懂了！VMware/KVM/Docker原来是这么回事儿

https://mp.weixin.qq.com/s/m1HmYMxMCDCQT0b2GkArMg

轩辕之风O Sun Mar, 28 23:43

云计算时代，计算资源如同小马哥当年所言，已经成为了互联网上的水和电。

虚拟主机、web服务器、数据库、对象存储等等各种服务我们都可以通过各种各样的云平台来完成。

而在云计算欣欣向荣的背后，有一个重要的功臣，那就是**虚拟化技术**。可以毫不客气的说，没有了虚拟化技术，云计算无从谈起。

说起虚拟化你会想到什么？从我们常用的虚拟机三件套VMware、VirtualPC、VirutalBox到如今大火的KVM和容器技术Docker？

这些技术是什么关系，背后的技术原理是怎样的，又有什么样的区别，各自应用的场景又是什么样的？

看完这篇文章，相信大家都能回答上面问题。

历史背景

什么是虚拟化技术？

维基百科中的解释是这样的：

> 虚拟化（技术）是一种资源管理技术，是将计算机的各种实体资源（CPU、内存、磁盘空间、网络适配器等），予以抽象、转换后呈现出来并可供分割、组合为一个或多个电脑配置环境。

对于一台计算机，我们可以简单的划分为三层：从下到上依次是物理硬件层，操作系统层、应用程序层

1974年，两位计算机科学家Gerald Popek 和 Robert Goldberg发表了一篇重要的论文**《虚拟化第三代体系结构的正式要求》**，在这篇论文中提出了虚拟化的三个基本条件：

- 等价性：程序在本地计算机执行和在虚拟机中执行应该表现出一样的结果（不包括执行时间的差异）
- 安全性：虚拟机彼此隔离，与宿主计算机隔离
- 性能：绝大多数情况下虚拟机中的代码指令应该直接在物理CPU中执行，少部分特殊指令可由VMM参与。

那如何实现对计算机底层的物理资源的虚拟化分割呢？在计算机技术的发展历史上，出现了两种著名的方案，分别是I型虚拟化和II型虚拟化

I型虚拟化

II型虚拟化

图中的VMM意为Virtual Machine Monitor，虚拟机监控程序，或者用另一个更专业的名词：**HyperVisor**

从图中可以清楚的看到两种虚拟化方案的区别：

Type I: 直接凌驾于硬件之上，构建出多个隔离的操作系统环境

Type II: 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

我们熟知的VMware事实上有两个产品线，一个是VMware ESXi，直接安装在裸金属之上，不需要额外的操作系统，属于第一类虚拟化。另一个是我们普通用户更加熟知的VMware WorkStation，属于第二类虚拟化。

如何实现上述的虚拟化方案呢？

一个典型的做法是—— 陷阱 & 模拟 技术

什么意思？**简单来说就是正常情况下直接把虚拟机中的代码指令放到物理的CPU上去执行，一旦执行到一些敏感指令，就触发异常，控制流程交给VMM，由VMM来进行对应的处理，以此来营造出一个虚拟的计算机环境。**

不过这一经典的虚拟化方案在Intel x86架构上却遇到了问题。

全虚拟化：VMware 二进制翻译技术

不同于8086时代16位实地址工作模式，x86架构进入32位时代后，引入了保护模式、虚拟内存等一系列新的技术。同时为了安全性隔离了应用程序代码和操作系统代码，其实现方式依赖于x86处理器的工作状态。

这就是众所周知的x86处理器的Ring0-Ring3四个“环”。

操作系统内核代码运行在最高权限的Ring0状态，应用程序工作于最外围权限最低的Ring3状态，剩下的Ring1和Ring2主流的操作系统都基本上没有使用。

这里所说的权限，有两个层面的约束：

- 能访问的内存空间
- 能执行的特权指令

来关注一下第二点，特权指令。

CPU指令集中有一些特殊的指令，用于进行硬件I/O通信、内存管理、中断管理等等功能，这一些指令只能在Ring0状态下执行，被称为特权指令。这些操作显然是不能让应用程序随便执行的。处于Ring3工作状态的应用程序如果尝试执行这些指令，CPU将自动检测到并抛出异常。

回到我们的主题虚拟化技术上面来，如同前面的定义所言，虚拟化是将计算资源进行逻辑或物理层面的切割划分，构建出一个个独立的执行环境。

按照我们前面所说的 陷阱 & 模拟 手段，可以让虚拟机中包含操作系统在内的程序统一运行在低权限的Ring3状态下，一旦虚拟机中的操作系统进行内存管理、I/O通信、中断等操作时，执行特权指令，从而触发异常，物理机将异常派遣给VMM，由VMM进行对应的模拟执行。

这本来是一个实现虚拟化很理想的模式，不过x86架构的CPU在这里遇到了一个跨不过去的坎。

到底是什么问题呢？

回顾一下前面描绘的理想模式，要这种模式能够实现的前提是执行敏感指令的时候**能够触发异常**，让VMM有机会介入，去模拟一个虚拟的环境出来。

但现实是，x86架构的CPU指令集中有那么一部分指令，它不是特权指令，Ring3状态下也能够执行，但这些指令对于虚拟机来说却是敏感的，不能让它们直接执行。一旦执行，没法触发异常，VMM也就无法介入，虚拟机就**露馅儿**了！

这结果将导致虚拟机中的代码指令出现无法预知的错误，更严重的是影响到真实物理计算机的运行，虚拟化所谓的安全隔离、等价性也就无从谈起。

怎么解决这个问题，让x86架构CPU也能支持虚拟化呢？

VMware和QEMU走出了两条不同的路。

VMware创造性的提出了一个**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。你可以简单理解成Java虚拟机执行Java字节码的过程，不同的是Java虚拟机执行的是字节码，而VMM模拟执行的就是CPU指令。

另外值得一提的是，为了提高性能，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！

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：Xen 内核定制修改

既然有全虚拟化，那与之相对的也就有半虚拟化，前面说了，由于敏感指令的关系，全虚拟化的VMM需要捕获到这些指令并完整模拟执行这个过程，实现既满足虚拟机操作系统的需要，又不至于影响到物理计算机。

但说来简单，这个模拟过程实际上相当的复杂，涉及到大量底层技术，并且如此模拟费时费力。

而试想一下，如果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**），接口的提供方VMM实现对应处理，省去了捕获和模拟硬件流程等一大段工作，性能将获得大幅度提升。

这就是**半虚拟化**，这项技术的代表就是**Xen**，一个诞生于2003年的开源项目。

这项技术一个最大的问题是：需要修改操作系统源码，做相应的适配工作。这对于像Linux这样的开源软件还能接受，充其量多了些工作量罢了。但对于Windows这样闭源的商业操作系统，修改它的代码，无异于痴人说梦。

硬件辅助虚拟化 VT / AMD-v

折腾来折腾去，全都是因为x86架构的CPU天然不支持经典虚拟化模式，软件厂商不得不想出其他各种办法来在x86上实现虚拟化。

如果进一步讲，CPU本身增加对虚拟化的支持，那又会是一番怎样的情况呢？

在软件厂商使出浑身解数来实现x86平台的虚拟化后的不久，各家处理器厂商也看到了虚拟化技术的广阔市场，纷纷推出了硬件层面上的虚拟化支持，正式助推了虚拟化技术的迅猛发展。

这其中为代表的就是Intel的VT系列技术和AMD的AMD-v系列技术。

硬件辅助虚拟化细节较为复杂，简单来说，新一代CPU在原先的Ring0-Ring3四种工作状态之下，再引入了一个叫工作模式的概念，有 `VMX root operation` 和 `VMX non-root operation` 两种模式，每种模式都具有完整的Ring0-Ring3四种工作状态，前者是VMM运行的模式，后者是虚拟机中的OS运行的模式。

VMM运行的层次，有些地方将其称为Ring -1，VMM可以通过CPU提供的编程接口，配置对哪些指令的劫持和捕获，从而实现对虚拟机操作系统的掌控。

换句话说，原先的VMM为了能够掌控虚拟机中代码的执行，不得已采用“中间人”来进行翻译执行，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。

上面只是硬件辅助虚拟化技术的一个简单理解，实际上还包含更多的要素，提供了更多的便利给VMM，包括内存的虚拟、I/O的虚拟等等，让VMM的设计开发工作大大的简化，VMM不再需要付出昂贵的模拟执行成本，整体虚拟化的性能也有了大幅度的提升。

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

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。

在虚拟化底层技术上，KVM和VMware后续版本一样，都是基于硬件辅助虚拟化实现。不同的是VMware作为独立的第三方软件可以安装在Linux、Windows、MacOS等多种不同的操作系统之上，而KVM作为一项虚拟化技术已经集成到Linux内核之中，可以认为Linux内核本身就是一个HyperVisor，这也是KVM名字的含义，因此该技术只能在Linux服务器上使用。

KVM技术常常搭配QEMU一起使用，称为KVM-QEMU架构。前面提到，在x86架构CPU的硬件辅助虚拟化技术诞生之前，QEMU就已经采用全套软件模拟的办法来实现虚拟化，只不过这种方案下的执行性能非常低下。

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

前面谈到的无论是基于翻译和模拟的全虚拟化技术、半虚拟化技术，还是有了CPU硬件加持下的全虚拟化技术，其虚拟化的目标都是一台完整的计算机，拥有底层的物理硬件、操作系统和应用程序执行的完整环境。

为了让虚拟机中的程序实现像在真实物理机器上运行“近似”的效果，背后的HyperVisor做了大量的工作，付出了“沉重”的代价。

虽然HyperVisor做了这么多，但你有没有问过虚拟机中的程序，这是它想要的吗？或许HyperVisor给的太多，而目标程序却说了一句：你其实可以不用这样辛苦。

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。

这样做的好处是什么？

虚拟出一台计算机的成本高还是只虚拟出一个隔离的程序运行环境的成本高？答案很明显是前者。一台物理机可能同时虚拟出10台虚拟机就已经开始感到乏力了，但同时虚拟出100个虚拟的执行环境却还是能够从容应对，这对于资源的充分利用可是有巨大的好处。

近几年大火的容器技术正是在这样的指导思想下诞生的。

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技术就是这种方案的一个典型代表，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。

举个不是太恰当的比喻，一套原来是三居室的房子，被二房东拿来改造成三个一居室的套间，每个一居室套间里面都配备了卫生间和厨房，对于住在里面的人来说就是一套完整的住房。

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对象，更便于移植和部署。

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

虚拟完整的计算机隔离性好但太过笨重，简单的容器技术又因为太过轻量纯粹靠软件隔离不够安全，有没有一个折中的方案同时兼具两者的优点，实现既轻量又安全呢?

近年来，一种**超轻虚拟化**的思想开始流行开来，亚马逊推出的**firecracker**就是一个典型的代表。

firecracker将虚拟化技术的强隔离性和容器技术的轻量性进行融合，提出了一个 `microVM` 的概念，底层通过KVM虚拟化技术实现各个microVM的强隔离，而隔离的虚拟机中运行的是一个个精简版的微型操作系统，砍掉了大量无用的功能，专为容器设计的微型OS。

超轻虚拟化如今成为一个新的浪潮，除了AWS的firecracker，谷歌的gVisor, Intel主导的NEMU也在向这个领域开始发力。

总结

本文简单介绍了虚拟化技术的基本概念和基本要求。随后引出由于早期的x86架构不支持经典的虚拟化方案，各家软件厂商只能通过软件模拟的形式来实现虚拟化，其代表是早期的VMware WorkStation和Xen。

不过纯粹依靠软件的方式毕竟有性能的瓶颈，好在Intel和AMD及时推出了CPU硬件层面的虚拟化支持，软件厂商迅速跟进适配，极大的改善了虚拟化的性能体验。这一时期的代表有新版本的VMware WorkStation、Hyper-V、KVM等。

近年来，随着云计算和微服务的纵深发展，对虚拟化技术的虚拟粒度逐渐从粗到细。从最早的虚拟化完整的计算机，到后来只需虚拟出一个操作系统，再到后来虚拟出一个微服务需要的环境即可，以Docker为代表的容器技术在这个时期大放异彩。

技术的发展总是伴随着市场的发展需要而不断演进，虚拟化的未来是怎样的，你有什么样的看法呢，欢迎评论区留言交流。

往期TOP5文章

真惨！连各大编程语言都摆起地摊了！

因为一个跨域请求，我差点丢了饭碗

完了！CPU一味求快出事儿了！

哈希表哪家强？几大编程语言吵起来了！

一个HTTP数据包的奇幻之旅