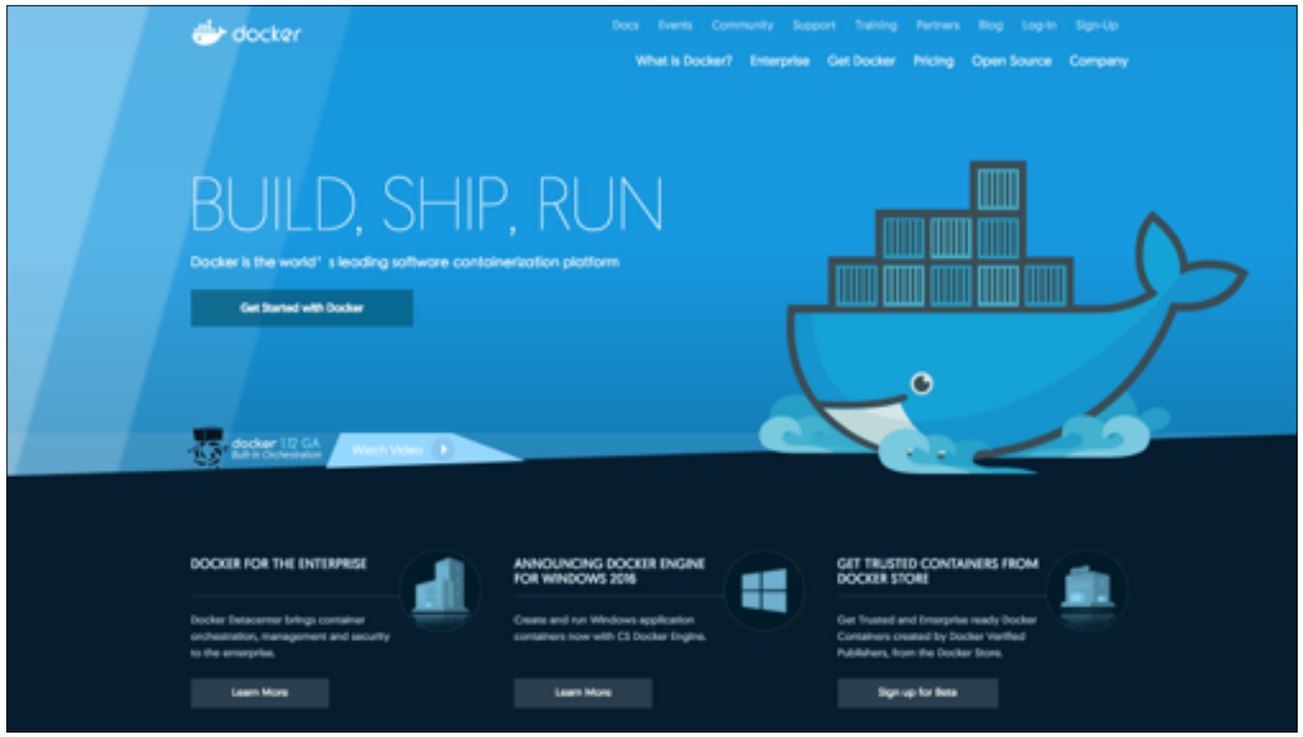
《Docker 的入门》

大家好，我是西电微信学生俱乐部的 Web Mentor 马寅彬。今天由我为大家带来今天的技术沙龙
原计划我们这周六做技术沙龙的，但因为设备问题，技术部做出的决定是延期这次技术沙龙。
但因为我不确定我之后的周末会有时间来给大家继续分享这次的主题，所以我决定把这次技术沙龙录成视频，然后公布给大家，大家可以选择自己合适的时间来自行的观看。

录成视频并不意味俱乐部的同学一定要来看这部视频，如果你感兴趣的话，你可以选择在闲暇时间来看，俱乐部并不会专门强制大家来看视频。
这次沙龙的主题主要由两个半场，第一个半场我主要讲一下 Docker ，借 Docker 给大家普及一下云计算和虚拟化的概念，并且会介绍 Docker 的架构和特色，最后会专门给大家展示一个基于 Docker 搭建的 Online Judge。这个半场并不会涉及过多的技术细节和编程问题，所以适合对虚拟化感兴趣的同学。
第二个半场我主要讲 OOP 的设计模式，我主要会用 Python 作为示例语言，并不意味只有指只有 Python 才适合。只要符合 OOP 模型的语言都适合来运用这些先进的设计模式。因为设计模式比较多，这次讲座我只会为大家介绍四个设计模式，而且这些设计模式在各个语言中有自己的具体运用，所以希望大家听了这次讲座以后，可以结合自己所会的语言，比如 Java，C++，甚至 Ruby，自行去结合各个语言自己有的语法糖来实现这些设计模式。
这就是这次沙龙的安排，让我们回归正题。
Docker 是最近虚拟化舞台非常火的一个角色，这只蓝色的鲸鱼的诞生纷纷引起的大家的主义。

Build Once, Run Anywhere.

这是 Docker 的宣传语，就像运输用的集装箱一样，字面意思是说一次打包后可以运输到任何地方。实际意思就是你只要使用 Docker 包装了你的应用程序，你就可以在任何一个支持 Docker 的平台上运行你的这个应用程序。在这里Docker 把自己比喻成货船，而包装的应用程序比喻成了集装箱。



这是 Docker 的官网，大家可以结束后自己来上这个官网来了解更多 Docker 的技术细节。

什么是Docker?

- 轻量、开放、安全的容器虚拟化平台
- 一次打包，多平台使用*
- 易于分享和部署

*：必须在同一架构

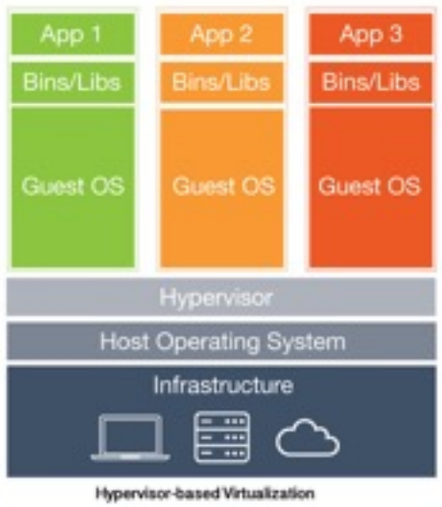
那么什么是 Docker 呢?
轻量…
这里大家只用先理解开放和安全两个词，我们先不看容器虚拟化和轻量这两个词。开放的意思是 Docker 的源码是向公众开放的，就在 Github 上面。安全就不用多说。
一次打包，多平台使用就是我刚刚给大家说的，build once run anywhere。这里的同一架构的意思是，你在 PC 上运行的 Docker ，打包的应用程序，可能会无法在你的安装了 Docker 的手机上运行，意思就是 X86 的 Docker 和 ARM 的 Docker 之间的集装箱是不通用的。
易于分享和部署，因为 Docker 的集装箱是通用的，你可以将这个集装箱发布到网上，然后在别的服务器上大量的复制你的集装箱运行起来，这对 Docker 来说非常的简单。

虚拟化?

虚拟化?

- 抽象：呈现外观，隐藏细节
- 在软件层上抽象出一台“计算机”任何一个部件或状态层
- 虚拟内存、虚拟平台、虚拟操作系统...

谈到虚拟化我们先提一个词，“抽象”。
这里说的抽象并不是说东西难以形容的“抽象”，而且指归纳总结后提炼结果的“抽象”。数学建模就是一种抽象的过程。实际上，计算机的整个发展都是围绕着抽象展开，比如计算机本身就是数学领域的一个模型，我们现在
用的计算机只是工程师按照这个模型搭建出来的工程成果。抽象的目标往往都是非常复杂繁琐的，而抽象的结果都是简单而又有规律的。这就是抽象。
虚拟化其实是抽象的一个具体而已，虚拟化其实就是 … ，比如在 Windows 10 下抽象出一个 Windows 8 的运行环境，其实就是我们所知的虚拟机，虚拟化其实是非常非常广的一个领域。
比如虚拟内存、虚拟平台、甚至虚拟一个操作系统，都是属于虚拟化的一部分。


App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host Operating System
Infrastructure
Hypervisor-based Virtualization

传统的虚拟机架构

云计算

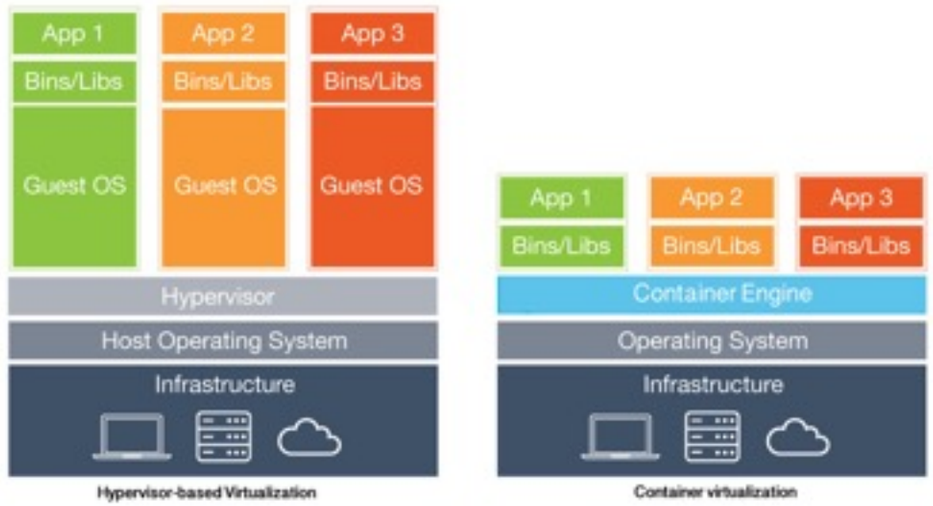
- 把计算力抽象成资源（类似水、电）
- 抽象计算机系统(IaaS)，抽象运行环境(PaaS)，抽象应用程序(SaaS)
- 灵活、高效、弹性

云计算其实是虚拟化发展到非常高程度后所得到的结果。
当计算机整个部分可以虚拟化出来的时候，就诞生了云计算这个概念。云计算就是把计算力抽象成了像水和电一样可以度量可以交易的资源，方便使用者根据自身的需求找云计算提供商来换取自己所需要的计算力，而不用去纠结具体的技术细节。比如把服务器放哪，服务器的网络怎么构建，选择什么样的操作系统这些问题。
现在云计算实际有三个领域，… 其实它们的翻译并不是这个意思，只是我为了方便大家理解给通俗化了。… IaaS 就是提供虚拟化的操作系统环境，比如阿里云和亚马逊。PaaS 就是提供虚拟化的应用程序运行的平台，通常这个平台都是指相应的编程语言运行环境，比如新浪的SAE。而 SaaS 就是提供虚拟化的应用程序，比如现成的数据库，类似的像 LeanCloud，这一块大家可能了解的比较少。
云计算的优点就是 … 这里说的灵活指的是部署灵活，因为整个运行环境都是虚拟的，这个环境可以放在北京甚至可以放在上海。高效指的是你不用再去处理具体的技术细节，从而带来的生产效率的提高。弹性是云计算的一个特色，比如我现在需要处理更多的计算任务，可以安排更多的内存甚至更多的处理器，因为你的运行环境都是虚拟化的，调整服务器的资源也非常的容易，而不用再去专门给服务器安装相应的零件。

云计算的目的

- “解放和发展生产力”
- 容易大量部署和快速备份
- 最终体现于应用程序

我们来看一下云计算的目的
第一点 … 其实也是云计算最突出的优点。云计算的诞生对于资源有限的用户来说能够将更多的力量集中于业务上，而不用关心繁杂和无意义的技术细节上。
第二点 … 其实是相对物理机来说的，“大量部署”很好的理解。因为虚拟化的优点，用户的东西对于提供商来说是非常容易进行备份的，对于提供商来说他不用再去停止服务器来备份用户的硬盘了，它只用将整个运行环境备份一份即可，这个过程非常迅速。
第三点 … 其实是对于大众来说云计算的最终目的，云计算提供的所有服务，实际上都是体现于应用程序的，这一点可以在之后的 Docker 中看出来。

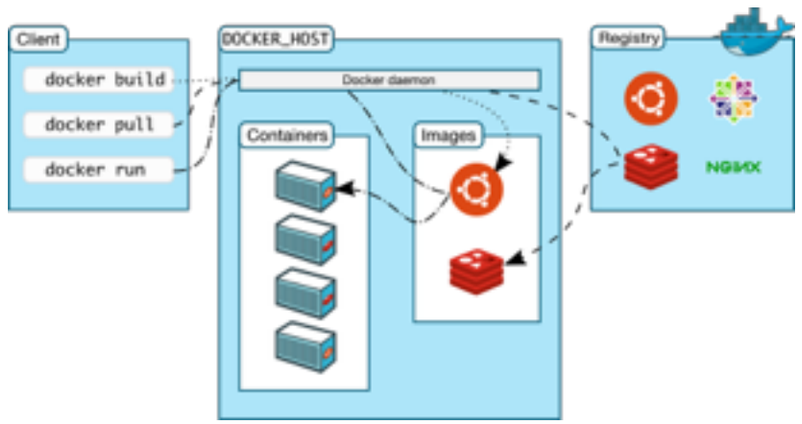


Docker 和传统虚拟机的对比

“集装箱运输和京东快递运输相比之下存在哪些优势？”

对比传统的 LXC 和 Docker

《Docker 的架构和特色》



Docker 的总架构图

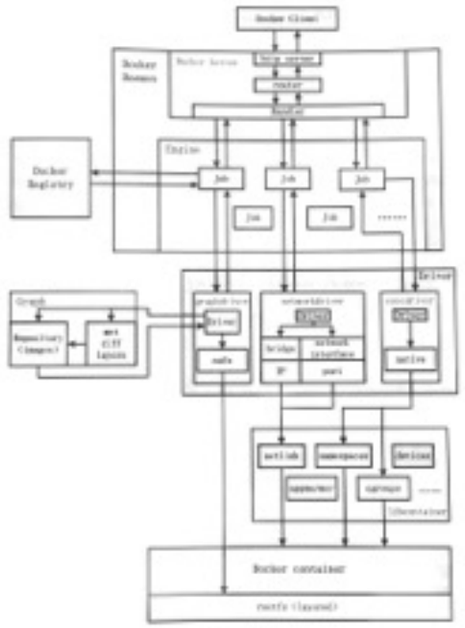
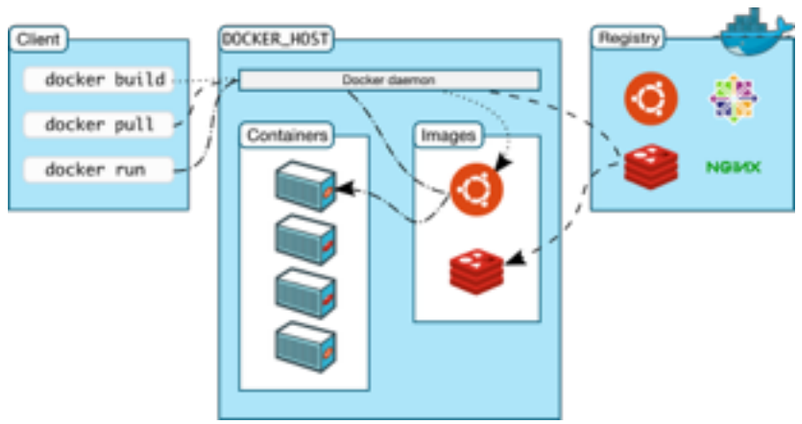


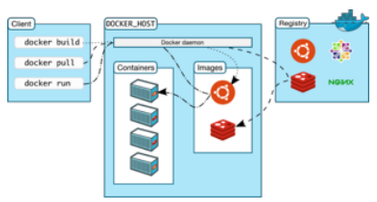
图 1-1 Docker 总架构图

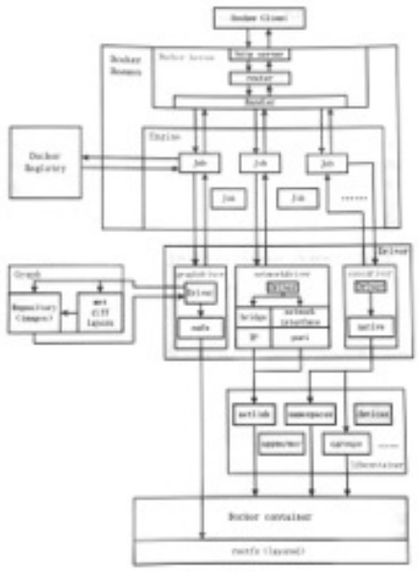


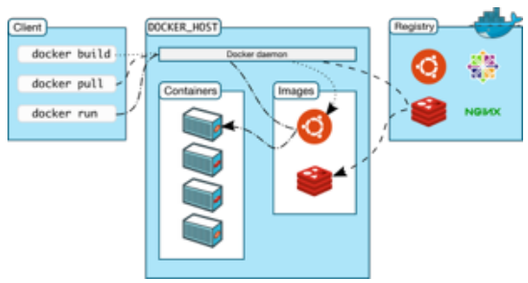
Docker 的总架构图

Docker 组成介绍

- client: 用户接口
- daemon: API 和任务调度
- images: 只能读取数据不能写的模板
- registries: 保存 images，可本地可远端
- containers: 管理“集装箱”




Docker Registry
Job
Job
Job
Job
Job
Docker container


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

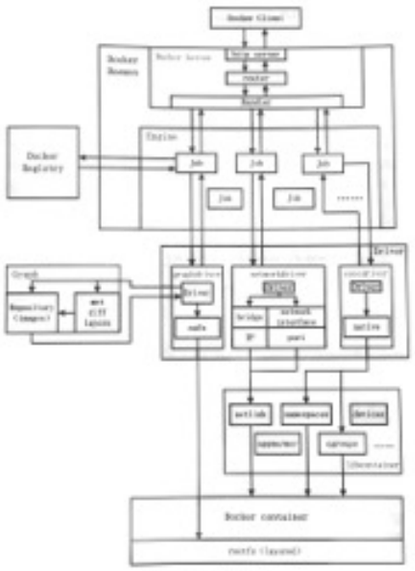
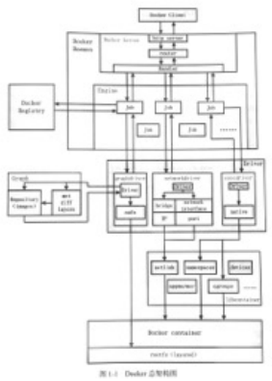


图 1.1 Docker 总架构图

libcontainer

- namespace: 隔离各个 container 的工作环境
  - pid,net,ipc,mnt,uls —— Linux Kernel
- cgroups: 控制资源分配(内存、CPU等)
- UFS: 统一格式，分层处理
- mount,network,device,netlink,security…



更多好玩的东西...

- Dockerfile: 快速且高效地构建一个 image
- Swarm,Machine,Compose：创建、管理 Docker 集群
- Docker(mine)craft: 史上最自由化的 GUI 管理界面

用 Docker 造一个 OnlineJudge

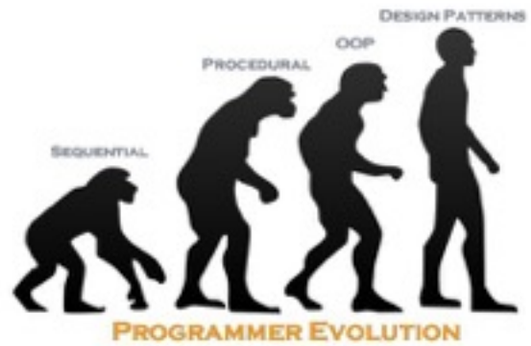
Cheers!

马寅彬
bin@xdmsc.club

《OOP 设计模式的介绍》


嗯对了，请一定强调
方法调用的本质是对象间的消息通信。
oop?
难假oop只打算将设计模式了
是调用者向被调用发送了一个消息
我就没打算深入oop本质。。
请一定要说明这个概念
就算他们不懂
切住了
未来的某一天会懂
或者学了ruby，马上懂
说OOP不说明这一点
你这就不像老司机了

Design patterns



设计模式是非常玄学非常抽象的东西，
而抽象化玄学化正是从复杂到简单必将经历的过程
其实我自己也讲不清楚设计模式是个什么东西

——马寅彬

设计模式

- 组织数据结构的设计方案
- 创建型、结构型、行为型
- 是一种思想，“解放生产力”

“设计模式是对被用来在特定场景下解决一般设计问题的类和相互通信的对象的描述”
创建型模式：抽象了实例化的过程，将如何创建、组合、表示对象的工作独立于系统之外。
结构型模式：设计如何组合类和对象以获得更大的结构，描述如何对一些对象进行组合，从而实现新功能的方法。
行为模式：设计算法和对象间的职能分配，不仅描述对象或类的模式，还描述他们之间的通信模式。
多用在重构产品。

设计模式

- 模式名称：叫什么
- 问题：使用范围
- 解决方案：设计的组成部分
- 效果：带来的效果以及弊病

设计模式

- 创造型：工厂模式
- 结构型：修饰器模式、MVC 模式
- 行为型：发布/订阅模式(观察者模式)

工厂模式

- 假设你要写一个做饭程序，需要提供火锅、烤肉、蛋糕、冒菜、臭豆腐、奶茶、串串香、日料、牛排、披萨...
- 而且以后支持做更多的菜
- 你会怎么处理这个问题?

工厂模式

- 问题：从相同作用不同类型对象之中选择一个返回
- 解决方案：抽象出一个接口，让对象按照接口规范使用
- 效果：将特定对象的创建和使用与架构分离（解耦）

修饰器模式

- “一切皆对象”
- 对象可以被函数修饰
- 函数也是对象

- –> 可以得出，函数可以被函数修饰

修饰器模式

- f(x) = x+1
- g(x) = 2x + 1
- f(g(x)) = g(x)+1 = 2x + 1 + 1 = 2x + 2

修饰器模式

- 假设你已经写了许多个相似结构做饭程序，比如中餐、西餐…
- 现在需要给所有做饭程序添加一个记录功能：向中心数据库记录何时做了什么菜品出来
- 你会怎么处理？回去调整源码重写每一个“做饭程序”？

修饰器模式

- 问题：给函数扩展功能
- 解决方案：写一个函数的函数，修饰(不是装饰的意思)这个函数
- 效果：在涉及代码量大或者专业性强的函数修改时，不用直接修改源码。

发布/订阅模式

- 假设你有两台机器来实现你的做饭程序，但是因为两台机器“先天”的问题，每台机器做出的口味有高有低，你需要事先决定使用哪台机器做哪道菜，事后发布做菜命令后，机器自行选择适合自己的菜品。
- 你会如何完成?

发布/订阅模式

- 问题：抽象调用类方法的具体过程(方法的调用本身就是消息的传递)
- 解决方案：加一个中间代理人，记录具体调用的过程
- 效果：方便程序发布后直接修改

消息的发送者不会发送其消息给特定的接收者，而且将发布的消息分成不同的类别进行直接发布，并不关注接收者是谁
而订阅者会对其中的一个或者多个类型感兴趣，且只接收感兴趣的消息，并不关注是谁发布了这个消息
这种发布者和接收者的解耦可以允许更好地可扩放性和更为动态的网络拓扑。

MVC 模式

- 模型–视图–控制器
- 控制器：负责转发请求，对请求进行处理。
- 视图：界面设计人员进行图形界面设计。
- 模型：算法、功能、数据

推荐书目

- 《精通 Python 的设计模式》—— 图灵电子书
- 《Python 的进阶》：https://eastlakeside.gitbooks.io/interpy-zh/content/
- 《编写高质量代码改善 Python 程序的 91 个建议》 —— 华章