1.

虚拟化的好处：

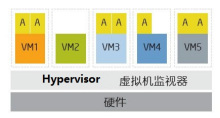
如果不使用虚拟化技术，那么物理服务器的硬件资源一般只使用了，20-30%的功率。并且占用实际物理面积，在机房难以管理，在执行例如冗余操作的时候，需要多台物理服务器十分麻烦。

使用了虚拟化技术以后，我们可以创造出数倍，数十倍于物理资源的服务器，使的物理硬件资源使用率达到80%以上。减少了维护的成本，并且可以提供中心一体化的管理等。

2.

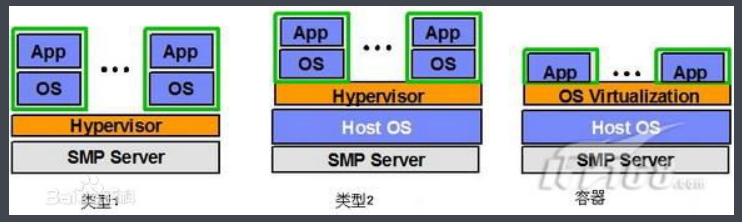
Hypervisor: Hypervisor，又称虚拟机监视器（英语：virtual machine monitor，缩写为 VMM），是用来建立与执行[虚拟机器](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA%E5%99%A8)的软件、固件或硬件.

Hypervisor——一种运行在基础物理服务器和[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F" \t "_blank)之间的[中间软件](https://baike.baidu.com/item/%E4%B8%AD%E9%97%B4%E8%BD%AF%E4%BB%B6)层，可允许多个操作系统和应用共享硬件。也可叫做VMM（ virtual machine monitor ），即[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA" \t "_blank)[监视器](https://baike.baidu.com/item/%E7%9B%91%E8%A7%86%E5%99%A8)。

[](https://baike.baidu.com/pic/hypervisor/3353492/0/99636c0e35df258c7bcbe1d0?fr=lemma&ct=single)

Hypervisors是一种在[虚拟环境](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E7%8E%AF%E5%A2%83" \t "_blank)中的“元”[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)。他们可以访问服务器上包括磁盘和内存在内的所有[物理设备](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E8%AE%BE%E5%A4%87)。Hypervisors不但协调着这些硬件资源的访问，也同时在各个[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA" \t "_blank)之间施加防护。当服务器启动并执行Hypervisor时，它会加载所有虚拟机[客户端](https://baike.baidu.com/item/%E5%AE%A2%E6%88%B7%E7%AB%AF" \t "_blank)的[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)同时会分配给每一台虚拟机适量的[内存](https://baike.baidu.com/item/%E5%86%85%E5%AD%98)，[CPU](https://baike.baidu.com/item/CPU)，[网络](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C" \t "_blank)和[磁盘](https://baike.baidu.com/item/%E7%A3%81%E7%9B%98)。

3.Hypervisor类型：



I型：

虚拟机直接运行在系统硬件上，创建硬件全仿真实例，被称为“裸机”型。

裸机型在虚拟化中Hypervisor直接管理调用硬件资源，不需要底层操作系统，也可以将Hypervisor看

作一个很薄的操作系统。这种方案的性能处于主机虚拟化与操作系统虚拟化之间。

II型：

虚拟机运行在传统操作系统上（例如windows, Linux），同样创建的是硬件全仿真实例，被称为“托管（宿主）”型。

托管型/主机型Hypervisor运行在基础操作系统上，构建出一整套虚拟硬件平台

（CPU/Memory/Storage/Adapter），使用者根据需要安装新的操作系统和应用软件，底层和上层的

操作系统可以完全无关化

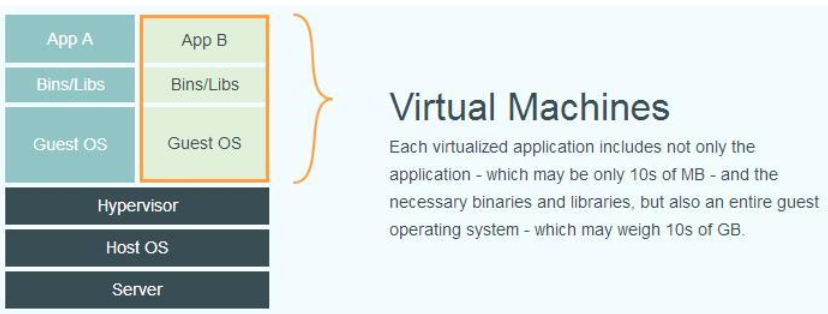
III型：

[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA)运行在传统[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)上，创建一个独立的虚拟化实例（容器），指向底层托管操作系统，被称为“操作系统虚拟化”。

4.

主机虚拟化（VM）和容器（Container）虚拟化

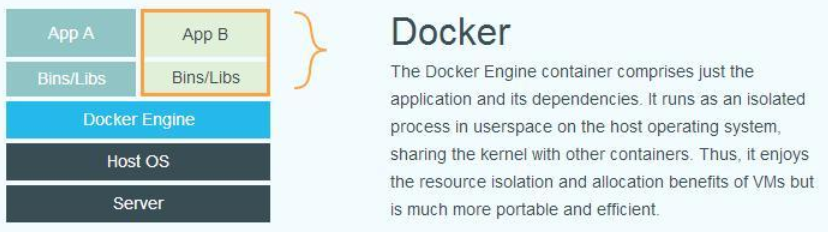
主机级虚拟化(VM)



主机级别的虚拟化是将物理资源转变为逻辑上可以管理的资源，以打破物理结构之间的壁垒，使计算元件运行在虚拟的基础上，而不是真实的物理资源上。通过虚拟化技术，可以将物理资源转变为逻辑资源(虚拟机)，应用程序服务运行在虚拟资源上，而不是真实的物理机上。因此，对于主机级虚拟化，我们的实现方式一般是想办法去模拟出硬件环境，模拟出虚拟的cpu、内存、硬盘、网卡等资源，然后在这些虚拟资源之上安装合适的操作系统来控制这些资源.

主机虚拟化中有独自的Guest OS. VM从VM image中产生。

容器(Container)虚拟化



容器是一种虚拟化的方案，和传统的虚拟机(通过中间层”guerst OS”运行服务)不同，Docker直接运行在操作系统之上。容器虚拟化中各容器分享同一个OS,每个容器中不含有OS，并且我们可以在OS层中看到，每个container是一个process进程。

一句话来概括的话，主机级虚拟化就是通过各种各样的手段，把物理资源重新分配，然后抽象出一部分拿来做虚拟机的虚拟硬件，是对硬件的模拟；而容器（Container）虚拟化技术相当于把操作系统进行虚拟化，把物理的操作系统模拟为逻辑上的多个操作系统，不同的操作系统有自己的用户空间，实现了应用程序间的隔离。

Container由Container image产生。产生出来的container可以进行迁移，但是需要注意迁移以后container的底层hostOS能提供所需要的内核。

5.

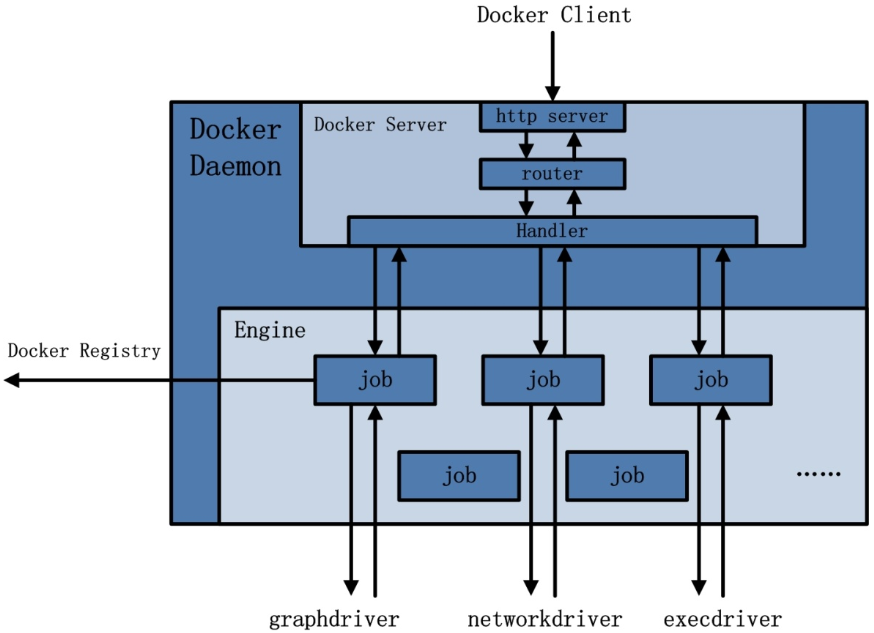
Docker2013年开始的开源软件。

**Docker Host**: 运行Docker Daemon的平台，可以是一台物理服务器也可以是一台VM.

**Docker registry**: Docker image的仓库。存放Docker的各种images. 通常，一个仓库会包含同一个软件不同版本的镜像，而标签就常用于对应该软件的各个版本 。我们可以通过<仓库名>:<标签>的格式来指定具体是这个软件哪个版本的镜像。如果不给出标签，将以 latest 作为默认标签。

**Docker Daemon**: Docker daemon是服务器组件，以 Linux 后台服务的方式运行，是 Docker 最核心的后台进程，我们也把它称为守护进程。它负责响应来自 Docker Client 的请求，然后将这些请求翻译成系统调用完成容器管理操作。该进程会在后台启动一个 API Server ，负责接收由 Docker Client 发送的请求，接收到的请求将通过Docker daemon 内部的一个路由分发调度，由具体的函数来执行请求。

Docker Daemon的架构如下所示：



Docker Daemon 可以认为是通过 Docker Server 模块接受 Docker Client 的请求，并在 Engine 中处理请求，然后根据请求类型，创建出指定的 Job 并运行。 Docker Daemon 运行在 Docker host 上，负责创建、运行、监控容器，构建、存储镜像。

**Docker Client**: 也称 Docker 客户端。它其实就是 Docker 提供命令行界面 (CLI) 工具，是许多 Docker 用户与 Docker 进行交互的主要方式。客户端可以构建，运行和停止应用程序，还可以远程与Docker\_Host进行交互。最常用的 Docker 客户端就是 docker 命令，我们可以通过 docker 命令很方便地在 host 上构建和运行 docker 容器。

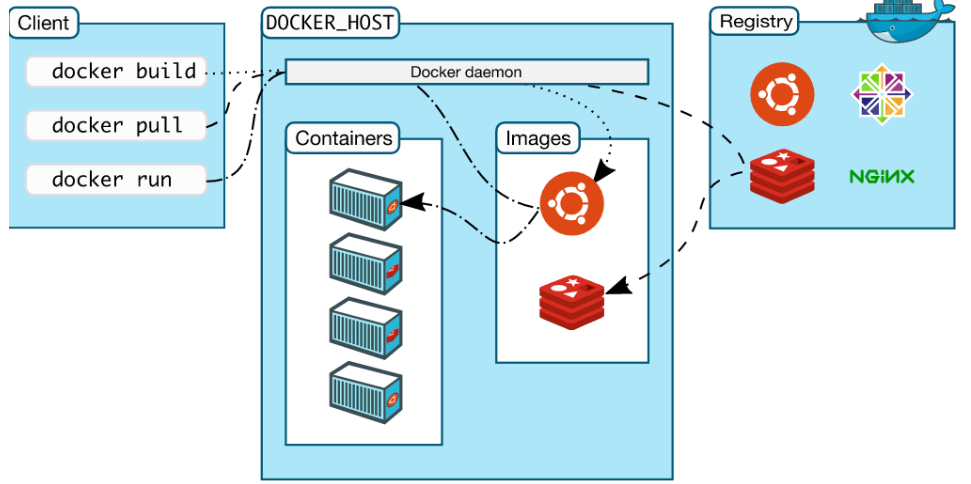
**Docker Image:** Docker 镜像可以看作是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。我们可将 Docker 镜像看成只读模板，通过它可以创建 Docker 容器。

镜像有多种生成方法

1. 从无到有开始创建镜像
2. 下载并使用别人创建好的现成的镜像
3. 在现有镜像上创建新的镜像

**Docker Container:** Docker 容器就是 Docker 镜像的运行实例，是真正运行项目程序、消耗系统资源、提供服务的地方。 Docker Container 提供了系统硬件环境，我们可以使用 Docker Images 这些制作好的系统盘，再加上我们所编写好的项目代码， run 一下就可以提供服务啦。

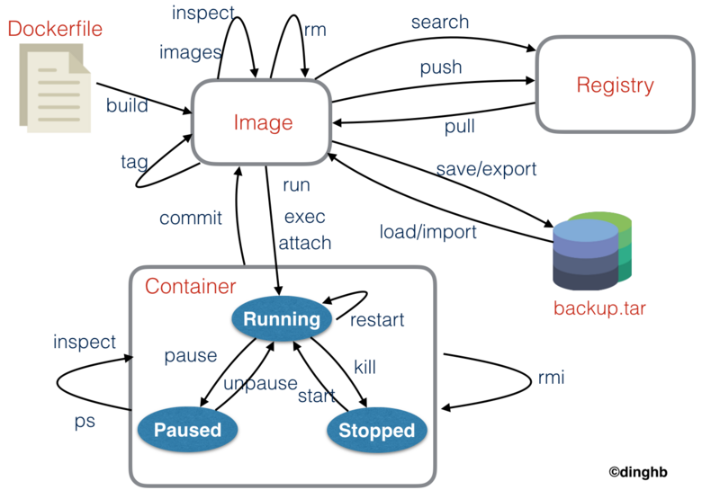
6.



如图：Docker 使用 C/S 结构，即客户端/服务器体系结构。 Docker 客户端与 Docker 服务器进行交互，Docker服务端负责构建、运行和分发 Docker 镜像。 Docker 客户端和服务端可以运行在一台机器上，也可以通过 RESTful 、 stock 或网络接口与远程 Docker 服务端进行通信。

7.

Docker常用的命令图：



docker -h: 显示docker命令的帮助信息

docker pull [image\_name]: 获取一个镜像

docker images: 显示本机存在的镜像

docker ps -a: 显示所有容器，包括未运行状态的容器

docker start [container\_name/container\_id]: 启动一个容器

docker restart [container\_name/container\_id]: 重新启动一个容器

docker stop [container\_name/container\_id]: 停止一个容器

docker rm [container\_name/container\_id]: 删除一个容器

docker rmi [image\_name]: 删除一个镜像

8.

Dockerfile定义镜像内容，是自动构建 docker 镜像的配置文件。Dockerfile 是由一行行命令语句组成，并且支持已 # 开头的注释行。

一般来说，我们可以将 Dockerfile 分为四个部分：

基础镜像(父镜像)信息指令 FROM

维护者信息指令 MAINTAINER

镜像操作指令 RUN 、 EVN 、 ADD 和 WORKDIR 等

容器启动指令 CMD 、 ENTRYPOINT 和 USER 等

Dockerfile 中所有的命令都是以下格式：INSTRUCTION argument ，指令 (INSTRUCTION) 不分大小写，但是推荐大写。

FROM命令

FROM 是用于指定基础的 images ，一般格式为 FROM <image> or FORM <image>:<tag> ，所有的 Dockerfile 都用该以 FROM 开头，FROM 命令指明 Dockerfile 所创建的镜像文件以什么镜像为基础，FROM 以后的所有指令都会在 FROM 的基础上进行创建镜像。可以在同一个 Dockerfile 中多次使用 FROM 命令用于创建多个镜像。比如我们要指定 python 2.7 的基础镜像，我们可以像如下写法一样：

FROM python:2.7

MAINTAINER命令

MAINTAINER 是用于指定镜像创建者和联系方式，一般格式为 MAINTAINER <name> 。这里我设置成我的 ID 和邮箱：

MAINTAINER Test <[Test8@gmail.com](mailto:Test8@gmail.com)>

COPY命令

COPY 是用于复制本地主机的 <src> (为 Dockerfile 所在目录的相对路径) 到容器中的 <dest>。

当使用本地目录为源目录时，推荐使用 COPY 。一般格式为 COPY <src><dest> 。例如我们要拷贝当前目录到容器中的 /app 目录下，我们可以这样操作：

COPY . /app

WORKDIR

WORKDIR 用于配合 RUN，CMD，ENTRYPOINT 命令设置当前工作路径。可以设置多次，如果是相对路径，则相对前一个 WORKDIR 命令。默认路径为/。一般格式为 WORKDIR /path/to/work/dir 。例如我们设置/app 路径，我们可以进行如下操作：

WORKDIR /app

RUN

RUN 用于容器内部执行命令。每个 RUN 命令相当于在原有的镜像基础上添加了一个改动层，原有的镜像不会有变化。一般格式为 RUN <command> 。例如我们要安装 python 依赖包，我们做法如下：

RUN pip install -r requirements.txt

EXPOSE

EXPOSE 命令用来指定对外开放的端口。一般格式为 EXPOSE <port> [<port>...]

开放5000端口：

EXPOSE 5000

ENTRYPOINT

ENTRYPOINT 可以让你的容器表现得像一个可执行程序一样。一个 Dockerfile 中只能有一个 ENTRYPOINT，如果有多个，则最后一个生效。

ENTRYPOINT 命令也有两种格式：

ENTRYPOINT ["executable", "param1", "param2"] ：推荐使用的 exec形式

ENTRYPOINT command param1 param2 ：shell 形式

例如下面这个，我们要将 python 镜像变成可执行的程序，我们可以这样去做：

ENTRYPOINT ["python"]

CMD

CMD 命令用于启动容器时默认执行的命令，CMD 命令可以包含可执行文件，也可以不包含可执行文件。不包含可执行文件的情况下就要用 ENTRYPOINT 指定一个，然后 CMD 命令的参数就会作为ENTRYPOINT的参数。

CMD 命令有三种格式：

CMD ["executable","param1","param2"]：推荐使用的 exec 形式。

CMD ["param1","param2"]：无可执行程序形式

CMD command param1 param2：shell 形式。

一个 Dockerfile 中只能有一个CMD，如果有多个，则最后一个生效。而 CMD 的 shell 形式默认调用 /bin/sh -c 执行命令。

CMD 命令会被 Docker 命令行传入的参数覆盖：docker run busybox /bin/echo Hello Docker 会把 CMD 里的命令覆盖。

例如我们要启动 /app ，我们可以用如下命令实现：

CMD ["app.py"]

9.