# 初识Dicker

## 问题引入：

在微服务中，有一个让人头疼的问题，就是服务部署问题，因为微服务非常的多，如果一个一个部署起来很麻烦，同时服务还会依赖各种应用（mysql、redis、mq、nginx、es。。。。。）部署起来很麻烦，docker正好可以解决这个问题。

## 什么是Docker:

### 项目部署的问题



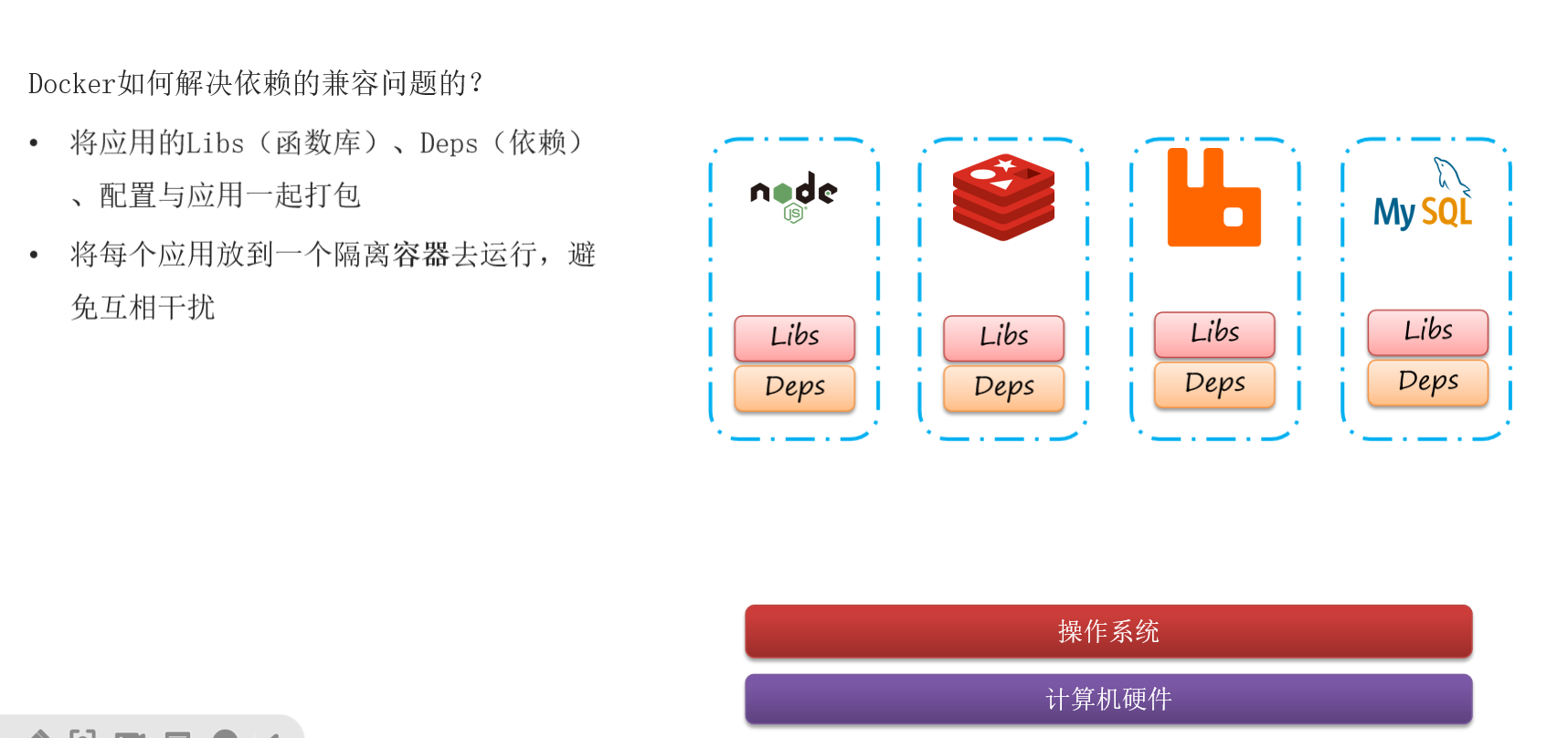
一个大型的项目，组件（即服务）往往很多（必然的，微服务架构嘛），运行环境也会很复杂，部署时通常会碰到一些问题，特别是微服务项目：各种各样的微服务，而且这些微服务还会依赖于各种各样的应用，比如说：前端部分，肯定会依赖node.js，而我们的服务端，还需要数据库（mysql）、缓存系统（redis）、异步通信（MQ）等等等等。。。。。。所有的应用都需要部署到服务器上，而大多数的服务器都会采用linux操作系统，这些应用在安装到Linux操作系统之前，往往要做很多的准备工作，因为这些应用都会有自己所需要的依赖和函数库，如：gcc、epel、libstdc++、glibc、scoat、binutils等等。

而每一个不同的应用，他们所需要的依赖和函数库可能有不同；或者一样，但是依赖和函数库版本不同，这个时候，依赖关系如此复杂，很容易产生兼容性问题，而且就算你花了很大功夫，终于解决了以上这些问题，你会发现，这仅仅是个开始，远远还没有结束，搞定了开发环境，还有测试环境、生产环境、预发布环境。。。在等着你去配置部署，更可怕的是，这些环境的linux系统还有可能不同，比如说，一个环境是centos系统，还有一个环境装的是Ubuntu系统。。。你在这个环境下配好了各种关系，你换到另外一个环境的操作系统同样的方式配几乎可以肯定是不能正常运行的，这就是我们目前在开发、部署时遇到的问题：效率极低！

就是微服务项目中，服务很多，服务运行需要依赖的应用很多，你要一个一个服务、一个一个应用部署吗？同时微服务架构中，服务必然做集群，那要部署的服务、应用就更多了，部署死你！！！

### 以上问题，Docker是如何解决的？

1：即然每一个应用，都有自己所需要的依赖和函数库，那为什么不把这些依赖、函数库跟应用一起打包呢，docker干的就是这些事情，比如，nodejs需要的libs（函数库、deps（依赖）、配置，我把你打包起来，形成一个整体；而后把这个整体放到一个隔离的容器中去运行，避免相互干扰。每一个应用可以都这么做，应用需要什么依赖、需要什么函数库，都提前准备好，然后和应用一起，打成一个包，将来要部署时，直接运行这个包，此时就不需要关心依赖的问题了！



**问：把上面应用扔到一台机器上，应用之间的依赖不会有干扰吗？**

答：不会！因为docker会把这些打包好的程序用一个隔离的容器去运行，会应用到沙箱的技术，他们相互之间是不可见的！

2：当然，1只是解决了我们说的依赖混乱的问题，即有了docker，依赖就不用去管了，应用程序随时都可以运行，但是，正常情况是仅限于同一个操作系统，因为我们在打包一个应用时，是基于某种操作系统去打包的，比如这个应用是ubuntu版本的，那么这个应用的依赖和函数库肯定也是Ubuntu版本的，把这个打包好的程序扔到centos上，是不能运行的。那么：不同环境的操作系统不同，Docker如何解决？

**要想了解Docker如何跨系统运行，我们要先了解下操作系统结构**

### linux系统的结构以及应用的执行原理



其实所有的Linux内核的操作系统都可以分为两层：一层是共享的一样的linux内核；二是上层的不同的系统应用（即不同linux的发行版本）。内核负责与计算机硬件进行交互，比如调用cpu、调用内存、读写文件、开辟进程等等，这些都是内核做的事，而内核会把以上这些事情变成一个一个指令，你调用这些指令就可以操作计算机硬件了，但是这些指令是往往比较简陋的，如果要基于这些指令去开发应用，那就太麻烦了，所以就有了系统应用，比如Ubuntu，Ubuntu会将内核指令进行封装，形成函数，许许多多的函数就形成了函数库，程序员基于这些函数库进行开发，实现功能：程序调用函数库，函数库调用内核指令，指令去调用计算机硬件，从而实现应用的执行，以上就是linux系统的结构以及应用的执行原理！

**那么问题来了，一个Ubuntu的应用为什么不能在Centos上运行呢？**

因为Ubuntu和Centos的内核是一样的，系统应用不同，也就是说函数库不同，存在Ubuntu上有的函数库，Centos上不一定有，或者说函数名字不一样，此时，如果把Ubuntu的应用（如Mysql）迁移到Centos上，尝试去执行，因为函数库不一样，应用去调用一个函数库时，会发现函数库在Centos不存在，肯定会程序报错。这就是应用不能跨系统运行的原因！

**针对这样的问题，Docker干了什么事？**

即然每个应用都依赖于系统函数库，为什么不把应用需要的系统函数库一起打包呢？这样一来，不就不愁找不到函数库了，所以Docker的解决方案就是：将用户程序与所需要调用的系统函数库（如Ubuntu）一起打包！比如说要打包Mysql应用，Mysql需要自己的依赖，同时还需要系统函数库（Ubuntu的函数库），那么就可以把需要用到的依赖、Ubuntu的函数库一起打包成一个整体，那么此时Mysql应用随便放到任何的linux操作系统上都可以运行，只要内核是Linux内核，Mysql应用执行的时候调用一起打包好的函数库，而不需要调用宿主机的函数库，而这个函数库直接调用操作系统的内核，而内核直接访问硬件，这个调用就完成了，就不需要管系统应用是什么了，因为这个打包好的应用自己就有，通过这种方式，Docker就解决了跨系统的问题，可以认为docker打包好的程序包可以运行在任何Linux内核的操作系统上：**Docker运行在不同操作系统时，直接基于打包的库函数，借助于操作系统的Linux内核来运行**。

### 总结

1：Docker如何解决大型项目依赖关系复杂，不同组件依赖的兼容性问题？

Docker允许开发中将应用、依赖、函数库、配置一起打包，形成可移值镜像；

Docker应用运行在容器中，使用沙箱机制，相互隔离，防止不同应用依赖之间的干扰。

2：Docker如何解决开发、测试、生产环境有差异的问题？

Docker镜像中包含完整运行环境，不仅仅是应和依赖，还包括系统函数库，仅需要依赖系统的Linux内核，而不是在linux内核之上的系统应用，因此可以在任意Linux操作系统上运行。

总结：

**Docker是一个快速交付应用、运行应用的技术！**

1：可以将程序及其依赖、运行环境（依赖、函数库、配置）一起打包为一个镜像，可以迁移到任意Linux操作系统；

2：运行时利用沙箱机制形成隔离容器，各个应用互不干扰，即docker容器之间相互隔离，互不干扰；

3：应用启动、移除都可以通过一行命令完成，方便快捷（因为这种完整的程序包，已经包含了自己所需要的一切，所以启动的时候只需要一个简单的命令就可以启动了，移除的时候也是如此，非常方便和快捷，因此做部署和运行时非常快速、高效）。