. 直到 2001 年， VMware 发布了第一个针对 x86 服务器的虚拟化产品
4. 之后的几年间，英国剑桥大学的一位讲师发布了同样针对 x86 虚拟化的开源虚拟化项目 Xen ；惠普发布了针对 HP-UX 的 Integrity 虚拟机； Sun 跟 Solaris 10 一同发布了同时支持 x86/x64 和 SPARC 架构的 Solaris Zone ；而微软也终于在 2008 年发布的 Windows Server 2008 R2 中加入了 Hyper-V 。期间， VMware 被 EMC 收购， XenSource 则被思杰收购。

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化的发展史：

5. 之后的几年间， VMware 逐渐在企业级市场中被广泛的接受， Xen 也逐渐在互联网领域展露头角。在成熟的服务器操作系统当中， Novell SUSE Linux Enterprise 10 是第一个采用 Xen 技术的。当时的 Xen 还很不成熟，乃至于红帽还为此取笑了 Novell 一番；不过几个月后，到了 RHEL 5.0 发布的时候，红帽决定也将 Xen 加入到自己的默认特性当中——那是 2006 年。一时之间，在 Linux 服务器领域， Xen 似乎成为了 VMware 之外的最佳虚拟化选择（事实上也没多少其他可选的）

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化的发展史：

6. 但是，作为一项 Linux 平台上的虚拟化技术， Xen 在很长一段时间内一直没有被接受到 Linux 内核的代码当中，这对于 Xen 的维护者而言，不仅意味着要多做很多工作，还意味着用户在废了半天劲装好 Xen 之后可能遇到意想不到的问题（注： 2011 年 6 月发布的 Linux 内核 3.0 中已经加入了对 Xen 的支持—— Xen 的工程师们表示这是清理了 7 年遗留代码、提交了 600 多个补丁的成果）。而红帽方面，也许是因为当时对这种脱离内核的维护方式很不爽，也许是因为采用 Xen 的 RHEL 在企业级虚拟化方面没有赢得太多的市场，也许是因为思杰跟微软走的太近了，种种原因，导致其萌生了放弃 Xen 的心思。事实上，当时整个 Xen 的市场表现的确一般， 2008 年 Hyper-V 推出的时候，甚至有评论猜测思杰自己都会抛弃 Xen 而投奔 Hyper-V

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

虚拟化的发展史：

7. 总之，红帽决定选择了一个新兴的、基于内核的虚拟化技术：KVM。而且在正式采用 KVM 一年后，就宣布在新的产品线中彻底放弃 Xen ，集中资源和精力进行 KVM 的工作

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

KVM --- Kernel-based Virtual Machine
是一个开源软件，实际是嵌入系统的一个虚拟化模块，自 Linux 2.6.20 之后逐步取代 Xen 被集成在 Linux 的各个主要发行版本中。它使用 Linux 自身的调度器进行管理，所以相对于 Xen ，其核心源码很少。 KVM 目前已成为学术界的主流 VMM（ Virtual Machine Monitor ）之一。

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

KVM 是基于 Linux 内核的虚拟机

2006 年 10 月由以色列的 Qumranet 组织开发的一种新的“虚拟机”方案，并将其贡献给开源世界

2007 年 2 月于 Linux Kernel-2.6.20 中第一次包含了 KVM

2008 年 9 月，红帽收购了一家名叫 Qumranet 的以色列小公司，由此入手了一个叫做 KVM 的虚拟化技术

虚拟化 ---KVM （ kernel-based VirtualMachine ）

2009 年 9 月，红帽发布其企业级 Linux 的 5.4 版本（ RHEL 5.4 ），在原先的 Xen 虚拟化机制之上，将 KVM 添加了进来

2010 年 11 月，红帽发布其企业级 Linux 的 6.0 版本（ RHEL 6.0 ），这个版本将默认安装的 Xen 虚拟化机制彻底去除，仅提供 KVM 虚拟化机制

2011 年初，红帽的老搭档 IBM 找上红帽，表示 KVM 这个东西值得加大力度去做。于是到了 5 月， IBM 和红帽，联合惠普和英特尔一起，成立了开放虚拟化联盟（ Open Virtualization Alliance ），一起声明要提升 KVM 的形象，加速 KVM 投入市场的速度，由此避免 VMware 一家独大的情况出现。联盟成立之时，红帽的发言人表示，“大家都希望除 VMware 之外还有一种开源选择。未来的云基础设施一定会基于开源

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

KVM 在标准的内核中增加了虚拟机技术，从而可以通过优化内核来使用虚拟技术。

在 KVM 模型中，每一个虚拟机都一个由 Linux 调度程序管理的标准进程

用户可以在用户空间启动客户端 (Guest OS)

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

在 Linux 中进程的运行模式有内核与用户 2 种

KVM 则增加了第三种模式 : 客户模式 (即自己的内核和用户模式)

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

当然， KVM 本身也有一些弱点，那就是相比裸金属虚拟化架构的 Xen、 VMware ESX 和 Hyper-V ， KVM 是运行在 Linux 内核之上的寄居式虚拟化架构，会消耗比较多的计算资源；不过针对这一点， Intel、 AMD 已经在处理器设计上有专门的 VT-x 和 AMD-V 扩展，这种特性在每次硬件更新的时候也会更新，往往每次更新后都对虚拟化性能和速度上有明显的提升，所以长远来看，也不是什么大问题。

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

？？如何查看当前系统的内核版本？？

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

KVM 的虚拟化需要硬件支持（需要处理器支持虚拟化：如 Intel 厂商的 Intel-VT （ vmx ）技术 &&AMD 厂商的 AMD-V （ svm ）技术。是基于硬件的完全虚拟化。而 Xen 早期则是基于软件模拟的半虚拟化（ Para-Virtualization ），新版本则是基于硬件支持的完全虚拟化。但 Xen 本身有自己的进程调度器，存储管理模块等，所以代码较为庞大。

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

？？查看 your 计算机的处理器的品牌？？

？？查看 your 处理器是否支持虚拟化？？

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

当你硬件本身支持虚拟化，但查询相应参数无果时，请检查 BIOS 设定，确认你的 BIOS 中开启了硬件支持虚拟化的功能！

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

？？ BIOS 中哪个选项控制虚拟化设定？？

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装前准备工作

1. 确定处理器有 VT

命令行： grep vmx /proc/cpuinfo (INTEL 芯片)

grep svm /proc/cpuinfo (AMD 芯片)

不知道芯片的生产厂商则输入：

egrep '(vmx|svm)' /proc/cpuinfo

如果 flags: 里有 vmx 或者 svm 就说明支持 VT ；如果没有任何的输出，说明你的 cpu 不支持，将无法成功安装 KVM 虚拟机。

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装前准备工作

2. 确保 BIOS 里开启 VT
Intel(R) Virtualization Tech [Enabled]

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装前准备工作

3. 确保内核版本较新，支持 KVM

查看内核版本

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

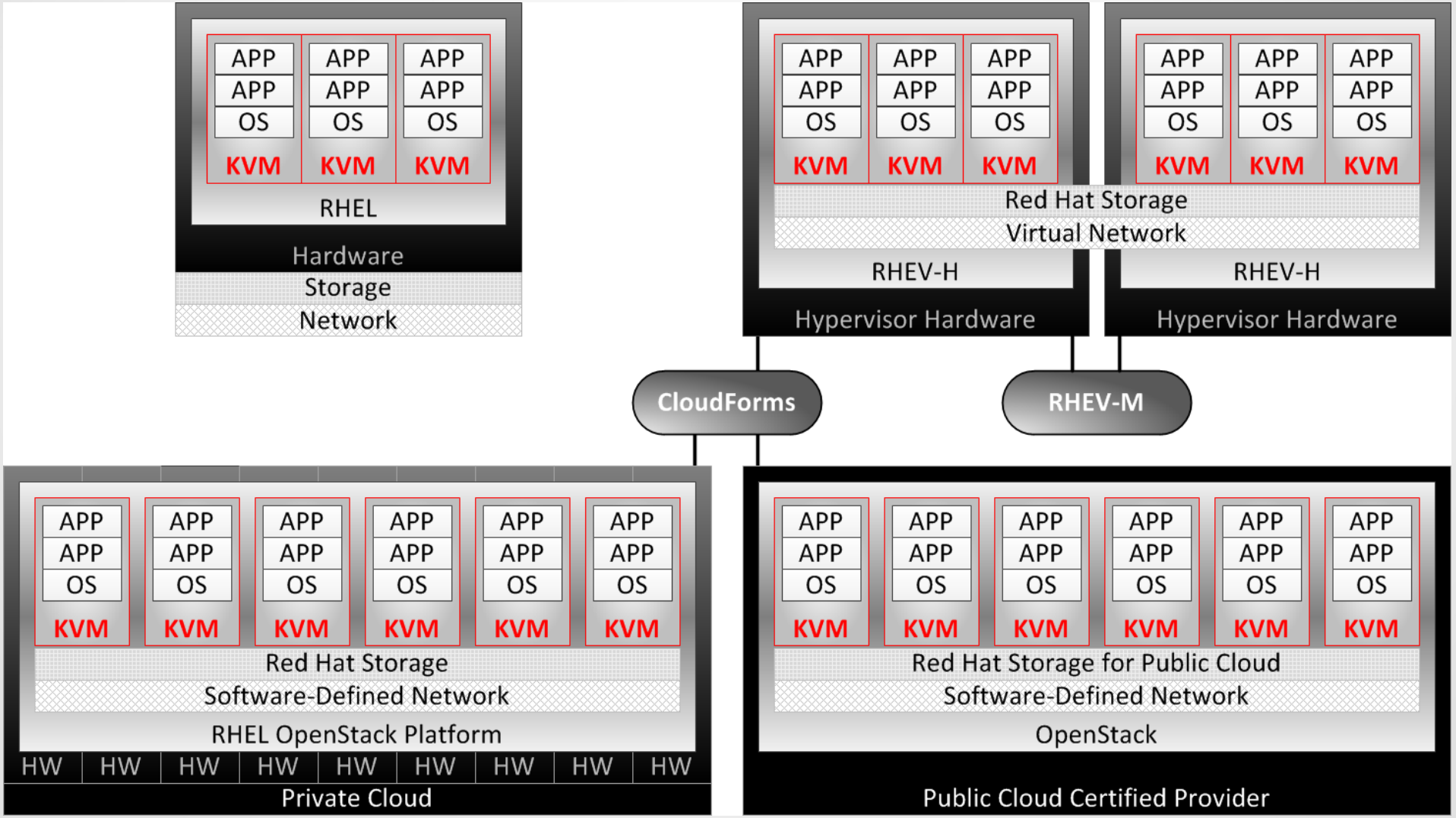
准确的描述， KVM 仅仅是 Linux 内核中一个模块

在管理和创建完整的 KVM 虚拟机则需要更多的辅助工具来配合及使用

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

RedHat 在经过长期的努力，使得 KVM 在其平台上蓬勃发展，并得到良好的支持。

虚拟化 ---KVM
（ kernel-based VirtualMachine ）



虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装 KVM 要求

1) 64bitCPU(支持虚拟化 VT-x or AMD-V)

2) 2G 以上空闲内存

3) 6GB 空闲存储空间

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装 KVM 的前期准备工作

1）确认本地设备 BIOS 支持虚拟化

(1）检查 BIOS 支持 Intel VT 或 AMD SVM

确认 BIOS 选项中有虚拟化支持并开启

“Intel(R) Virtualization Tech [Enable]

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装 KVM 的前期准备工作

2) 确认本地设备支持虚拟化

(1) 检查 CPU 信息 (是否支持 Intel VT 或 AMD SVM)

#cat /proc/cpuinfo | grep -e vmx -e nx -e svm

(注 : 从 2.6.15 开始 cpuinfo 才支持虚拟化)

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装 KVM 的前期准备工作

2) 本地资源剩余空间充足

(2) 确认内存大小

#grep -e MemTotal /proc/meminfo

(3) 确认可用内存

#free -m

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装 KVM 并检测

1)YUM 安装 KVM

#yum install kvm virt-manager libvirt libvirt-python python-virtinst libvirt-client qemu-kvm qemu-img

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装 KVM 并检测

1)YUM 安装 KVM(简单版)

#yum install kvm virt-manager libvirt*

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装 KVM 并检测

kvm: 核心套件

virt-manager: 图形化 KVM 管理软件

libvirt: 提供虚拟机与宿主相互通信的机制

libvirt-python: 允许使用 libvirt API

python-virtinst:CLI 下创建 KVM 的工具

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

安装 KVM 并检测

libvirt-client: 提供 client 访问 kvm 服务器的机制，并包含 virsh 命令进行管理和控制 VMs

qemu-kvm: 提供用户级 KVM 环境

qemu-img:VMs 磁盘管理

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

2) 如果无法连入互联网，可自行建立源来完成安装并避免各种关联。

- 2.1）建立一个本地的 YUM 源

< 挂载 RHEL 光盘 >

#mount /dev/cdrom /mnt/cdrom

< 建立 YUM 本地源 >

#cd /etc/yum.repos.d

#vim kvms.repo

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

2.1)kvms.repo 内容

```
[kvms]
name=Red Hat
baseurl=file:///mnt/cdrom/
enabled=1
gpgcheck=0
```

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

3) 启动 KVM

1) modprobe kvm ← 加载 kvm 模块

2) lsmod | grep kvm ← 查看加载成功

3) systemctl start libvirtd

4) systemctl status libvirtd

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

3) 启动并检测 KVM

4) virsh –c qemu:///system list

如出现

Id Name State

则安装成功

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

5) 在 GUI 模式下安装虚拟机，启动虚拟系统管理器

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

更改原来 NAT 的网络链接模式为 Bridge

NAT 网络链接的特点？

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

更改原来 NAT 的网络链接模式为 Bridge

NAT (默认上网 ） 虚拟机利用 host 机器的 ip 进行上网 . 对外显示一个 ip
Bridge 将虚拟机桥接到 host 机器的网卡上 ,guest 和 host 机器都通过 bridge 上网 . 对外不同的 ip
网桥的基本原理就是创建一个桥接接口 br0 ，在物理网卡和虚拟网络接口之间传递数据。

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

添加 br0 网卡的配置文件
Cd /etc/sysconfig/network-scripts
cp ifcfg-em1 ifcfg-br0

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

Vim ifcfg-br0

TYPE="Bridge"
BOOTPROTO="static"
NAME="br0"
DEVICE="br0"
ONBOOT="yes"
IPADDR="192.168.4.200"
NETMASK="255.255.255.0"
GATEWAY="192.168.4.1"

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

Vim ifcfg-em1

TYPE="Ethernet"
NAME="em1"
DEVICE="em1"
ONBOOT="yes"
BRIDGE="br0"

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

Ip a

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: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master br0 state UP qlen 1000 link/ether 00:26:b9:53:47:f9 brd ff:ff:ff:ff:ff:ff

3: em2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN qlen 1000 link/ether 00:26:b9:53:47:fa brd ff:ff:ff:ff:ff:ff

4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN link/ether 52:54:00:12:51:9b brd ff:ff:ff:ff:ff:ff
inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
valid_lft forever preferred_lft forever

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

5: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master virbr0 state DOWN qlen 500link/ether 52:54:00:12:51:9b brd ff:ff:ff:ff:ff:ff
7: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP link/ether 00:26:b9:53:47:f9 brd ff:ff:ff:ff:ff:ff inet 192.168.4.200/24 brd 192.168.4.255 scope global br0   valid_lft forever preferred_lft forever
inet6 fe80::226:b9ff:fe53:47f9/64 scope link valid_lft forever preferred_lft forever

12: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br0 state UNKNOWN qlen 500 link/ether fe:54:00:e6:a5:03 brd ff:ff:ff:ff:ff:ff inet6 fe80::fc54:ff:fee6:a503/64 scope link valid_lft forever preferred_lft forever

14: vnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br0 state UNKNOWN qlen 500 link/ether fe:54:00:26:55:40 brd ff:ff:ff:ff:ff:ff inet6 fe80::fc54:ff:fe26:5540/64 scope link valid_lft forever preferred_lft forever

虚拟化 ---KVM
（ kernel-based VirtualMachine ）

验证网络设置是否正确？？
虚拟机之间互相通信