# 概念介绍

## Emulation和Simulation

这两个词在不同的领域代表的意思不尽相同，甚至其意相反。在计算机领域中，**Emulation**翻译为**仿真**，而**Simulation**翻译为**模拟**，而在自控领域正好相反。

总的来说，目前计算机领域对两个词的理解是这样的：

**仿真（Emulation）**表示对整个系统包括其内部工作原理的抽象模拟，而**模拟（Simulation）**则仅仅模拟该系统的外部行为特征。

## 虚拟化的目的

虚拟化的目的或者说功能包括以下几个：

* **多OS环境**：虚拟化使得可以在一台物理机上运行几个不同的操作系统环境，Windows、Linux、FreeBSD，或者不同版本的Linux可以并存在一台机器上。
* **经济型**：若干个虚拟环境共同运行在一个物理机上可以省去额外购买硬件的费用，通过配置工具灵活的分配虚拟机的运行地点则可以进一步增加这种经济型。
* **独立性**：虚拟环境的用户可以被视为独占一个OS，无需考虑其他用户，任意的设定自己的配置。
* **安全性**：虚拟化使得虚拟机/容器变成一个沙盒，运行于其上的程序不会影响到其外的环境。
* **移植性**：虚拟环境可以以文件的形式迅速的迁移到其他的物理机器上并运行，无需考虑各种环境变量、不同库版本、系统配置等等（这些都包含在虚拟环境内）。而程序虚拟机比如Java则可以“一次编译，到处运行”

## 虚拟机和容器

**Virtual Machine（VM）**是对某种计算机系统的模拟，其实现可能由硬件/软件达成。一个狭义点的虚拟机概念指的是**硬件虚拟机**。而一个广义点的虚拟机概念也包括**程序虚拟机（Application VM，比如JVM）**。

虚拟机和容器的区别在于，虚拟机试图虚拟出一个硬件，而容器试图隔离出一个独立的环境，虽然这两者之间的界限并不是那么清晰。

从独立性和其中所能包括的内容排序如下：

1. 裸金属Hypervisor虚拟机（OS）
2. Hosted Hypervisor虚拟机（OS）
3. OS容器（一个隔绝的OS环境）
4. 应用容器（一个隔绝的应用）
5. 应用虚拟机（一个特定格式的程序）

换句话说，1中可以有2/3/4/5；2中可以有2/3/4/5；3中可以有4/5；4中可以有5。

## 微服务

**微服务 (Microservices)** 是一种[软件架构风格](https://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E6%9E%B6%E6%9E%84" \o "软件架构)，它是以专注于单一责任与功能的**小型功能区块 (Small Building Blocks)** 为基础，利用模块化的方式组合出复杂的大型应用程序，各功能区块使用**与语言无关 (Language-Independent，Language agnostic)** 的 API 集相互通讯。

微服务的起源是于 2005 年度云端运算博览会提出的**微 Web 服务 (Micro-Web-Service)** 开始，将类别变成**细粒服务 (granular services)**，以作为 [Microsoft](https://zh.wikipedia.org/wiki/Microsoft) 下一阶段的软件架构，其核心想法是让服务是由类似 Unix 管道的存取方式使用，而且复杂的服务背后是使用简单 [URI](https://zh.wikipedia.org/wiki/URI) 来开放界面，任何服务，任何细粒都能被开放 (exposed)。这个设计在 HP 的实验室被实现，具有改变复杂软件系统的强大力量。

2014年，[Martin Fowler](https://zh.wikipedia.org/wiki/Martin_Fowler" \o "Martin Fowler) 与 [James Lewis](https://zh.wikipedia.org/w/index.php?title=James_Lewis&action=edit&redlink=1) 共同提出了微服务的概念，定义了微服务是由以单一应用程序构成的小服务，自己拥有自己的进程与轻量化处理，服务依业务功能设计，以全自动的方式部署，与其他服务使用 **HTTP API** 通讯。同时服务会使用最小的规模的集中管理 (例如 [**Docker**](https://zh.wikipedia.org/wiki/Docker)) 能力，服务可以用不同的编程语言与数据库等组件构成。

## 虚拟专用服务器（VPS）

**虚拟专用服务器（VPS，Virtual Private Server），**是将一台[服务器](https://zh.wikipedia.org/wiki/服务器)分割成多个虚拟服务器的服务。实现VPS的技术分为**OS容器技术**和[**虚拟机**](https://zh.wikipedia.org/wiki/虚拟机)**技术** 。在容器或虚拟机中，每个VPS都可分配独立公网[IP地址](https://zh.wikipedia.org/wiki/IP地址)、独立[操作系统](https://zh.wikipedia.org/wiki/操作系统)、实现不同VPS间磁盘空间、[内存](https://zh.wikipedia.org/wiki/内存)、[CPU](https://zh.wikipedia.org/wiki/CPU)资源、进程和系统配置的隔离，为用户和[应用程序](https://zh.wikipedia.org/wiki/应用程序)模拟出“独占”使用计算资源的体验。

## 虚拟化的公司

**VMWare**：**威睿**，是[戴尔科技](https://zh.wikipedia.org/w/index.php?title=%E6%88%B4%E7%88%BE%E7%A7%91%E6%8A%80&action=edit&redlink=1)旗下一家[软件](https://zh.wikipedia.org/wiki/%E9%9B%BB%E8%85%A6%E8%BB%9F%E9%AB%94)[公司](https://zh.wikipedia.org/wiki/%E5%85%AC%E5%8F%B8)，它提供[**云计算**](https://zh.wikipedia.org/wiki/%E4%BA%91%E8%AE%A1%E7%AE%97)和[**硬件虚拟化**](https://zh.wikipedia.org/wiki/%E7%A1%AC%E4%BB%B6%E8%99%9A%E6%8B%9F%E5%8C%96)的软件和服务，并号称是第一个商业化的成功的虚拟化的[x86](https://zh.wikipedia.org/wiki/X86)架构。公司成立于1998年。

**Citrix System Inc.** ：**思杰系统**，成立于1989年的一家软件和[**云运算**](https://zh.wikipedia.org/wiki/%E9%9B%B2%E7%AB%AF%E9%81%8B%E7%AE%97)的[科技公司](https://zh.wikipedia.org/wiki/%E8%B7%A8%E5%9B%BD%E5%85%AC%E5%8F%B8)。它的[跨国](https://zh.wikipedia.org/wiki/%E8%B7%A8%E5%9B%BD%E5%85%AC%E5%8F%B8)业绩包括提供**服务器、软件及**[**桌面虚拟化**](https://zh.wikipedia.org/wiki/%E6%A1%8C%E9%9D%A2%E8%99%9A%E6%8B%9F%E5%8C%96)**、网络连结**、以及[**软件即服务**](https://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E5%8D%B3%E6%9C%8D%E5%8A%A1)**（SaaS）**等产品。

**Parallels**：

**Microsoft**：

**Oracle**：

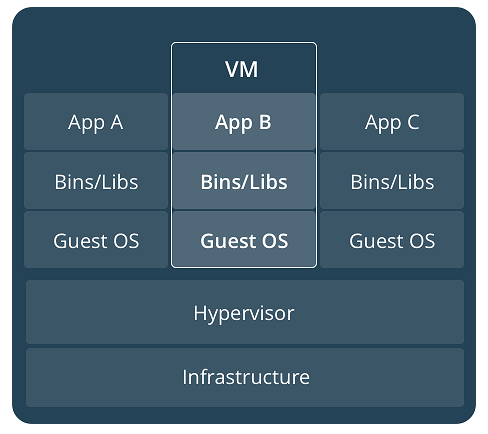
# 硬件虚拟化

**Hardware virtualization**，即**系统虚拟机（System virtual machine）**，简称**虚拟机（VM，Virtual Machine）**。硬件虚拟化实现了对硬件平台（或者其功能）的虚拟，隐藏了硬件的物理特征，而给用户提供了抽象的计算平台，以使得操作系统可以运行其上。控制虚拟化的软件被称为**hypervisor**或**VMM（virtual machine monitor）**。

### 全虚拟化和半虚拟化

**全虚拟化（Full virtualization）**，模拟出完整的硬件环境，允许未经任何修改的**guest OS**（与主机同样的指令集）在其上隔离运行。

**半虚拟化（Paravirtualization）**并不完全模拟硬件，而是提供了一套专门的API，**guest OS**需要作出对应的修改，以便运行在其上。



### 硬件辅助虚拟化

**硬件辅助虚拟化（Hardware-assisted virtualization）**，指的是通过硬件提供的支持实现**全虚拟化**。通常来自于CPU，比如Intel x86架构的 **VT-x**，AMD x86架构的**AMD-V**。

### Hypervisor

又叫**虚拟机监视器（Virtual Machine Monitor，VMM）**，即我们平时所说的虚拟机程序。用来建立与执行虚拟机的软件、硬件或者固件。

被**Hpyervisor**用来执行一个或者多个虚拟机的物理主机被称为**Host machine**，其上的虚拟机则被称为**Guest machine**。Hypervisor提供虚拟的作业平台来执行**Guest OS**，负责管理其他Guest OS的执行阶段。这些guest OS共同分享虚拟化后的硬件资源。

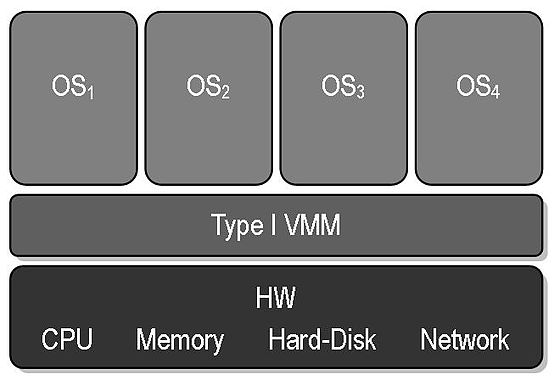
在1974年的一篇论文“Formal Requirements for Virtualizable Third Generation Architecture”中，Hypervisor被分成两种，分别是**类型I（Type I）**和**类型II（Type II）**。

#### 类型I：裸金属Hypervisor

**Bare-metal hypervisor**，又叫**本地Hypervisor（Native hypervisor）**。这些Hypervisor直接运行在主机的硬件来控制硬件和管理客体操作系统。

特点：

1. 需要硬件支持
2. 运行效率高（没有host OS占用资源）



知名类型I Hypervisor有以下这些：

##### Hyper-V（Microsoft）

微软推出的**x86-64**架构上运行Windows虚拟机的裸金属Hypervisor，之前叫**Windows Server Virtualization**。Hyper-V替代了早先的**Windows Virtual PC**和**Virtual Server**（Windows NT的硬件虚拟化组件）。

Hyper-V作为硬件虚拟化组件跟随**Windows Server 2008**首次发布。独立的Hyper-V是免费的，但是只有命令行接口。

* **Host Machine**：x86-64
* **Guest OS**：Windows

##### KVM（开源）

**Kernel-based Virtual Machine**，是一个Linux内核中的虚拟化模块，使得Linux内核可以作为一个**Hypervisor**工作，KVM需要**硬件辅助**（VT、AMD-V）。

另外通过VirtIO接口，KVM也提供了对Linux、BSD、Windows的**半虚拟化**支持。

**RHV（Red Hat Virtualization）**是**红帽公司**提供的，基于KVM的虚拟化产品。

* **Host Machine**：x86、IA-64、S/390、PPC、ARM
* **Guest OS**：Linux、BSD、Solaris、Windows、Haiku、ReactOS、Plan 9、OS X

##### VMWare ESXi

**Elastic Sky X integrated**，之前叫**ESX**，是**VMWare**公司的服务器虚拟化产品。

* **Host Machine**：x86-64、x86（4.0之前）
* **Guest OS**：Linux、Windows（5.x之后）

**vCenter Server**是一个中心化的管理工具，用于管理多个虚拟机、多个ESXi主机及各模块。

**vMotion**是其中一个功能模块，用于即时迁移（live migration）。

**svMotion**是另外一个功能模块，用于虚拟磁盘的开机迁移。

##### VMWare vSphere

是**VMWare**推出的云计算虚拟化平台客户端，前身是**VMWare Infrastructure** 4，后来更名为vSphere 4，目前最新的版本是6.7。可与**vCenter Server**进行交互。

**VMWare Infrastructure**是VMWare公司的虚拟化套件（最后的版本是3.5，2009年发布）。这两者都基于VMWare **ESX/ESXi**。

##### Xen（开源）

Xen是一个开源的Hypervisor，由XenProject开发。

Xen虚拟机可以在不停止的情况下在多个物理主机之间**即时迁移（live migration）**。在操作过程中，虚拟机在没有停止工作的情况下内存被反复的复制到目标机器。虚拟机在最终目的地开始执行之前，会有一次60-300毫秒的非常短暂的暂停以执行最终的同步化，给人无缝迁移的感觉。类似的技术被用来暂停一台正在运行的虚拟机到磁盘，并切换到另外一台，第一台虚拟机在以后可以恢复。

甲骨文开发的开源的企业级Hypervisor **Oracle VM**基于Xen实现。

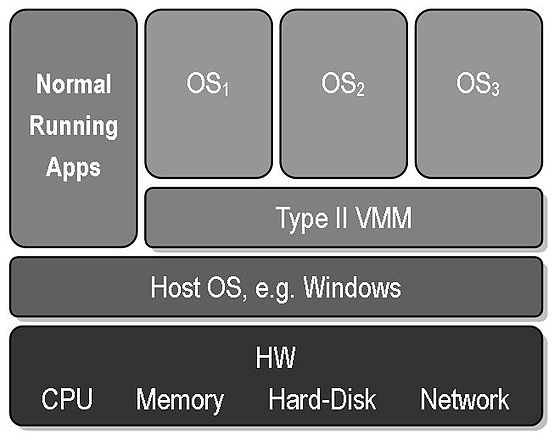
* **Host Machine**：x86、x86-64、ARM、IA-64（4.2之前）、PPC（3.2之前通过XenPPC）
* **Guest OS**：Linux、BSD、GNU/Hurd、Plan9、Windows

#### 类型II：Hosted hypervisor

这些Hypervisor与正常的应用程序一样，运行在传统的操作系统（即host OS）上。

特点：

1. 无需硬件支持
2. 运行效率较类型I低



##### Microsoft Virtual Server（废弃）

微软推出的**Hosted hypervisor**，最后一版发布于2007年，之后被**Hyper-V**替代。

* **Host Machine**：x86、x86-64
* **Host OS**：Windows XP、Windows Vista、Windows Server 2003
* **Guest Machine**：x86
* **Guest OS**：Windows XP、Windows Vista、Windows Server 2003、Linux

Virtual Server和Virtual PC的区别在于：

* Virtual PC支持多媒体，Virtual Server不支持
* Virtual PC不支持多线程，Virtual Server支持
* Virtual Server不能装在Windows 7、Windows Server 2008等较高版的Windows上
* Virtual PC的.vhd文件（虚拟机镜像）不能超过127G，而Virtual Server的限制是2048G。

##### Windows Virtual PC（废弃）

**Windows Virtual PC**是[Microsoft](https://zh.wikipedia.org/wiki/Microsoft)于[Windows 7](https://zh.wikipedia.org/wiki/Windows_7)发表后发表的[虚拟机](https://zh.wikipedia.org/wiki/虛擬機器)软件，是[**Microsoft Virtual PC**](https://zh.wikipedia.org/wiki/Microsoft_Virtual_PC)的后续软件。Windows Virtual PC主要是搭配安装[Windows XP Mode](https://zh.wikipedia.org/wiki/Windows_XP_Mode)使用。

最后一版稳定版发布于2010年，之后被**Hyper-V**替代。

* **Host Machine**：x86、x86-64
* **Host OS**：Windows早期版本（低于Windows 7）
* **Guest Machine**：x86
* **Guest OS**：Windows早期版本

##### QEMU

**QEMU（quick emulator）**是一款由Fabrice Bellard等人编写的免费的可执行[硬件虚拟化](https://zh.wikipedia.org/wiki/%E7%A1%AC%E4%BB%B6%E8%99%9A%E6%8B%9F%E5%8C%96" \o "硬件虚拟化)的（hardware virtualization）开源托管**[虚拟机](https://zh.wikipedia.org/wiki/%E8%99%9A%E6%8B%9F%E6%9C%BA" \o "虚拟机)（VMM）**。

其与[Bochs](https://zh.wikipedia.org/wiki/Bochs" \o "Bochs)，[PearPC](https://zh.wikipedia.org/wiki/PearPC" \o "PearPC)类似，但拥有高速（配合[KVM](https://zh.wikipedia.org/wiki/KVM" \o "KVM)），跨平台的特性。

QEMU是一个托管的虚拟机镜像，它通过动态的二进制转换，模拟[CPU](https://zh.wikipedia.org/wiki/CPU" \o "CPU)，并且提供一组设备模型，使它能够运行多种**未修改的Guest OS**，可以通过与**[KVM](https://zh.wikipedia.org/wiki/KVM" \o "KVM)**一起使用进而接近本地速度运行虚拟机（接近真实计算机的速度）。

QEMU还可以为user-level的进程执行CPU仿真，进而允许了为一种架构编译的程序在另外一中架构上面运行（借由VMM的形式）。

QEMU有多种模式：

* **User mod**：**用户模式**，在这种模块下，QEMU运行针对不同指令编译的单个**[Linux](https://zh.wikipedia.org/wiki/Linux" \o "Linux)**或**[Darwin](https://zh.wikipedia.org/wiki/Darwin" \o "Darwin)/**[**macOS**](https://zh.wikipedia.org/wiki/MacOS)程序。在这种模式下，我们可以实现**交叉编译（cross-compilation）**与**交叉调试（cross- debugging）**。
* **System mod**：**系统模式**，在这种模式下，QEMU模拟一个完整的计算机系统，包括外围设备。它可以用于在一台计算机上提供多台虚拟计算机的虚拟主机。 QEMU可以实现许多**Guest OS**的引导，包括**Linux、Solaris、Windows、DOS和BSD**。QEMU还可以模拟多种体系结构，比如**x86，MIPS，32-bit ARMv7，PowerPC**等等。
* **KVM Hosting**：在这种模式下，QEMU负责处理KVM镜像的设置与迁移，并参加硬件的仿真，但是客户端的执行则由KVM完成。
* **Xen Hosting**：在这种托管下，客户端的执行几乎完全在**[Xen](https://zh.wikipedia.org/wiki/Xen" \o "Xen)**中完成，并且对QEMU屏蔽。QEMU只提供硬件仿真的支持。

##### VirtualBox（Oracle）

**VirtualBox**是甲骨文公司xVM[虚拟化](https://zh.wikipedia.org/wiki/虛擬化)平台技术的一部分。

与同性质的**[VMware](https://zh.wikipedia.org/wiki/VMware_Workstation)**及**[Virtual PC](https://zh.wikipedia.org/wiki/Microsoft_Virtual_PC)**比较下，VirtualBox独到之处包括[远程桌面协议](https://zh.wikipedia.org/wiki/遠端桌面協定)（RDP）、[iSCSI](https://zh.wikipedia.org/wiki/ISCSI)及[USB](https://zh.wikipedia.org/wiki/USB)的支持，VirtualBox在客户机操作系统上已可以支持[USB 3.0](https://zh.wikipedia.org/wiki/USB_3.0)的硬件设备，不过要安装Virtualbox Extension Pack。

* **Host Machine**：x86（仅支持Windows和Linux）、x86-64
* **Host OS**：Linux、macOS、Windows、Solaris、FreeBSD
* **Guest Machine**：x86、x86-64
* **Guest OS**：[Solaris](https://zh.wikipedia.org/wiki/Solaris)、[Windows](https://zh.wikipedia.org/wiki/Microsoft_Windows)、[DOS](https://zh.wikipedia.org/wiki/DOS)、[Linux](https://zh.wikipedia.org/wiki/Linux)、[OS/2 Warp](https://zh.wikipedia.org/wiki/OS/2)、[OpenBSD](https://zh.wikipedia.org/wiki/OpenBSD)及[FreeBSD](https://zh.wikipedia.org/wiki/FreeBSD)等

##### VMWare Fusion

**VMWare Fusion**是VMWare公司专门为苹果**Macintosh**推出的一款虚拟机软件。

* **Host Machine**：Apple-Intel架构
* **Host OS**：macOS
* **Guest Machine**：同Host Machine
* **Guest OS**：Windows、Linux、Netware、Solaris

##### VMWare Workstation Player

即之前的**VMWare Player**，是VMWare Workstation的免费版，可免费用于非商业用途。

* **Host Machine**：x86、x86-64
* **Host OS**：Windows、Linux
* **Guest Machine**：x86、x86-64
* **Guest OS**：Windows、Linux

##### VMWare Workstation Pro

即之前的**VMWare Workstation**，收费的商用版本。

* **Host Machine**：x86、x86-64
* **Host OS**：Windows、Linux
* **Guest Machine**：x86、x86-64
* **Guest OS**：Windows、Linux、BSD、MS-DOS

##### VMWare Server（废弃）

即之前的**VMWare GSX Server**，一款VMWare已经不再支持的虚拟化软件，最后一个稳定版本发布于2009年。

* **Host Machine**：x86
* **Host OS**：Windows、Linux
* **Guest Machine**：x86
* **Guest OS**：Windows、Linux

##### Parallel Workstation（废弃）

一款**Parallel**发布的已经不再支持的虚拟化软件，最后一个稳定版本发布于2011年。

* **Host Machine**：x86
* **Host OS**：Windows、Linux
* **Guest Machine**：x86
* **Guest OS**：Windows、Linux

##### Parallel Desktop for Mac

一款**Parallel**发布的硬件虚拟化软件，支持Intel架构的MacOS

* **Host Machine**：Apple-intel
* **Host OS**：MacOS
* **Guest Machine**：x86
* **Guest OS**：Windows、Linux、MacOS、MacOS Server

##### Parallel Server for Mac（废弃）

一款**Parallel**发布的硬件虚拟化软件，支持Intel架构的MacOS Server，最后一个版本发布于2011年。

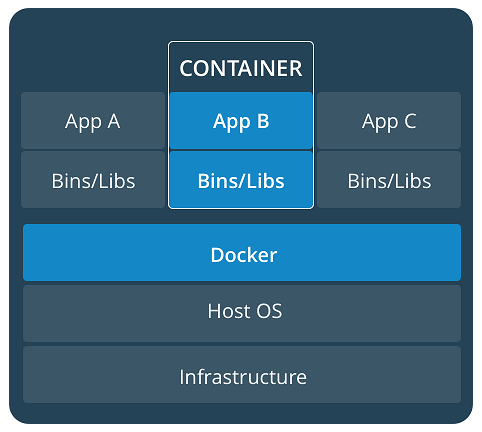
* **Host Machine**：Apple-intel
* **Host OS**：MacOS Server
* **Guest Machine**：x86
* **Guest OS**：Windows、Linux、FreeBSD

# 容器化

**容器化（Containerization），**也就是**操作系统层虚拟化（OS-level virtualization）**，是由操作系统提供的功能，可以将操作系统内核虚拟化，允许若干用户空间的**实例（instance，或叫container、jail）**运行。在程序角度看来，如同运行在在一个独立的OS之上。

在类Unix系统中，起源于**chroot**机制，具体实现包括**chroot、Docker、Linux-VServer、OpenVZ、LXC、FreeBSD jail**等。

和虚拟机技术相比，容器化的一个局限是所有的容器使用的是同一个操作系统内核（即Host OS的内核），也就是说Linux上的容器无法运行Windows的程序，或者其他版本的Linux内核。



### 内核支持

容器的实现需要**宿主操作系统（Host OS）**内核提供的功能支持。容器技术在BSD系列的操作系统中有很长的历史，通过**OpenVZ**才进入到Linux世界中。

#### chroot

**chroot（change root）**是1970年代就在类Unix操作系统上实现的一个内核技术。虽然并不能提供的完全的容器，但可说是后来所有容器技术的雏形和启发者。

对正在运作的进程和它的[子进程](https://zh.wikipedia.org/wiki/子进程)，chroot会改变它们的[根目录](https://zh.wikipedia.org/wiki/根目录)。一个经由chroot设置根目录的程序，无法访问这个指定根目录之外的文件。chroot可由**chroot(2)系统调用**或**chroot(8)程序**调用。

由chroot创造出的那个根目录，**叫做“chroot监狱”（chroot jail）**。

#### cgroups

**控制群组（Control Groups）**，是Linux内核的一个功能，用于限制、控制、分离一个**进程组（Process group）**的资源（CPU、内存、存储、输入输出）

**cgroups**的一个设计目标是为不同的应用情况提供统一的接口，从控制单一进程（像[nice](https://zh.wikipedia.org/w/index.php?title=Nice&action=edit&redlink=1)）到[操作系统层虚拟化](https://zh.wikipedia.org/wiki/作業系統層虛擬化)（[**OpenVZ**](https://zh.wikipedia.org/wiki/OpenVZ)**，[Linux-VServer](https://zh.wikipedia.org/w/index.php?title=Linux-VServer&action=edit&redlink=1)，[LXC](https://zh.wikipedia.org/wiki/LXC)**）。cgroups提供：

* **资源限制**：组可以被设置不超过设定的[内存](https://zh.wikipedia.org/wiki/内存)限制；这也包括[虚拟内存](https://zh.wikipedia.org/wiki/虚拟内存)。
* **优先级**：一些组可能会得到大量的CPU或磁盘IO吞吐量。
* **结算**：用来衡量系统确实把多少资源用到适合的目的上。
* **控制**：冻结组或检查点和重启动。

#### namespace

**namespace**是Linux内核的一个功能，将内核资源分区，使得每一组进程只能访问对应的内核资源。各namespace中同名的内核资源（比如进程ID、挂载点、hostnames、用户ID、文件名等等）会被映射为不同的资源。

有3个系统调用可以直接使用namespace：

* **clone**：通过flag来决定新clone出的进程应该属于哪个namespace
* **unshare**：使得一个进程或线程可以具有自己独立的资源
* **setns**：进入指定的namespace

#### seccomp

**secomp（secure computing mode）**是2005年出现于Linux内核中的一个技术（由Andrea Arcangeli提出）。最初的目的是用于安全的运行不被信任的程序，相当于一个**沙盒**。

seccomp使得一个进程可以转变为一个“安全状态”，在这种状态下，进程只能调用几个有限的系统调用：分别是exit()、sigreturn()、以及对已经打开的文件的read()和write()，如果它试图调用其他系统调用，则会被干掉。这个转变是单向的，无法从“安全状态”变回正常状态。也就是说，seccomp并没有将系统资源虚拟化，而仅仅是将进程完全的隔离出来。

可以通过seccomp和prctl两个系统调用来调用这个功能。

**seccomp-bpf**是seccomp的一个扩展，可以通过配置更灵活的过滤系统调用。

### 操作系统容器

**操作系统容器（OS Container）**实现了一个OS的容器，有点类似虚拟机，但仅仅是共享了Host OS的内核，隔离出一个独立的OS环境，并没有做任何硬件/虚拟机的模拟。也就是说Guest OS其实使用的就是Host OS的内核，并且只能使用Host OS**同样版本**的内核。

OS容器有时候被用来搭建**VPS（虚拟专用服务器，Virtual Private Server）**。

#### OpenVZ（Linux）

**OpenVZ（Open Virtuozzo）**是基于[Linux内核](https://zh.wikipedia.org/wiki/Linux内核)的容器技术。OpenVZ允许物理[服务器](https://zh.wikipedia.org/wiki/伺服器)运行多个操作系统，被称[**虚拟专用服务器**](https://zh.wikipedia.org/wiki/虚拟专用服务器)**（VPS，Virtual Private Server）**或**虚拟环境（VE，Virtual Environment）**。

与**[VMware](https://zh.wikipedia.org/wiki/VMware)**和**Xen**这些[虚拟机](https://zh.wikipedia.org/wiki/虚拟机)相比，OpenVZ的主机系统（Host OS）与客户系统（Guest OS）都必须是同样的[Linux](https://zh.wikipedia.org/wiki/Linux)内核（包括版本）。OpenVZ声称这样做有性能上的优势。根据OpenVZ网站的说法，使用OpenVZ与使用独立的实体服务器相比，性能只会有1-3%的损失。

OpenVZ的[Linux](https://zh.wikipedia.org/wiki/Linux)客户系统其实是共享OpenVZ主机Linux系统的内核，也就意味着OpenVZ的Linux客户系统不能升级内核。

OpenVZ是SWsoft, Inc.公司开发的专有软件**[Virtuozzo](https://zh.wikipedia.org/w/index.php?title=Virtuozzo&action=edit&redlink=1)**（Virtuozzo虚拟机还支持Windows客户系统）的基础。OpenVZ的授权为[GPLv2](https://zh.wikipedia.org/wiki/GNU通用公共許可證)。

OpenVZ由两部分组成，一个经修改过的操作系统[核心](https://zh.wikipedia.org/wiki/内核)与一套用户工具。

基于OpenVZ的Linux [VPS](https://zh.wikipedia.org/wiki/VPS)可能配置起来不如基于**[KVM](https://zh.wikipedia.org/wiki/基于内核的虚拟机)**的Linux虚拟机灵活。例如，OpenVZ VPS无法更新内核；不支持部分软件；其不能升级内核的限制也可能会给Linux的[发行版升级](https://zh.wikipedia.org/w/index.php?title=发行版升级&action=edit&redlink=1)带来麻烦。

#### LXC（Linux）

**LXC（Linux Container）**是一种OS容器，使得在单个Linux内核上可以运行多个独立的Linux系统。通过Linux内核中的**cgroups**和**namespace**功能，LXC可以在无需虚拟机/Hypervisor的情况下实现隔离的OS环境。

LXC是OpenVZ精神上的继承者，OpenVZ更多的在Red Hat世界中使用，并且使用更老的内核。

#### Linux-VServer（Linux）

Linux-VServer是一个虚拟专用服务器（Virtual Private Server，VPS）。

Linux-VServer中的一个虚拟环境被称为一个security context。

#### Lmctfy（废弃，Linux）

**lmctfy （Let Me Contain That For You）**，基于Linux内核的**cgroups**实现的OS容器化技术。由谷歌推出，在**2015**年的时候，开发者决定将其中的工作并入**Docker**，并停止lmctfy的开发。

#### FreeBSD Jail（FreeBSD）

**FreeBSD jail**在**[FreeBSD](https://zh.wikipedia.org/wiki/FreeBSD)**操作系统中运行。利用这个技术，FreeBSD的系统管理者，可以分隔出几个小型的系统，这些软件系统被称为**监狱（jails）**。

2000年，伴随FreeBSD 4.0版的发布，正式对外公开。系统管理者可以使用**[jail(8)](http://www.freebsd.org/cgi/man.cgi?query=jail&sektion=8" \l "end)**这个命令来调用这个服务。

这个技术是基于[类Unix](https://zh.wikipedia.org/wiki/类Unix)系统下的**[chroot](https://zh.wikipedia.org/wiki/Chroot)**机制，进一步发展而来。但**chroot jail**只能限制进程对文件系统的访问，而**FreeBSD Jail**中的进程，无法影响宿主OS的其他部分。也就是说，在FreeBSD Jail中的进程，是运作在一个**[沙盒](https://zh.wikipedia.org/wiki/沙盒_(電腦安全))（Sandbox）**上的。

#### Solaris Container（Solaris）

**Solaris Containers**（包括**Solaris Zones**），是一个Solaris系统上的OS容器，由[Sun Microsystem](https://zh.wikipedia.org/wiki/昇陽電腦)开发，在2005年，与Solaris 10一同完整发布。它支持[x86](https://zh.wikipedia.org/wiki/X86)与[SPARC](https://zh.wikipedia.org/wiki/SPARC)平台，在[OpenSolaris](https://zh.wikipedia.org/wiki/OpenSolaris)及其后继操作系统中也继承了这个机制。现由[甲骨文公司](https://zh.wikipedia.org/wiki/甲骨文公司)继续维护，由甲骨文公司发布的[Solaris 11](https://zh.wikipedia.org/w/index.php?title=Solaris_11&action=edit&redlink=1)中也保持了这个机制。

zone 是一个单一的操作系统实体，是一个独立虚拟服务器，完全与其他实体隔开。透过zone，**Solaris Containers**获得系统资源，并且与其他软件隔离，在沙盒中独立运作。这个技术是由unix系统中的**[chroot](https://zh.wikipedia.org/wiki/Chroot)**进一步发展而来，曾被称为“吃了[类固醇](https://zh.wikipedia.org/wiki/類固醇)的chroot”（chroot on steroids）。

#### Workload Partition（AIX）

**AIX Workload Partitions（WPARs）**是IBM开发的**AIX**上的OS容器。

#### iCore Virtual Accounts（Windows）

**iCore Virtual Accounts**是**Windows XP**上的操作系统容器。

### 应用容器

**应用容器（Application Container）**和**OS容器**的区别在于，应用容器虽然也是隔离的运行环境，但是在应用容器中仅有一个应用。

#### Docker

**Docker**由Docker.Inc公司发布，是目前最流行的应用容器，目前主要支持**Linux**（**Windows**和**macOS**也有支持），支持的体系结构包括**x86-64、ARM、S390x和PPC64le**。

Docker的实现使用了Linux内核中**cgroups**和**namespace**功能，以及**UnionFS**作为其存储结构，**UnionFS**是一种文件系统服务，不特指某种文件系统，利用**union mount**其它文件系统来实现**分层存储**，比如**OverlayFS**。

版本0.9之前的Docker使用**LXC**命令行工具作为缺省的执行环境，之后被**libcontainer**库替代。LXC成为Docker的一个可选的运行环境，从1.10开始Docker停止了对LXC的支持。

#### rkt

**rkt**（原先叫**rocket**）是**CoreOS**公司发布的一个应用容器，**rkt**跟随**Container** **Linux**（**CoreOS** **Linux**）发布，作为**Docker**的备选。

### 容器操作系统

**容器操作系统（Container OS ）**，专门用于运行容器的操作系统，其上的应用程序都是以容器的形式运行，这样的组合是为了提高开发/部署的效率。

1. 请勿和**操作系统容器（OS Container）**搞混， **OS Container**是一个容器，里面装的是OS（其实只是个隔离的OS环境），而**Container OS**是一个专门运行容器的OS。

2. 此外**容器化（Containerization）**也叫**操作系统级虚拟化（OS-level Virtualization）**。

#### Container Linux

**Container Linux**之前叫**CoreOS Linux**，是一个开源的轻量化的OS，基于Linux内核定制。这是一个**容器操作系统**（专门运行容器的OS），只提供能支持容器运行的最少的功能，没有包管理机制。

Container Linux由**CoreOS**公司开发，侧重于大规模部署，面向企业，专注于自动化，轻松部署，安全，可靠，规模化。

Container Linux同时支持**rkt**（CoreOS公司开发）和**Docker**两种容器。

#### RancherOS

**Rancher Labs**公司开发的基于Linux内核的开源容器操作系统，**RancherOS**仅提供支持容器运行的最少的内核/应用组件。

Rancher Labs公司发布了两个产品，一个是**Rancher OS**，另一个是基于**Kubernetes**的容器管理工具**Rancher**。

### Orchestration

**Orchestration**在计算机科学中指的是复杂系统中的自动化的配置管理。

#### ECS

#### Kubernetes

**Kubernetes**（常简称为**K8s**）是用于自动部署、扩展和管理[容器化](https://zh.wikipedia.org/wiki/%E4%BD%9C%E6%A5%AD%E7%B3%BB%E7%B5%B1%E5%B1%A4%E8%99%9B%E6%93%AC%E5%8C%96" \o "操作系统层虚拟化)（containerized）应用程序的[开源](https://zh.wikipedia.org/wiki/%E5%BC%80%E6%BA%90%E8%BD%AF%E4%BB%B6" \o "开源软件)系统。该系统由[Google](https://zh.wikipedia.org/wiki/Google)设计并捐赠给Cloud Native Computing Foundation（CNCF，今属[Linux基金会](https://zh.wikipedia.org/wiki/Linux%E5%9F%BA%E9%87%91%E4%BC%9A)）来使用。

它旨在提供“跨主机集群的自动部署、扩展以及运行应用程序容器的平台”。 它支持一系列容器工具, 包括**[Docker](https://zh.wikipedia.org/wiki/Docker_(%E8%BB%9F%E9%AB%94)" \o "Docker (软件))**等。CNCF于2017年宣布首批Kubernetes认证服务提供商（KCSPs），包含[IBM](https://zh.wikipedia.org/wiki/IBM" \o "IBM)、[华为](https://zh.wikipedia.org/wiki/%E8%8F%AF%E7%82%BA" \o "华为)、MIRANTIS、inwinSTACK迎栈科技等服务商。

#### OpenShift

OpenShift是红帽公司开发的一系列的容器化工具。其中的旗舰产品是OpenShift Container Platform。

### 虚拟机和应用容器的区别

**虚拟机的优点**：

1、系统的完整抽象：由于您的应用程序的所有部分都运行在相同的“服务”或服务器上，它们之间的可以直接通信，不需要额外的复杂网络。

2、不需要分解应用程序：因为运行在类似于裸机的环境中，所以不需要改变应用程序本身的架构。

3、同时运行多个应用程序：通常在单个虚拟机上运行多个应用程序，从而简化整体基础架构的管理。

4、安全：虚拟机具有长期的使用记录，被认为是相当安全的，提供了相当小的攻击面的隔离。

5、可用的各种操作系统：在虚拟机中，您可以使用几乎任何操作系统，因此您可以在单个物理服务器上运行多个不同操作系统。

**虚拟机的缺点：**

1、占用资源太大：由于包含太多内容，VM可以很大，无论是在定义它们所需的镜像，还是在运行镜像所需的资源。

2、启动速度可能很慢：启动虚拟机与启动计算机相同;可能需要一些时间。如果你只是开始一次，让它运行几个星期或几个月或几年，这可能不是一个问题。但是，如果您正在处理必须不断的迁移过程，这种延迟肯定会是一个问题。

3、可能运行缓慢：由于它本质上是模拟的计算机，运行在VM上的应用程序通常不如在裸机上运行的那样高效。

4、不能或者不容易嵌套：虽然在某些情况下可以在VM中运行另一个VM，但并不总是这样。更重要的是，性能损失可能很大。

5、需要仔细的安全配置：托管虚拟机的平台需要仔细分析和配置，以防止由于安全域桥接引起的潜在安全问题，例如跨越多个安全域（如公共和管理）的管理组件以及相应数据。

**容器的优点**

1、相对较小：容器共享主机的内核，它们只包含绝对必要的操作系统和库组件，它们（通常应该）将自己限于单个功能，因此它们往往非常小。

2、快：因为它们很小，它们可以在几秒甚至更少的时间内启动，使其对于需要反复启动和关闭的应用程序（例如所谓的“无服务器”应用程序）很有用。

3、CI / CD：容器经常启动或重启，所以很容易迭代。

4、便携：因为它们是独立的，只要正确的内核就位，容器可以相对容易地在主机之间迁移。

5、生命周期和交付模式：容器的生命周期更易于整合诸如漏洞评估和镜像注册、签名等高级功能。

**容器的缺点**

1、可能需要复杂的网络：由于将所有功能（理想情况下）分解成多个容器，因此这些容器需要相互通信才能完成任务。但是由于容器不是单一的单元，它们必须相互通信。一些编排系统（如Kubernetes）具有更高级别的单位，如多容器pod，使其更容易一些，但仍然比使用VM更复杂。

2、可能不那么安全：正如我上面提到的，容器仍然是新技术，由于许多的原因，它们仍然被认为不如VM那么安全。

3、可能需要更多的工作：如果您正在使用容器，您必须将您的应用程序分解为各种服务再构成一个整体，如果您使用虚拟机，则不需要这些工作。

4、可能是不可靠的：虽然这听起来是负面的，但容器通常设计用于云本地计算，它假定任何组件可以随时死亡;您需要确保应用程序适应这种架构。

总结：

# 桌面虚拟化

**Desktop Virtualization**，或者叫**虚拟桌面**，是基于服务器[虚拟化](https://zh.wikipedia.org/wiki/%E8%99%9A%E6%8B%9F%E5%8C%96)诞生出的一种技术，其将所有桌面PC所需的[操作系统软件](https://zh.wikipedia.org/wiki/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F" \o "操作系统)、应用程序软件、用户数据全部存放到后台服务器中，通过专门的管理系统赋予给特定用户，用户通过专用的网络传输协议连接到后端服务器分配的桌面资源，连接后，用户可在连接本地终端上直接使用后台运行的桌面系统，使用体验基本与物理PC一致。

在使用虚拟桌面时，PC操作系统在后台服务器端运行，本地终端仅用于连接显示作用；用户可使用任意终端在任意地点、任意可连接的网络环境下，使用虚拟桌面。

## XenDesktop（Citrix）

**Citrix**公司发布的Windows虚拟桌面软件。用户可以通过**XenDesktop**远程的访问和操作微软的Windows桌面，同时使用**XenApp**来远程的访问/操作Windows的应用程序，**Citrix** **Receiver**是用于连接XenDesktop和XenApp的客户端。

## 远程桌面服务（Microsoft）

**Remote Desktop Serivce**，**RDS**，是微软从Windows 2000 Server开始提供的组件，使得所有支持远程桌面协议（Remote Desktop Protocol）的瘦客户端可以连接支持**RDS**的机器。

目前有3个客户端程序使用RDS：

1. Quick Assistant，原先叫远程支持（Windows Remote Assistance）。
2. Remote Desktop Connection（RDC），远程桌面连接
3. 快速用户切换（Fast User Switching）

## VMWare Horizon View

VMWare公司开发的虚拟桌面，之前叫**VMWare** **VDM**，后改名**VMWare** **View**，最后改成**VMWare Horizon View**。

VMWare Horizon View支持**Windows**、**Linux**和**MacOS**。

# 应用虚拟化

**Application Virtualization**，也称**程序虚拟机（Process virtual machine）**，被定义为**将应用程序打包封装的技术**，使得一个被虚拟化的应用不运行在传统意义上的环境中，尽管用户仿佛仍然在传统的操作系统上使用它。

在这种语境下，和**硬件虚拟化**中的意义不同，“虚拟化”这个词指的是**封装应用**的技术。应用虚拟化中有一个**应用虚拟层**，应用虚拟层替代了部分通常由OS提供的运行环境，拦截了虚拟应用所有的磁盘操作，并将之重定向到另一个位置。

虚拟机的一个定义在**host OS（宿主操作系统）**上作为一个**普通的**进程运行，向外提供平台（硬件/操作系统）无关的虚拟平台，最典型的便是**JVM（Java虚拟机）**。

应用虚拟化的另外一个定义是远程桌面。

## Java虚拟机

**Java虚拟机**（**Java Virtual Machine，JVM**），一种能够运行**[Java bytecode](https://zh.wikipedia.org/wiki/Java_bytecode" \o "Java bytecode)（字节码）**的[虚拟机](https://zh.wikipedia.org/wiki/%E8%99%9B%E6%93%AC%E6%A9%9F%E5%99%A8)，是一种[堆栈机器](https://zh.wikipedia.org/wiki/%E5%A0%86%E7%96%8A%E7%B5%90%E6%A7%8B%E6%A9%9F%E5%99%A8)。最早由[**太阳微系统**](https://zh.wikipedia.org/wiki/%E6%98%87%E9%99%BD%E9%9B%BB%E8%85%A6)**（Sun Microsystem）**所研发并实现第一个实现版本，是[Java平台](https://zh.wikipedia.org/wiki/Java%E5%B9%B3%E8%87%BA" \o "Java平台)的一部分，能够运行以[Java](https://zh.wikipedia.org/wiki/Java)语言写作的[软件](https://zh.wikipedia.org/wiki/%E8%BB%9F%E9%AB%94" \o "软件)[程序](https://zh.wikipedia.org/wiki/%E7%A8%8B%E5%BC%8F)。

JVM有自己完善的[硬体](https://zh.wikipedia.org/wiki/%E7%A1%AC%E4%BD%93" \o "硬体)架构，如[处理器](https://zh.wikipedia.org/wiki/%E5%A4%84%E7%90%86%E5%99%A8)、[堆栈](https://zh.wikipedia.org/wiki/%E5%A0%86%E6%A0%88)、[寄存器](https://zh.wikipedia.org/wiki/%E5%AF%84%E5%AD%98%E5%99%A8)等，还具有相应的[指令](https://zh.wikipedia.org/wiki/%E6%8C%87%E4%BB%A4)系统。JVM屏蔽了与具体[操作系统](https://zh.wikipedia.org/wiki/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F" \o "操作系统)平台相关的信息，使得Java[程序](https://zh.wikipedia.org/wiki/%E7%A8%8B%E5%BA%8F)只需生成在Java虚拟机上运行的目标代码（[字节码](https://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82%E7%A0%81" \o "字节码)），就可以在多种平台上不加修改地运行。通过对中央处理器（[CPU](https://zh.wikipedia.org/wiki/CPU" \o "CPU)）所执行的软件实现，实现能执行[编译](https://zh.wikipedia.org/wiki/%E7%BC%96%E8%AF%91" \o "编译)过的Java程序码（[Applet](https://zh.wikipedia.org/wiki/Applet" \o "Applet)与应用程序）。

作为一种编程语言的虚拟机，实际上不只是专用于Java语言，只要生成的编译文件匹配JVM对加载编译文件格式要求，任何语言都可以由JVM编译运行。目前原生就可以运行在JVM的语言包括：**Clojure、Groovy、Java、Kotlin、Scala**等。此外某些语言也实现了JVM编译器：**Python（Jython）、Ruby（JRuby）、JavaScript（Rhino）、Erlang（Erjang**）等。

此外，除了[甲骨文](https://zh.wikipedia.org/wiki/%E7%94%B2%E9%AA%A8%E6%96%87%E5%85%AC%E5%8F%B8" \o "甲骨文公司)，也有其他开源或闭源的实现。

## Dalvik

**Dalvik虚拟机**，是[**Google**](https://zh.wikipedia.org/wiki/Google)等厂商合作开发的**[Android](https://zh.wikipedia.org/wiki/Android" \o "Android)**移动设备平台的核心组成部分之一。

它可以支持已转换为**.dex**（即“Dalvik Executable”）格式的[Java](https://zh.wikipedia.org/wiki/Java" \o "Java)应用程序的运行。.dex格式是专为Dalvik设计的一种压缩格式，适合[内存](https://zh.wikipedia.org/wiki/%E5%86%85%E5%AD%98" \o "内存)和[处理器](https://zh.wikipedia.org/wiki/%E5%A4%84%E7%90%86%E5%99%A8)速度有限的系统。

大多数[虚拟机](https://zh.wikipedia.org/wiki/%E8%99%9A%E6%8B%9F%E6%9C%BA" \o "虚拟机)包括[**JVM**](https://zh.wikipedia.org/wiki/JVM)都是一种**[堆栈机器](https://zh.wikipedia.org/wiki/%E5%A0%86%E7%96%8A%E6%A9%9F%E5%99%A8" \o "堆栈机器)**，而Dalvik虚拟机则是**[寄存器机](https://zh.wikipedia.org/wiki/%E5%AF%84%E5%AD%98%E5%99%A8%E6%9C%BA" \o "寄存器机)**。两种架构各有优劣，一般而言，基于堆栈的机器需要更多指令，而基于寄存器的机器指令更长。

从[Android 5.0](https://zh.wikipedia.org/wiki/Android_5.0" \o "Android 5.0)版起，**[Android Runtime](https://zh.wikipedia.org/wiki/Android_Runtime" \o "Android Runtime)（ART）**取代Dalvik成为系统内默认虚拟机。

**dx**是一套工具，可以将Java的**.class**转换成**.dex**格式。一个dex档通常会有多个.class。由于dex有时必须进行优化，会使文件大小增加1-4倍，以**ODEX**结尾。

## XenApp（Citrix）

**XenApp**是**Citrix**开发的一个应用虚拟化软件，XenApp允许用户从另外一台机器访问某个Windows应用。XenApp之前叫做**WinFrame**、**MetaFrame**和**Presentation Server**。

XenApp使用Citrix开发的专有平台**FlexCast Management Architecture（FMA）**，将单个应用而非整个桌面传送到其他设备。有时候**XenApp**被用来和**XenDesktop**一起使用以便将整个桌面远程化。

用户使用**Citrix Receiver**作为客户端连接**XenApp**。

## VMWare ThinApp

**VMWare ThinApp**（之前叫**Thinstall**）是VMWare公司推出的应用虚拟化以及**便携应用**（**Portable application，Portable Software，**即**绿色软件**，无需安装直接运行的软件）创建的套件。传统的应用程序可以通过**ThinApp**被打包成便携应用，打包成功的概率在90%到95%左右。

# 云平台

云平台用于管理系统资源，虚拟机是云平台的基础，但云平台不仅限于对虚拟机的管理，还有大数据、分布式数据库、负载均衡甚至机器学习等等。

以下是各云平台支持的虚拟机

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Xen** | **KVM** | **XenServer / XCP** | **VMWare** | **LXC** | **openVZ** |
| **Eucalyptus** | Y | Y |  | Y |  |  |
| **OpenStack** | Y | Y | Y | Y | Y |  |
| **OpenNebula** | Y | Y |  | Y |  |  |
| **OpenQRM** | Y | Y | Y | Y | Y | Y |
| **XenServer** | Y |  |  |  |  |  |
| **Oracle VM** | Y |  |  |  |  |  |
| **CloudStack** |  | Y | Y | Y |  |  |
| **ConVirt** | Y | Y |  |  |  |  |