容器技术是近几年计算机领域的热门技术，特别是随着各种云服务的发展，越来越多的服务运行在以 Docker 为代表的容器之内。

这一课时我们就来分享一下容器化技术相关的知识。

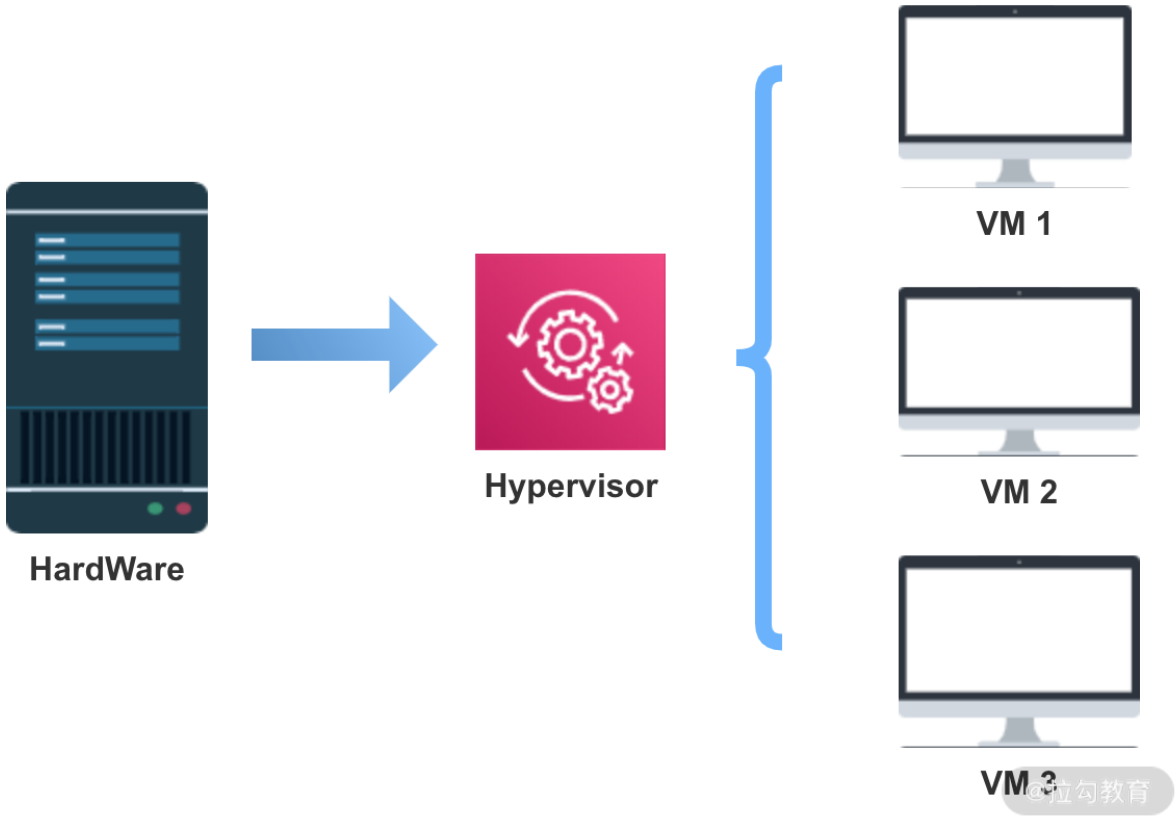
容器化技术简介

相比传统虚拟化技术，容器技术是一种更加轻量级的操作系统隔离方案，可以将应用程序及其运行依赖环境打包到镜像中，通过容器引擎进行调度，并且提供进程隔离和资源限制的运行环境。

虚拟化技术

虚拟化技术通过 Hypervisor 实现虚拟机与底层硬件的解耦，虚拟机实现依赖 Hypervisor 层，Hypervisor 是整个虚拟机的核心所在。

Hypervisor 是什么呢？也可以叫作虚拟机监视器 VMM（Virtual Machine Monitor），是一种运行在基础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硬件。



Hypervisor 虚拟机可以模拟机器硬件资源，协调虚拟机对硬件资源的访问，同时在各个虚拟机之间进行隔离。

每一个虚拟机都包括执行的应用，依赖的二进制和库资源，以及一个完整的 OS 操作系统，虚拟机运行以后，预分配给它的资源将全部被占用。

容器化技术

在容器技术中，最具代表性且应用最广泛的是 Docker 技术。

Docker 是一个开源的应用容器引擎，可以打包应用以及依赖包到一个可移植的容器中，然后发布到服务器上，Docker 容器基于镜像运行，可部署在物理机或虚拟机上，通过容器引擎与容器编排调度平台实现容器化应用的生命周期管理。

使用容器化技术有哪些好处呢？

Docker 不同于 VM，只包含应用程序及依赖库，处于一个隔离的环境中，这使得 Docker 更加轻量高效，启动容器只需几秒钟之内完成。由于 Docker 轻量、资源占用少，可以更方便地部署标准化应用，一台主机上可以同时运行上千个 Docker 容器。

两种虚拟化技术的对比

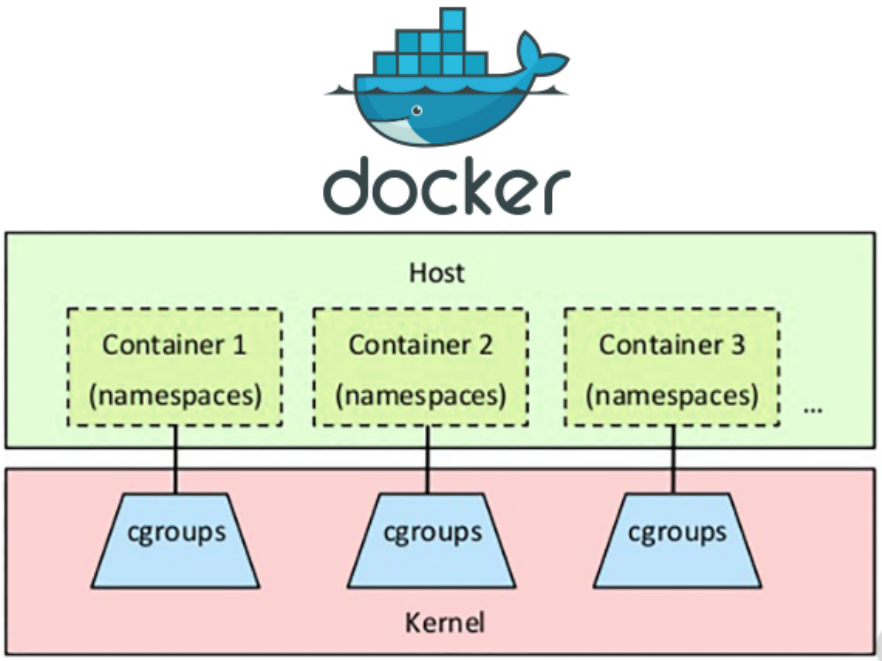
虚拟机是一个运行在宿主机之上的完整操作系统，虚拟机运行自身操作系统会占用较多的 CPU、内存、硬盘资源等。

虚拟化技术为用户提供了一个完整的虚拟机，包括操作系统在内，容器化技术为应用程序提供了隔离的运行空间，容器之间共享同一个上层操作系统内核。虚拟化技术有更佳的隔离性和安全性，但是更新和升级困难，容器化具有快速扩展、灵活性和易用性等优势，但其隔离性较差、安全性相对较低。

实际部署一般是把两种技术结合起来，比如一个虚拟机中运行多个容器，这样既保证了较好的强隔离性和安全性，也有了快速扩展、灵活性和易用性。

容器化的原理

容器技术的核心是如何实现容器内资源的限制，以及不同容器之间的隔离，这些是基于 Linux 的 Namespace 和 CGroups 技术。



Namespace

> Namespace 的目的是通过抽象方法使得 Namespace 中的进程看起来拥有它们自己的隔离的全局系统资源实例。
>
> Linux 内核实现了六种 Namespace：Mount namespaces、UTS namespaces、IPC namespaces、PID namespaces、Network namespaces、User namespaces，功能分别为：隔离文件系统、定义 hostname 和 domainame、特定的进程间通信资源、独立进程 ID 结构、独立网络设备、用户和组 ID 空间。

Docker 在创建一个容器的时候，会创建以上六种 Namespace 实例，然后将隔离的系统资源放入到相应的 Namespace 中，使得每个容器只能看到自己独立的系统资源。

Cgroups

Docker 利用 CGroups 进行资源隔离。CGroups（Control Groups）也是 Linux 内核中提供的一种机制，它的功能主要是限制、记录、隔离进程所使用的物理资源，比如 CPU、Mermory、IO、Network 等。

简单来说，CGroups 在接收到调用时，会给指定的进程挂上钩子，这个钩子会在资源被使用的时候触发，触发时会根据资源的类别，比如 CPU、Mermory、IO 等，然后使用对应的方法进行限制。

CGroups 中有一个术语叫作 Subsystem 子系统，也就是一个资源调度控制器，CPU Subsystem 负责 CPU 的时间分配，Mermory Subsystem 负责 Mermory 的使用量等。Docker 启动一个容器后，会在 /sys/fs/cgroup 目录下生成带有此容器 ID 的文件夹，里面就是调用 CGroups 的配置文件，从而实现通过 CGroups 限制容器的资源使用率。

微服务如何适配容器化

微服务的设计思想是对系统功能进行解耦，拆分为单独的服务，可以独立运行，而容器进一步对这种解耦性进行了扩展，应用容器技术可以对服务进行快速水平扩展，从而到达弹性部署业务的能力。在各种云服务概念兴起之后，微服务结合 Docker 部署，更加方便微服务架构运维部署落地。

微服务结合容器有很多优点，但是另一方面，也给服务的部署和应用提出了一些新的问题。

以 Java 服务为例，容器与虚拟机不同，其资源限制通过 CGroup 来实现，而容器内部进程如果不感知 CGroup 的限制，就进行内存、CPU 分配的话，则可能会导致资源冲突的问题。

Java 8 之前的版本无法跟 Docker 很好的配合，JVM 通过容器获取的可用内存和 CPU 数量并不是 Docker 允许使用的可用内存和 CPU 数量。

我们在开发中会应用一些线程池，通常会根据 CPU 核心数来配置，比如使用：

```
Runtime.getRuntime().availableProcessors()
```

在 1.8 版本更早的实现，在容器内获取的是上层物理机或者虚拟机的 CPU 核心数，这就使得线程池配置不符合我们期望的设置。

另一个影响体现在 GC 中，JVM 垃圾对象回收对 Java 程序执行性能有一定的影响，默认的 JVM 使用公式 “ParallelGCThreads = (ncpus <= 8) ? ncpus : 3 + ((ncpus * 5) / 8)” 来计算并行 GC 的线程数，其中 ncpus 是 JVM 发现的系统 CPU 个数。如果 JVM 应用了错误的 CPU 核心数，会导致 JVM 启动过多的 GC 线程，导致 GC 性能下降，Java 服务的延时增加。

总结

这一课时和你分享了容器技术的发展，以 Docker 为代表的容器化技术的实现原理，以及大规模容器化之下，微服务如何适配等问题。

本课时的内容以概念为主，如果你在工作中没有接触过容器化场景，可以到 Docker 官网学习入门指南、了解 Docker 命令，并动手实践一下 Docker 部署。

精选评论

Daniel：

> 现在都是分布式部署，大公司基本都全面容器化了，这部分内容挺有价值

****5125：**

> 容器化后，A-B服务之间如果采用gRPC通信时，长链接，如果被调用方B docker得ip发生变化，A服务无法感知B服务的Ip，此时改怎么处理呢？有没有好的方法

讲师回复：

> 容器IP变化以后，服务会有一个上下线过程