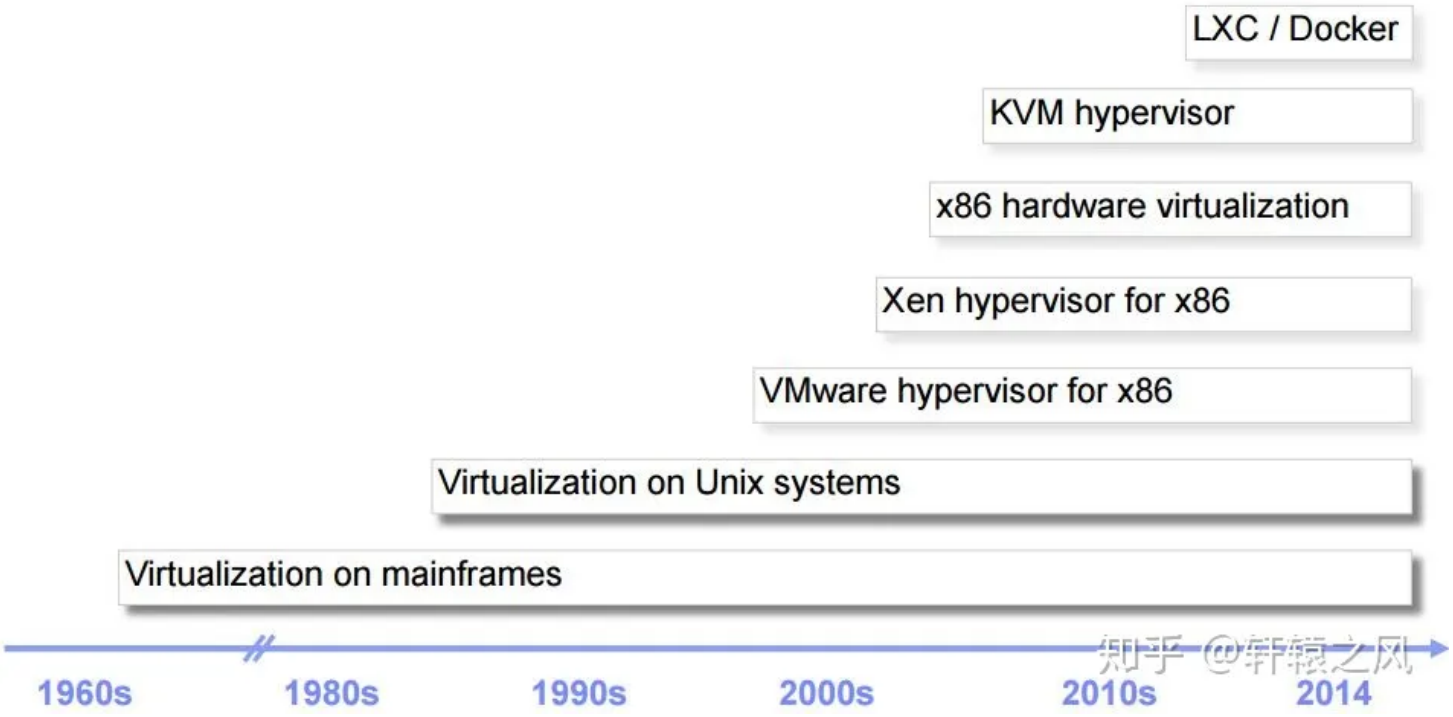
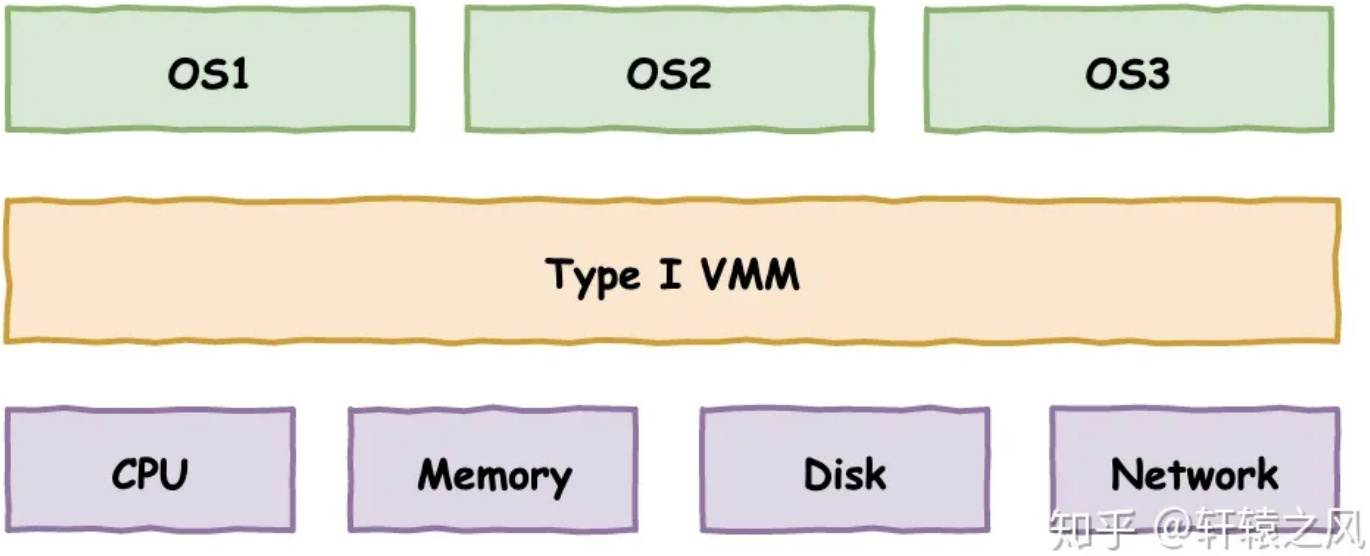
虚拟化简介

- ref
  - https://zhuanlan.zhihu.com/p/272202324

基本发展过程 —— from轩辕之风（懂了！VMware、KVM、Docker原来是这么回事儿）





- 直接凌驾于硬件之上，构建出多个隔离的操作系统环境
- VMware ESXi，直接安装在裸金属之上，不需要额外的操作系统，属于**该类虚拟化**

正好自己操作一下ESXi的裸金属安装

- ref
  - https://xvcat.com/post/2842

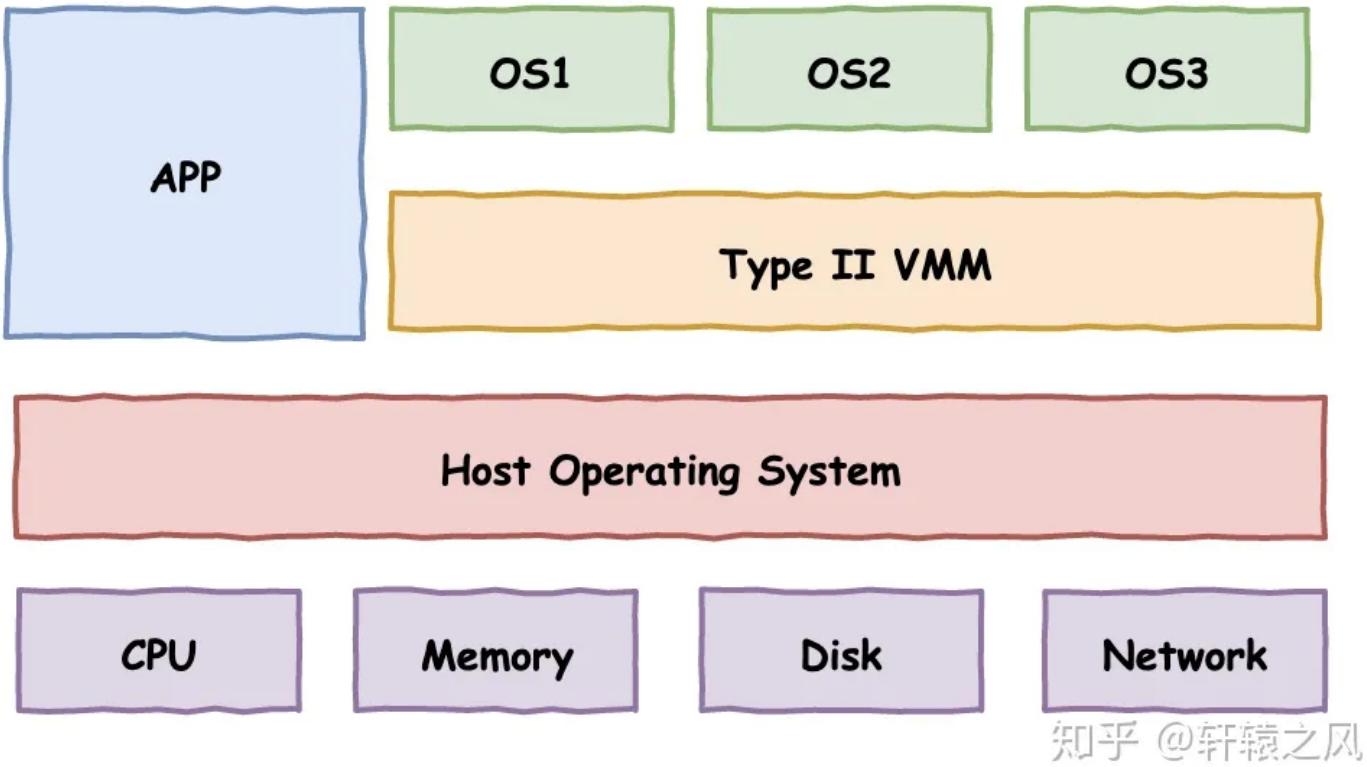
- 首先官网下载试用版，需要注册，**不仅仅注册Vmware，还需要注册对应的产品**，会下载到一个 `.iso` 文件，然后把它丢到一个**ventoy的启动U盘中**
- 然后顺着网上的安装教程安装完毕（俊伟之前安装失败是那台机器确实有一个网卡不正常，但是他没有观察网线插上之后都**不带亮灯**的），全部安装成功之后一切正常，可以在Web完成登录（**在这个界面可操作性就大了**）
  - 有一个关键的地方可以注意一下，安装完成后它是这样来显示的
    - `VMware ESXi 8.0.2 (VMKernel Release Build xxxx)` —— 所以它有一个名字**VMKernel**
- Web界面的内容相对直观，首先在Web中开启SSH服务之后，**就可以直接ssh登录的说了**，先看看通过ssh命令行能get到什么的说
  - 很直观，ESXi使用了busybox
    - 但是确实看起来并不是Linux，虽然也有 `/` 目录
  - 一些可以与Linux无缝链接的命令包括但不限于 ——

    `lspci/df -h(所以ESXi的内核是包含文件系统的VMFS，下面会详叙)/...`
  - 关于文件系统
    - 文件系统名为VMFS，存在挂载点的概念
- 正好看到一个帖子是在说ESXi的内核是否是Linux内核的问题，这里刚好的记录一下
  - ref

        - https://www.quora.com/Is-ESXi-a-firmware-or-an-OS-software
    - 首先应该明确，Linux kernel的license就决定了当前的ESXi不会是Linux Kernel，因为ESXi是一个**闭源产品**，这与Linux kernel的license是背道而驰的
        - 一些标准的说法是，ESX阶段是Linux Kernel，但是到了ESXi的阶段就不使用Linux Kernel了。而且因为license，**Linux kernel还起诉过Vmware**
    - 回答中，看起来有一个VMware员工的回答，比较详细且可信度也比较高 —— from Kit Colbert (Engineer at VMware) —— **他的回答主要表明ESXi内核与其它OS内核在设计上的决策是不同的**
        - 大多数的OS都对**运行应用程序**进行了优化，而vmkernel则对**运行OS**进行了优化
        - 文件系统VMFS，默认块大小是1MB，可以在Web界面中得到直接的验证
            - 至少在这种情况下，VMware向上暴露的存储是**文件**，所以上层的guest OS是直接**使用的文件作为存储**（这个和我在Linux下想的还真不一样，**但是毕竟VMkernel完全是定制的**）
            - VMFS的块为什么大呢
                - 因为vmfs上的大多数文件都是巨大的vmdk文件，作为guest OS的虚拟盘（已GB为单位）
        - 关于写缓冲区
            - VMkernel原本是没有缓冲区的，直接要求对齐写入到磁盘中
            - **但是在变化...**
        - 被调度的实体被称为**world**
            - 其实这个是类似于**线程**的概念，VMKernel为什么要称之为world呢，因为对于VMKernel来说，它切换的是OS，包括CPU寄存器，页表等全部。所以一般来说切换任务要重，所以称之为world（这个名字其实很**贴切**）
            - 在终端中，ps也可以看到WID等的字眼
        - 资源管理
            - 先说一下Linux，Linux的资源如果被耗尽，就开始随机杀死task（**这个我不确定**）
            - ESXi不会这样，它保证用户的虚拟机不会发生这种情况
        - ...
    - ESXi是Linux吗
        - No, ESXi based on kernel called **VMKernel** which is propraiatery kernel made by VMWare. the rest of the OS/HyperVisor is **partially POSIX compliant**, that's why it looks **similar to Linux**

关于我所关注的HBA驱动

- 博通9400-16i —— 对应驱动**lsi_msgpt35**(ESXi 8.0.2)



- 依赖于宿主操作系统，在其上构建出多个隔离的操作系统环境
- VMware WorkStation，则属于该类虚拟化

陷阱 & 模拟

- 那么应该如何实现虚拟化呢，典型的做法是**陷阱 & 模拟**
  - 将虚拟机中的指令直接拿到CPU上去执行，如果遇到**敏感指令，就触发异常，控制流交给VMM**，由VMM来进行对应的处理，这就可以构造一个虚拟的计算机环境
  - 但是在x86上遇到了问题
    - x86有4个Ring，有**特权指令**的说法，特权指令只能在**Ring0**下运行，如果在Ring3运行特权指令，则会**抛出异常**。听起来也没什么问题，甚至有那么一丝完美
      - 但是x86的**敏感指令不全是特权指令**
  - 所以应该如何让x86架构的CPU也能支持虚拟化呢?
    - Vmware
      - VMware创造性的提出了**二进制翻译**技术
        - VMM在虚拟机操作系统和宿主计算机之间扮演一个**桥梁**的角色，将虚拟机中的要执行的指令**翻译**成**恰当的指令**在宿主物理计算机上执行，以此来模拟执行虚拟机中的程序

        - 了提高性能，也并非所有的指令都是模拟执行的，VMware在这里做了不少的优化，对一些**安全**的指令，就让它直接执行也未尝不可。所以VMware的二进制翻译技术也融合了部分的直接执行
        - 对于虚拟机中的操作系统，VMM需要完整模拟底层的硬件设备，包括处理器、内存、时钟、I/O设备、中断等等，换句话说，VMM用**纯软件**的形式**模拟**出一台计算机供虚拟机中的操作系统使用
            - 这种完全模拟一台计算机的技术也称为**全虚拟化**，这样做的好处显而易见，虚拟机中的操作系统感知不到自己是在虚拟机中，代码无需任何改动，直接可以安装
            - 而缺点也是可以想象
                - 完全用软件模拟，转换翻译执行，**性能堪忧**
    - QEMU
        - 而QEMU则是**完全软件层面的模拟**，乍一看和VMware好像差不多，不过实际**本质是完全不同的**
            - VMware是将原始CPU指令序列翻译成经过处理后的CPU指令序列来执行
            - 而QEMU则是完全模拟执行整个CPU指令集，更像是**解释执行**，两者的性能不可同日而语

半虚拟化

- 如果能够把操作系统中所有执行敏感指令的地方都改掉，改成一个接口调用（**HyperCall**），就会很省事。这就是半虚拟化，**典型代表是Xen**，但是最大的问题是需要**修改源码**
- 来说说Xen
    - Xen是运行在裸机上的虚拟化管理程序（HyperVisor），是半虚拟化（Para-Virtualization）技术的典型代表
    - Para-Virtualization可以理解为在Guest VM旁边运行着的**管理VM**，**Xen称这个特别的VM为Dom0**，虚拟机操作系统被称做DomU（**怎么感觉好像Dell PowerStore X**）
    - **管理VM（Dom0）** 负责管理整个硬件平台上的**所有输入输出设备驱动**，半虚拟化中的Hypervisor**不对I/O设备作模拟**，而仅仅对CPU和内存做模拟
        - 这就是**Para-Virtualization**被翻译成**半虚拟化**的原因
    - 半虚拟化还有一个叫法 —— 操作系统辅助虚拟化（OS Assisted Virtualization）
        - 这是因为**Guest VM自身不带设备驱动**，需要向**管理VM**寻求帮助。这种虚拟化技术允许虚拟化操作系统**感知**到自己运行在XEN HyperVisor上而不是直接运行在硬件上，同时也可以识别出其他运行在相同环境中的虚拟机
            - **最大的一个特点是虚拟机知道自己是虚拟机**
- 相对于**VMwareESX/ESXi**和**微软Hyper-V**来说，Xen支持**更广泛**的CPU架构
    - 前两者只支持CISC的X86/X86_64 CPU架构，**Xen除此之外还支持RISC CPU架构，如IA64、ARM等**

    - Xen的Hypervisor是服务器经过BIOS启动之后载入的首个程序，然后启动一个具有特定权限的虚拟机（即Dom 0）。**Dom 0的操作系统可以是Linux或Unix**，它实现对Hypervisor控制和管理功能
        - 在所承载的虚拟机中，Dom0是**唯一**可以直接访问物理硬件（如存储和网卡）的**虚拟机**，它通过本身加载的物理驱动，为其它虚拟机（即DomU）提供访问存储和网卡的桥梁
- XEN需要修改操作系统内核，Windows操作系统由于其封闭性，不能被Xen的半虚拟化所支持
    - 为了解决这个问题，**Xen也支持全虚拟化**（Full Virtualization）。Xen称其为HVM（Hardware Virtual Machine）
- Xen的Hypervisor层非常精简，少于15万行的代码量，不包含任何物理设备驱动，这一点与Hyper-V是非常类似的，物理设备的驱动均是驻留在Dom 0中，可以重用现有的Linux设备驱动程序
    - **因此，Xen对硬件兼容性也是非常广泛的，Linux支持的，它就支持**

硬件辅助虚拟化

- CPU厂商说，你们别瞎玩了，我来吧
    - 代表就是**Intel的VT**和**AMD的AMD-v**
- 简单来说，在**Ring0-Ring3**的基础上，引入了工作模式的概念
    - **Root模式与No Root模式**
- 现在新的CPU告诉VMM
    - 不用那么麻烦了，你提前告诉我你对哪些指令哪些事件感兴趣，我在执行这些指令和发生这些事件的时候就通知你，你就可以实现掌控了
    - 完全由硬件层面提供支持，**性能自然高了不少**
- VMware从5.5版本开始引入对硬件辅助虚拟化的支持，随后在2011年的8.0版本中正式全面支持
    - 于是乎，我们在创建虚拟机的时候，可以选择要使用哪一种虚拟化引擎技术，是用原先的**二进制翻译执行**，还是基于**硬件辅助虚拟化**的新型技术
- 同一时期的XEN从3.0版本也加入对**硬件辅助虚拟化**的支持，从此**基于XEN的虚拟机中也能够运行Windows**系统了

KVM-QEMU

- 详见 `iov.md`
- 有了硬件辅助虚拟化的加持，虚拟化技术开始呈现井喷之势。**VirtualBox、Hyper-V、KVM等技术**如雨后春笋般接连面世。这其中在云计算领域声名鹊起的当属**开源的KVM**技术了
    - KVM全称**for Kernel-based Virtual Machine**，意为**基于内核**的虚拟机
- KVM本身基于硬件辅助虚拟化，仅仅实现CPU和内存的虚拟化，但一台计算机不仅仅有CPU和内存，还需要各种各样的I/O设备，不过KVM不负责这些

- 这个时候，**QEMU就和KVM搭上了线**，经过改造后的QEMU，负责外部设备的虚拟，KVM负责底层执行引擎和内存的虚拟，两者彼此互补，成为新一代云计算虚拟化方案的宠儿

容器技术-LXC & Docker

- 之前虚拟化的目标都是一台**完整**的计算机，但是应用程序需要怎么多么？实际上并非如此，应用程序可能会说一句：**你其实可以不用这样辛苦的**
- 确实存在这样的情况，虚拟机中的程序说
  - 我只是想要一个**单独的执行环境**，不需要你**费那么大劲**去虚拟出一个完整的计算机来
- 容器技术就是在这样的环境下诞生的。不同于虚拟化技术要完整虚拟化一台计算机，容器技术更像是操作系统层面的虚拟化，**它只需要虚拟出一个操作系统环境**
- **LXC技术**
  - LXC全称是LinuX Container，**通过Linux内核的Cgroups技术和namespace技术的支撑**，隔离操作系统文件、网络等资源，在原生操作系统上隔离出一个单独的空间，将应用程序置于其中运行，这个空间的形态上类似于一个容器将应用程序包含在其中，故取名容器技术
- **Docker**
  - 如今各个大厂火爆的Docker技术底层原理与LXC并不本质区别，甚至在早期Docker就是直接基于LXC的高层次封装。Docker在LXC的基础上更进一步，将执行执行环境中的各个组件和依赖打包封装成独立的**镜象，更便于移植和部署**
- 优缺点
  - 容器技术的好处是轻量，所有隔离空间的程序代码指令不需要翻译转换，就可以直接在CPU上执行，大家底层都是同一个操作系统，通过软件层面上的逻辑隔离形成一个个单独的空间
  - 容器技术的缺点是**安全性**不如虚拟化技术高，毕竟软件层面的隔离比起硬件层面的隔离要弱得多。隔离环境系统和外面的主机共用的是同一个操作系统内核，一旦利用内核漏洞发起攻击，程序突破容器限制，实现逃逸，危及宿主计算机，安全也就不复存在

超轻虚拟化firecracker

- 如何即轻量又安全呢，亚马逊提出的firecracker就是一个典型代表