# 1、虚拟化技术

虚拟化技术是将底层物理设备（CPU、内存、外设等）与上层操作系统、应用软件分离的一种解耦技术，在一台物理机上逻辑划分出多个虚拟机使用这些计算资源。

虚拟化类型一般分为两类：I型虚拟化和II型虚拟化。

## （1）I型虚拟化

Hypervisor直接安装在物理机上，多个虚拟机在Hypervisor上运行。Hypervisor实现方式一般是一个特殊定制的Linux系统。Xen和VMWare的ESXi都属于这个类型。

## （2）II虚拟化

物理机上首先安装常规的操作系统，比如RedHat、Ubuntu和Windows。Hypervisor作为OS上的一个程序模块运行，并对管理虚拟机进行管理。KVM、VirtualBox和VMWare Workstation都属于这个类型。

目前虚拟化技术，大致包括：软件模拟、虚拟化层翻译、容器虚拟化。

## （3）软件模拟

软件模拟是通过软件完全模拟CPU、内存、网卡、磁盘等计算机硬件设备，其优点在于可以模拟任何硬件设备，但是其效率非常低，不具备实际应用价值。

采用该技术的产品有QEMU。

## （4）虚拟化层翻译

Hypervisor是一种运行在物理服务器和操作系统之间的中间软件层，可允许多个操作系统和应用共享一套基础物理硬件，因此也可以看作是虚拟环境中的“元”操作系统，它可以协调访问服务器上的所有物理设备和虚拟机，也叫虚拟机监视器（Virtual Machine Monitor）。Hypervisor是所有虚拟化技术的核心，非中断地支持多工作负载迁移的能力是Hypervisor的基本功能。当服务器启动并执行Hypervisor时，它会给每一台虚拟机分配适量的内存、CPU、网络和磁盘，并加载所有虚拟机的客户操作系统。

该技术又分为三种类型：

1. 全虚拟化：通过虚拟化引擎VMM，捕获虚拟机的指令并处理。

说明：这里的处理不是完完整整的将虚拟机的指令传递给宿主机，因为虚拟机传递的是越级的指令，所以VMM需要做些额外处理工作，比如虚拟机执行reboot，宿主机不会重启。

采用该技术的产品有VMWare ESXi。

1. 半虚拟化：全虚拟化方式由于在虚拟化层做了大量的“翻译”工作，效率

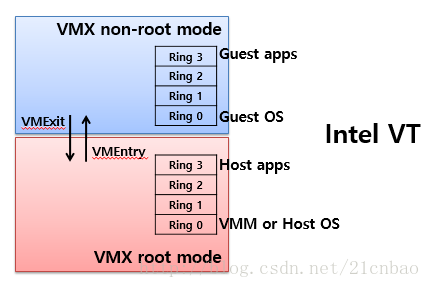
较低，所以XEN提出将虚拟化层的部分功能剥离，放到虚拟机内部处理，这就要求对虚拟机内核进行改造。

采用该技术的产品有XEN、Hyper-V。

1. 硬件支持的全虚拟化：前面的两种虚拟化方式都是在软件层面上进行的，

后来芯片厂商Intel推出了硬件解决方案，对CPU指令进行改造，即VT-x（AMD的硬件解决方案为AMD-v），其效率非常高。

VT-x增加了两种操作模式：VMX root模式和VMX non-root 模式，虚拟化引擎VMM运行在VMX root模式，虚拟机运行在VMX non-root模式，这两种操作模式都支持ring0~ring4这4个特权级。



对于虚拟机而言，Guest OS运行在non-root模式的ring0，其上的应用程序运行在non-root模式的ring3；而对于物理机而言，host OS和VMM则运行于root模式下的ring0，物理机的应用程序运行在root模式的ring3。Guest OS下发的越级指令，会VMExit到root模式下ring0内的VMM，从root环境到non-root环境叫做VMEntry，实际使用时在二者之间不断切换。

采用该技术的产品有KVM。

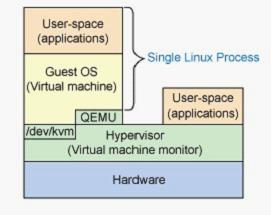
## （5）容器虚拟化

容器虚拟化基于CGroup和Namespace技术实现进程隔离的，每个进程有自己隔离的资源、根目录、进程号、内存空间等。但是，Docker在网络、磁盘上还受限制。

# KVM概述

KVM全称是Kernel-Based Virtual Machine，也就是说KVM是基于Linux内核实现的。

## （1）KVM架构



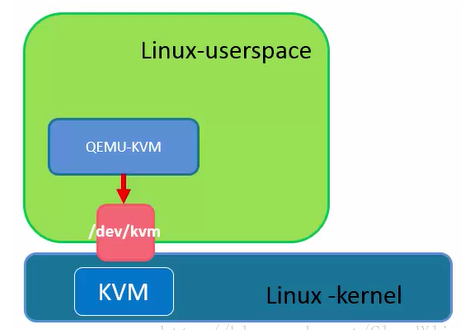
Guest：客户机系统，包括CPU（vCPU）、内存（vMem）、驱动（网卡、I/O 设备驱动等），被 KVM 置于一种受限制的CPU（即non-root）模式下运行。

KVM：运行在内核空间，提供CPU和内存的虚级化，以及客户机的I/O拦截。Guest的I/O被KVM拦截后，交给QEMU处理。

QEMU：修改过的为KVM虚机使用的QEMU代码，运行在用户空间，提供硬件I/O虚拟化，通过ioctl /dev/kvm设备和KVM交互。

## （2）QEMU与KVM

QEMU是一个硬件模拟器，内建HDCP、DNS、TFTP等服务器，但是其速度慢实用价值不大。而KVM只是内核的一个模块，它没有用户空间的管理工具，KVM的虚拟机可以借助QEMU的管理工具管理，QEMU也可以借助KVM实现加速。现在，KVM与QEMU已经结合在一起。



/dev/kvm是KVM内核模块提供给用户空间的一个接口，这个接口被qemu-kvm调用，通过ioctl系统调用就可以给用户提供一个工具用于管理虚拟机生命周期。qemu-kvm就是通过open()、close()、ioctl()等方法打开、关闭、调用这个接口实现与KVM交互的。

## （3）libvirt与KVM

libvirt是Redhat开源的虚拟化管理工具，除了能够管理KVM这种Hypervisor，还可以管理Xen、VirtualBox等。

主要有3部分组成：

1. lib库：支持C、Python、Ruby等编程语言；
2. libvirtd服务
3. virsh命令行工具

libvirt可以管理不同的虚拟化引擎，但是目前实际应用环境中只有KVM在使用。可以实现虚拟机的生命周期管理以及外设的添加。

# 3、环境搭建

## （1）CPU检测

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

输出结果大于0表示支持虚拟化，如果等于0则需要手动选择支持虚拟化。

## （2）安装KVM

yum install qemu-kvm

安装虚拟化管理工具：

yum install virt-manager libvirt libvirt-python python-virtinst bridge-utils

## （3）加载kvm.ko

modprobe kvm

查看驱动是否加载成功：

lsmod | grep kvm

如果加载成功会有输出信息，没有加载成功grep不到指定驱动信息。

## （4）启动libvirt

service libvirtd restart

chkconfig libvirtd on

## （5）创建系统盘

qemu-img create -f qcow2 /kvmtest/centos-7.qcow2 10G

## （6）启动虚拟机

virt-install --virt-type kvm --name centos-7 --ram 1024 \

--vcpus 1 \

--cdrom=/kvmtest/CentOS-7-x86\_64-minimal.iso \

--disk /kvmtest/centos-6.8.qcow2,format=qcow2 \

--network network=default \

--graphics vnc,listen=0.0.0.0 --noautoconsole \

--os-type=linux --os-variant=rhel6

各参数的含义如下：

-n NAME 指定虚拟机的名称

-r MEMORY 指定虚拟机的内存用量

-u UUID 指定虚拟机的唯一通用标识符（UUID）。

省略这个参数时，virt-install 将会自动产生

–vcpus=VCPUS 指定虚拟机的虚拟 CPU （Virtual CPU，VCPU） 数量

-f DISKFILE 指定虚拟磁盘的文件路径名称

-s DISKSIZE 用来指定虚拟磁盘的大小，这个参数需配合-f 使用。

DISKSIZE为虚拟磁盘的大小，单位是GB

-m MAC 指定虚拟机的网络卡之硬件地址。

这个参数可以省略，省略时virt-install 将自动产生

-p 以半虚拟化的方式建立虚拟机

-l LOCATION 指定安装来源

## （7）VNC连接虚拟机

netstat -natp

# 4、虚拟化技术

KVM虚拟化主要包括CPU虚拟化、内存虚拟化、网络虚拟化、存储虚拟化。

其中，KVM虚拟机CPU的软件调休主要是借助NUMA技术；内存方面的调优手段主要是KSM，即相同的内存页合并、内存气球技术及大页内存的使用。这些都在后面专门详述。