## 1、简述Docker的特性？

Docker主要有如下特性：

标准化

 保证一致的运行环境

 弹性伸缩，快速扩容

 方便迁移

 持续集成、持续交付与持续部署

高性能

 不需要进行硬件虚拟以及运行完整的操作系统

轻量级

 快速启动

 隔离性

 进程隔离  
达到应用组件级别的"一次封装，到处运行"的目

## 2、简述Docker容器的几种状态？

Docker容器可以有四种状态：

 运行

 已暂停

 重新启动

 已退出

## 3、简述Dockerfile、Docker镜像和Docker容器的区别？

Dockerfile 是软件的原材料，Docker 镜像是软件的交付品，而 Docker 容器则可以认为是软件的运行态。从应用软件的角度来看，Dockerfile、Docker 镜像与 Docker 容器分别代表软件的三个不同阶段， Dockerfile 面向开发， Docker 镜像成为交付标准，Docker 容器则涉及部署与运维。

## 4、简述Docker与KVM（虚拟机）的区别？

1、隔离与共享

虚拟机通过添加Hypervisor层（虚拟化中间层），虚拟出网卡、内存、CPU等虚拟硬件，再在其上建立虚拟机， 每个虚拟机都有自己的系统内核。而Docker容器则是通过隔离（namesapce）的方式，将文件系统、进程、设备、网络等资源进行隔离，再对权限、CPU资源等进行控制（cgroup），最终让容器之间互不影响，容器无法影响宿主机。

容器与宿主机共享内核、文件系统、硬件等资源。

2、性能与损耗

与虚拟机相比，容器资源损耗要少。同样的宿主机下，能够建立容器的数量要比虚拟机多。

但是，虚拟机的安全性要比容器稍好， 要从虚拟机攻破到宿主机或其他虚拟机，需要先攻破 Hypervisor 层，这是极其困难的。

而docker 容器与宿主机共享内核、文件系统等资源， 更有可能对其他容器、宿主机产生影响。

|  |  |  |
| --- | --- | --- |
| 不同点 | Docker容器 | 虚拟机 |
| 启动速度 | 快，几秒钟 | 慢，几分钟 |
| 运行性能 | 接近原生（直接在内核中运行） | 运行于Hypervisor上，50%左右损失 |
| 磁盘占用 | 小，甚至几十KB（根据镜像层的情况） | 非常大，上GB |
| 并发性 | 一台宿主机可以启动成百上千个容器 | 最多几十个虚拟机 |
| 隔离性 | 进程级别 | 系统级别（更彻底） |
| 操作系统 | 主要支持Linux | 几乎所有（KVM） |
| 封装程度 | 只打包项目代码和依赖关系，共享宿主机内核 | 完整的操作系统， 与宿主机隔离 |

## 5、简述Docker主要使用的技术？

容器主要使用如下技术：

 Cgroup：资源控制

 Namespace：资源隔离

 rootfs：文件系统隔离

 容器引擎（用户态工具）：生命周期控制

docker本质就是宿主机的一个进程，docker是通过namespace（命名空间）实现资源隔离，通过cgroup实现资源限制，通过写时复制技术（copy -on-write）实现了高效的文件操作（类似虚拟机的磁盘比如分配500g并不是实际占用物理磁盘500g）

|  |  |  |
| --- | --- | --- |
| namespace的六项隔离 | | |
| namespace | 系统调用参数 | 隔离内容 |
| UTS | CLONE\_NEWUTS | 主机名与域名 |
| IPC | CLONE\_NEWWIPC | 信号量、消息队列和共享内存 |
| PID | CLONE\_NEWPID | 进程编号 |
| NETWORK | CLONE\_NEWNET | 网络设备、网络栈、端口等 |
| MOUNT | CLONE\_NEWNS | 挂载点（文件系统） |
| USER | CLONE\_NEWUSER | 用户和用户组（3.8以后的内核才支持） |

## 6、简述Docker体系架构？

Docker体系相对简单，主要涉及如下5个组件：

Docker客户端 – Docker

docker客户端则扮演着docker服务端的远程控制器，可以用来控制docker的服务端进程。

Docker服务端 – Docker Daemon资源限制

docker服务端是一个服务进程，管理着所有的容器。

Docker镜像 – Image

Docker的镜像是创建容器的基础，类似虚拟机的快照，可以理解为一个面向Docker容器引擎的只读模板。

通过镜像启动一个容器，一个镜像是一个可执行的包，其中包括运行应用程序所需要的所有内容包含代码，运行时间，库、环境变量、和配置文件

Docker镜像也是一个压缩包，只是这个压缩包不只是可执行文件，环境部署脚本，它还包含了完整的操作系统。因为大部分的镜像都是基于某个操作系统来构建，所以很轻松的就可以构建本地和远端一样的环境，这也是Docker镜像的精髓。

 Docker容器 – Docker Container

Docker的容器是从镜像创建的运行实例，它可以被启动、停止和删除。所创建的每一个容器都是相互隔离、互不可见，以保证平台的安全性。

可以把容器看做是一个简易版的linux环境（包括root用户权限、镜像空间、用户空间和网络空间等）和运行在其中的应用程序。

 Docker镜像仓库 -- Registry

Docker仓库是用来集中保存镜像的地方，当创建了自己的镜像之后，可以使用push命令将它上传到公有仓库（Public）或者私有仓库（Private、常用）。当下次要在另外一台机器上使用这个镜像时，只需从仓库获取。

Docker 的镜像、容器、日志等内容全部都默认存储在/var/lib/docker目录下。

## 7、简述Docker如何实现网络隔离？

Docker利用了网络的命名空间特性，实现了不同容器之间的网络隔离。命名空间可以支持网络协议栈的多个实例，独立的协议栈被隔离到不同的命名空间中。

因此处于不同命名空间中的网络栈是完全隔离的，彼此之间无法通信。每个独立的命名空间中可以有自 己独立的路由表及独立的iptables设置来提供包转发、 NAT及IP包过滤等功能。

## 8、简述Linux文件系统和Docker文件系统？

Linux文件系统： 由bootfs和rootfs组成， bootfs主要包含bootloader和kernel， bootloader主要是引 导加载kernel，当kernel被加载到内存之后bootfs就被卸载掉了。 rootfs包含的就是典型linux系统 中/dev,/proc,/bin,/etc等标准目录。

Docker文件系统： Docker容器是建立在Aufs分层文件系统基础上的， Aufs支持将不同的目录挂载到同 一个虚拟文件系统下，并实现一种layer的概念。 Aufs将挂载到同一虚拟文件系统下的多个目录分别设置 成read-only， read-write以及whiteout-able权限。 docker 镜像中每一层文件系统都是read-only。

## 9、简述Docker网络模式？

Docker使用Linux的Namespaces技术来进行资源隔离，其中Network Namespace实现隔离网络。

一个Network Namespace提供了一份独立隔离的网络环境，包括网卡、路由、 Iptable规则等，Docker 网络有如下五种模式：

host模式

host模式下容器将不会获得独立的Network Namespace，该模式下与宿主机共用一个Network Namespace。容器将不会虚拟出自己的网卡，不会配置独有的IP等，而是使用宿主机的 IP和端口。

container模式

Container 网络模式是 Docker 中一种较为特别的网络的模式，处于container模式下的 Docker 容器会共享其他容器的网络环境，因此，两个或以上的容器之间不存在网络隔离， 而配置container模式的容器又与宿主机以及除此之外其他的容器存在网络隔离。很多容器共享一个ip地址

none模式

none模式下，Docker容器拥有自己的Network Namespace，但是，并不为Docker容器进行任何网络配置和构造任何网络环境，关闭了容器的网络功能。Docker 容器采用了none 网络模式，那么容器内部就只能使用loopback网络设备，不会再有其他的网络资源。Docker Container的none网络模式意味着不给该容器创建任何网络环境，容器只能使用127.0.0.1的本机网络。只有lo接口，没有其他的接口

bridge模式

bridge模式是Docker默认的网络设置，此模式会为每一个容器分配NetworkNamespace、设置IP等，并将该宿主机上的Docker容器连接到一个虚拟网桥上。

docker bridge容器只能通过docker内部bright网络

自定义网络

## 10、简述Docker跨主机通信的网络实现方式？

docker跨主机通信按原理可通过以下三种方式实现：

 直接路由方式：直接在不同宿主机之间添加静态路由；

 桥接方式（如pipework）：通过静态指定容器IP为宿主机IP同一个网络的形式，即可实现。

 Overlay隧道方式： 使用overlay网络实现， Overlay网络指在现有网络层之上叠加的虚拟化技术，

 实现应用在网络上的承载，并能与其他网络业务分离，并且以基于IP的网络技术为主，如flannel、 ovs+gre。

## 11、简述flannel网络模型实现原理？

Flannel为每个host分配一个subnet，容器从subnet中分配IP，这些IP可以在host间路由，容器间无需使用nat和端口映射即可实现跨主机通信。每个subnet都是从一个更大的IP池中划分的，flannel会在每 个主机上运flanneld的agent，负责从池子中分配subnet。

Flannel使用etcd存放网络配置、已分配的subnet、 host的IP等信息， Flannel数据包在主机间转发是由

backend实现的，目前已经支持UDP、VxLAN、 host-gw、AWS VPC和GCE路由等多种backend。

安装flannel工具后，数据从源node节点的pod发出后，会经由 docker0 网卡转发到 flannel0 网卡，在flannel0 网卡的flanneld 服务会对其进行监听，把数据包里面的pod IP（源podip和目的pudip报文头）封装到 udp 报文中，然后根据自己在etcd中维护的路由表信息通过物理网卡转发到pod所在的目标node节点上，数据包到达目标node节点后会被 flanneld 服务解封装暴露出udp报文中的podip地址，根据目的pudip地址经由该主机上的flannel0 网卡和docker0网卡转发到目标 pod

## 12、Flannel和Calico的区别

Flannel 常见采取 UDP Overlay 方案，VxLAN 性能比 TUN 强一点，一个是内核态一个是用户态。flannel在进行路由转发的基础上进行了封包解包的操作，这样浪费了CPU的计算资源

Calico 是一个纯三层的方案，不需要 Overlay，基于 Etcd 维护网络准确性，也基于 Iptables 增加了策略配置Cilium 就厉害了，基于 eBPF 和 XDP 的方案，eBPF/XDP 处理数据包的速度可以和 DPDK 媲美，零拷贝直接内核态处理，缺点就是用户不太容易进行配置。

## 12、联合文件系统和镜像加载原理（Docker镜像层构建）

联合文件系统（UnionFS）

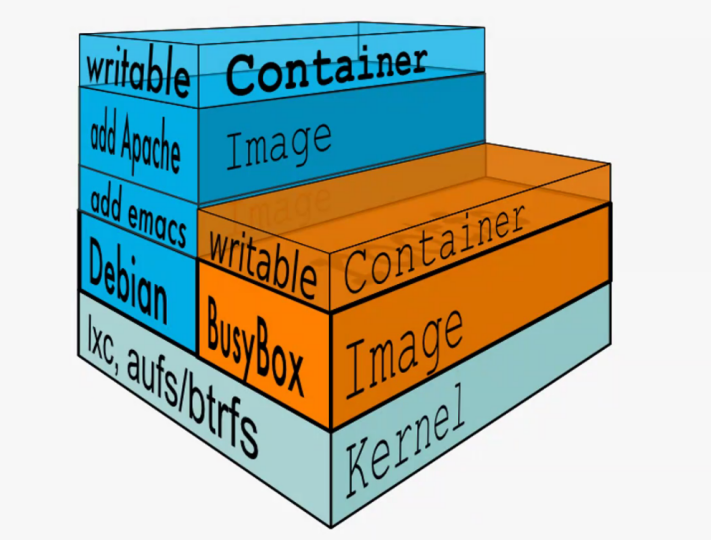
UnionFS（联合文件系统）：Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下。AUFS、OverlayFS 及 Devicemapper 都是一种 UnionFS（现在用的overlay2联合文件系统）。

Union文件系统是Docker镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

特性∶一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录。

我们下载的时候看到的一层层的就是联合文件系统。

镜像加载原理



Docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统就是UnionFS。

bootfs主要包含bootloader和kernel，bootloader主要是引导加载kernel，Linux刚启动时会加载bootfs文件系统。

在Docker镜像的最底层是bootfs，这一层与我们典型的Linux/Unix系统是一样的，包含boot加载器和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权己由bootfs转交给内核，此时系统也会卸载bootfs（bootfs加载宿主机内核并提供与镜像共享）。

rootfs在bootfs之上。包含的就是典型Linux系统中的/dey、/proc、/bin、/etc等标准目录和文件。rootfs就是各种不同的操作系统发行版，比如Ubuntu、Centos等等（rootfs提供相同的运行环境，不同发行的操作系统和目录环境）

简答来说一开始bootfs引导加载宿主机内核，所有镜像的运行创建基于共享宿主机的内核，操作一个命令下载debian（通用操作系统），这时就会在内核上面加了一层基础镜像，再安装一个emacs， 会在基础镜像上叠加一层image，接着再安装一个apache，又会在images上面再叠加一层image。最后它们看起来就像一个文件系统即容器的rootfs。在Docker的体系里把这些rootfs叫做Docker的镜像。但是，此时的每一层rootfs都是read-only的，我们此时还不能对其进行操作。当我们创建一个容器，也就是将Docker镜像进行实例化，系统会在一层或是多层read-only的rootfs（只读层）之上分配一层空的read-write的rootfs（可读可写层）。

在镜像实例化运行就会附加可读可写层，也就是容器化运行，用户会对可读可写层进行操作，数据会只会存储在读写层，伴随着容器的生命周期。不会对镜像本身进行操作，如果删除容器，那之前在读写层进行的操作数据也会消失，读写层之前累计的层数都为只读层

## 13、Dockerfile是什么，常用的命令？（ADD和COPY，ENTRYPOINT和CMD的作用和区别）

Dockerfile 是用来构建自定义镜像的文本文档

Dockerfile 里包含的信息主要有：

基础镜像

镜像元信息

镜像操作指令

容器启动时执行的命令

在平常我们主要围绕基础镜像信息、维护者信息、镜像操作指令和容器启动时执行指令这四个部分创建镜像

第一条 使用FROM命令指定基于的基础镜像

第二条 一般使用MAINTAINER添加一些维护者的信息

后面主要是镜像操作的指令例如：RUN（基于镜像执行命令），EXPOSE（镜像需要开启的端口），ENV（设置些环境变量），ADD和COPY（复制本地文件到镜像命令），VOLUME（挂载），WORKDIR（切换到某个路径），USER（运行容器的用户）

最后 可以可以使用ENTRYPOINT和CMD命令指定容器启动时默认执行的命令

##ENTRYPOINT和CMD命令的作用

ENTRYPOINT是容器运行时执行的第一条命令，CMD是容器启动时默认执行的命令，ENTRYPOINT需要严格遵守exec形式，CMD命令可以使用shell形式

ENTRYPOINT的优先级比CMD命令更高，如果同时存在会覆盖CMD指定的命令，docker run指定的命令优先级是最高的CMD可以给ENTRYPOINT传参

Dockerfile中只能有一条CMD命令，多条CMD只会执行最后一条

一般用ENTRYPOINT和CMD指定一条前台命令作为容器PID为1的命令，保证容器的正常运行不退出

##ADD和COPY命令的作用和区别

他们都具有复制本地文件、目录到镜像的功能

add在复制tar压缩包的归档文件时候自动支持解压缩，并且支持使用url路径拉取文件，但使用url拉取和解压特性不能同时使用

copy只能复制本地文件/目录到镜像

## 14、退出容器后，通过docker ps命令查看不到，数据会丢失吗？

可以用 docker ps -a 来查看

如果仅仅是退出容器的话，数据是不会丢失的，重启容器后，数据依然在

如果删除容器的话，但数据做了持久化存储，也不会丢。

容器删除，那数据也会一起被删除

## 15、docker 的优点

资源隔离：比如限制应用最大内存使用量，或者资源加载隔离等。

低消耗：虚拟化本身带来的损耗需要尽量的低。

Docker 很好的权衡了两者，即拥有不错的资源隔离能力，又有很低的虚拟化开销。

## 16、docker的应用场景

1、可以简化配置

不同的软件可能需要不同的运行环境，docker可以将运行环境和配置放在代码中，然后部署，同一个docker可以在不同的环境中使用

2、简化代码流水线（Code Pipeline）

代码从开发者的机器到最终在生产环境上的部署，需要经过很多的中间环境。而每一个中间环境都有自己微小的差别，Docker给应用提供了一个从开发到上线均一致的环境，让代码的流水线变得简单不少。

3、提高开发效率

这就带来了一些额外的好处：Docker能提升开发者的开发效率，不同的开发环境中，我们都想把两件事做好。一是我们想让开发环境尽量贴近生产环境，二是我们想快速搭建开发环境。理想状态中，要达到第一个目标，我们需要将每一个服务都跑在独立的虚拟机中以便监控生产环境中服务的运行状态。然而，我们却不想每次都需要网络连接，每次重新编译的时候远程连接上去特别麻烦。这就是Docker做的特别好的地方，开发环境的机器通常内存比较小，使用虚拟机的时候，我们经常需要为开发环境的机器加内存，而Docker可以轻易的让几十个服务在Docker中跑起来。

4、隔离应用

有很多种原因会让你选择在一个机器上运行不同的应用，比如之前提到的提高开发效率的场景等。

5、整合服务器

正如通过虚拟机来整合多个应用，Docker隔离应用的能力使得Docker可以整合多个服务器以降低成本。由于没有多个操作系统的内存占用，以及能在多个实例之间共享没有使用的内存，Docker可以比虚拟机提供更好的服务器整合解决方案。

6、调试能力

Docker提供了很多的工具，这些工具不一定只是针对容器，但是却适用于容器。它们提供了很多的功能，包括可以为容器设置检查点、设置版本和查看两个容器之间的差别，这些特性可以帮助调试Bug。

7、 快速部署

在虚拟机之前，引入新的硬件资源需要消耗几天的时间。Docker的虚拟化技术将这个时间降到了几分钟，Docker只是创建一个容器进程而无需启动操作系统，这个过程只需要秒级的时间。这正是Google和Facebook都看重的特性。你可以在数据中心创建销毁资源而无需担心重新启动带来的开销。通常数据中心的资源利用率只有30%，通过使用Docker并进行有效的资源分配可以提高资源的利用率。

## 17、docker-compose 是什么？

容器编排工具

例如：我们现在需要启动10个容器，其中3个nginx，2个redis，3个mysql，1个zabbix，1个ansible,有些容器需求先启动，有容器需要后启动，在启动的时候是有先后顺序的。

这时候需要批量启动容器，而且启动的时候容器之间是有依赖关系，需要考虑启动顺序的

我们可以将编排的内容全部写到一个yaml文件里，docker 的compose根据这个yaml文件里的安排去启动容器。

## 18、解释基本的Docker使用工作流程

一切都从Dockerfile开始。Dockerfile是镜像的源代码。

创建Dockerfile后，您可以构建它以创建容器的镜像。图像只是“源代码”的“编译版本”，即Dockerfile。

获得容器的镜像后，应使用注册表重新分发容器。注册表就像一个git存储库 - 你可以推送和拉取图像。

接下来，您可以使用该图像来运行容器。在许多方面，正在运行的容器与虚拟机（但没有虚拟机管理程序）非常相似。

## 19、Docker Image和Layer有什么区别？

Image：Docker镜像是由一系列只读层构建的

Layer：每个层代表Dockerfile中的指令。

下面的Dockerfile包含四个命令，每个命令都创建一个层。

FROM ubuntu:15.04

COPY . /app

RUN make /app

CMD python /app/app.py

每个层只是与之前层的一组差异。

## 20、如何监控生产中的Docker？

Docker提供docker stats和docker events等工具来监控生产中的Docker。我们可以使用这些命令获取重要统计数据的报告。

Docker stats：当我们使用容器ID调用docker stats时，我们获得容器的CPU，内存使用情况等。它类似于Linux中的top命令。

Docker events：Docker events是一个命令，用于查看Docker守护程序中正在进行的任务。

一些常见的Docker事件是：attach，commit，die，detach，rename，destroy等。我们还可以使用各种选项来限制或过滤我们感兴趣的事件。

## 21、什么是Docker Swarm模式？

Docker Swarm是Docker的群集管理工具。它将Docker主机池转变为一个虚拟Docker主机。Docker Swarm提供标准的Docker API，任何已经与Docker守护进程通信的工具都可以使用Swarm扩展到多个主机。

## 22、docker的持久化，网络，发布

docker的持久化：在生成容器的同时，加上-v选项，指定把当前服务器的目录映射到容器中

docker的网络：

创建网络：docker network create --subnet=10.10.10.0/24 docker1，其实就是创建一个虚拟交换机、一个虚拟局域网，只有属于同一个虚拟局域网的容器之间才能够相互通讯

发布docker：我们可以使用-p参数把容器端口和宿主机端口绑定

-p 宿主机端口:容器端口 #宿主端口可以是任意的端口，只要访问的时候加上端口号，即可访问到相应的容易的端口的服务

## 23、怎么做一个拥有指定功能的镜像

方法一：生产一个新的容器==>进入模板镜像容器=>操作(制作成自己想要的环境)=>退出容器=>利用容器产生新的镜像

方法二：写Dockerfile文件，把需要对容器的操作写到文件中，通过执行脚本文件来产生新的镜像

## 24、说下对docker run和docker start的理解？

docker run 只在第一次运行时使用，将镜像放到容器中，以后再次启动这个容器时，只需要使用命令docker start 即可。

docker run相当于执行了两步操作：将镜像放入容器中（docker create）,然后将容器启动，使之变成运行时容器（docker start）。

docker start的作用是，重新启动已存在的容器。也就是说，如果使用这个命令，我们必须事先知道这个容器的ID，或者这个容器的名字，我们可以使用docker ps找到这个容器的信息。

## 25、docker stop和docker kill的的理解？

kill是不管容器同不同意，我直接执行kill -9，强行终止；stop的话，首先给容器发送一个TERM信号，让容器做一些退出前必须的保护性、安全性操作，然后让容器自动停止运行，如果在一段时间内，容器还是没有停止，再进行kill -9，强行终止。

## 26、Harbor是什么、特性、工作流程（私有仓库）

Harbor 是 VMware 公司开源的企业级 Docker Registry项目，其目标是帮助用户迅速搭建一个企业级的 Docker Registry 服务。

Harbor以Docker公司开源的 Registry 为基础，提供了图形管理 UI 、基于角色的访问控制(Role Based AccessControl) 、AD/LDAP 集成、以及审计日志（Auditlogging）等企业用户需求的功能，同时还原生支持中文。

Harbor 的每个组件都是以 Docker 容器的形式构建的，使用 docker-compose 来对它进行部署。用于部署 Harbor 的 docker-compose 模板位于 harbor/docker-compose.yml。

Harbor的特性

1、基于角色控制：用户和仓库都是基于项目进行组织的，而用户在项目中可以拥有不同的权限。

2、基于镜像的复制策略：镜像可以在多个Harbor实例之间进行复制（同步）。

3、支持 LDAP/AD：Harbor 可以集成企业内部已有的 AD/LDAP（类似数据库的一张表），用于对已经存在的用户认证和管理。

4、镜像删除和垃圾回收：镜像可以被删除，也可以回收镜像占用的空间。

5、图形化用户界面：用户可以通过浏览器来浏览，搜索镜像仓库以及对项目进行管理。

6、审计管理：所有针对镜像仓库的操作都可以被记录追溯，用于审计管理。

7、支持 RESTful API：RESTful API 提供给管理员对于 Harbor 更多的操控, 使得与其它管理软件集成变得更容易。

8、Harbor和docker registry的关系：Harbor实质上是对docker registry（docker原生的私有仓库）做了封装，扩展了自己的业务模板。

Harbor的构成

Harbor 在架构上主要有 Proxy、Registry、Core services、Database（Harbor-db）、Log collector（Harbor-log）、Job services 六个组件。

●Proxy: Harbor的Registry、UI、Token 服务等组件，都处在 nginx 反向代理后边。该代理将来自浏览器、docker clients 的请求转发到后端不同的服务上（代理的作用）。

●Registry: 负责储存 Docker 镜像，并处理 Docker push/pull 命令。由于要对用户进行访问控制，即不同用户对 Docker 镜像有不同的读写权限，Registry 会指向一个 Token（令牌验证）服务，强制用户的每次 Docker pull/push 请求都要携带一个合法的 Token， Registry 会通过公钥对 Token 进行解密验证（存储镜像，处理拉去和推送命令，如果没有Token服务会指向一个Token服务）。

●Core services: Harbor的核心功能，主要提供以下3个服务:

1）UI（harbor-ui）: 提供图形化界面，帮助用户管理 Registry 上的镜像（image）, 并对用户进行授权。

2）WebHook：为了及时获取Registry 上image 状态变化的情况，在Registry 上配置 Webhook，把状态变化传递给 UI 模块。

3）Token 服务：负责根据用户权限给每个 Docker push/pull 命令签发 Token。Docker 客户端向 Registry 服务发起的请求， 如果不包含 Token，会被重定向到 Token 服务，获得 Token 后再重新向 Registry 进行请求。

●Database（harbor-db）：为core services提供数据库服务，负责储存用户权限、审计日志、Docker 镜像分组信息等数据。

●Job services: 主要用于镜像复制，本地镜像可以被同步到远程 Harbor 实例上（与其他Harbor对接，同步）。

●Log collector（harbor-log）: 负责收集其他组件的日志统一到一个地方。

Harbor 的每个组件都是以 Docker 容器的形式构建的，因此，使用 Docker Compose 来对它进行部署。

总共分为7个容器运行，通过在docker-compose.yml所在目录中执行 docker-compose ps 命令来查看， 名称分别为：nginx、harbor-jobservice、harbor-ui、harbor-db、harbor-adminserver、registry、harbor-log。

其中 harbor-adminserver 主要是作为一个后端的配置数据管理，并没有太多的其他功能。harbor-ui 所要操作的所有数据都通过 harbor-adminserver 这样一个数据配置管理中心来完成。

## 27、什么是服务注册与发现

服务注册与发现是微服务架构中不可或缺的重要组件。起初服务都是单节点的，不保障高可用性，也不考虑服务的压力承载，服务之间调用单纯的通过接口访问。直到后来出现了多个节点的分布式架构，起初的解决手段是在服务前端负载均衡，这样前端必须要知道所有后端服务的网络位置，并配置在配置文件中。这里就会有几个问题：

●如果需要调用后端服务A-N，就需要配置N个服务的网络位置，配置很麻烦

●后端服务的网络位置变化，都需要改变每个调用者的配置

既然有这些问题，那么服务注册与发现就是解决这些问题的。后端服务A-N可以把当前自己的网络位置注册到服务发现模块，服务发现就以K-V（键值对）的方式记录下来，K一般是服务名，V就是IP:PORT。服务发现模块定时的进行健康检查，轮询查看这些后端服务能不能访问的了。前端在调用后端服务A-N的时候，就跑去服务发现模块问下它们的网络位置，然后再调用它们的服务。这样的方式就可以解决上面的问题了，前端完全不需要记录这些后端服务的网络位置，前端和后端完全解耦！

## 28、Docker consul的容器服务更新与发现

consul是google开源的一个使用go语言开发的服务管理软件。支持多数据中心、分布式高可用的、服务发现和配置共享。采用Raft算法，用来保证服务的高可用。内置了服务注册与发现框架、分布一致性协议实现、健康检查、Key/Value存储、多数据中心方案，不再需要依赖其他工具（比如ZooKeeper等）。服务部署简单，只有一个可运行的二进制的包。每个节点都需要运行agent，他有两种运行模式server 和 client。 每个数据中心官方建议需要3或5个server节点以保证数据安全，同时保证server-leader的选举能够正确的进行。

在client模式下，所有注册到当前节点的服务会被转发到server节点，本身是不持久化这些信息。

在server模式下，功能和client模式相似，唯一不同的是，它会把所有的信息持久化到本地，这样遇到故障，信息是可以被保留的。

server-leader是所有server节点的老大，它和其它server节点不同的是，它需要负责同步注册的信息给其它的server节点，同时也要负责各个节点的健康监测。

## 29、consul特性

服务注册与发现：consul通过DNS或者HTTP接口使服务注册和服务发现变的很容易，一些外部服务，例如saas提供的也可以一样注册。

健康检查：健康检测（探针）使consul可以快速的告警在集群中的操作。和服务发现的集成，可以防止服务转发到故障的服务上面。

Key/Value存储：一个用来存储动态配置的系统。提供简单的HTTP接口，可以在任何地方操作。

多数据中心：无需复杂的配置，即可支持任意数量的区域。

安装consul是用于服务注册，也就是容器本身的一些信息注册到consul里面，其他程序可以通过consul获取注册的相关服务信息，这就是服务注册与发现。 [](af://n1029)

## 30、Docker 安全方面配置

下面从内核、主机、网络、镜像、容器以及其它等 6 个方面总结 Docker 安全基线标准。

1、内核级别

（1）及时更新内核

（2）User NameSpace （容器内的 root 权限在容器之外处于非高权限状态，容器内的root在外部权限减小）

（3）Cgroups（对资源的配额和度量），设置CPU、内存、磁盘 IO等资源限制。

（4）通过启用SELinux/AppArmor/GRSEC（控制文件访问权限）适当的强化系统来增加额外的安全性，使得即使受到攻击，黑客获取了root权限也无法篡改系统配置。

（5）Capability（权限划分），比如划分指定的CPU给容器。

（6）Seccomp（限定系统调用）限制不必要的系统调用。

（7）禁止将容器的命名空间与宿主机进程命名空间共享，比如 host 网络模式。

2、主机级别

（1）为容器创建独立分区，比如创建在分布式文件系统上

（2）仅运行必要的服务，注意尽量避免在容器中运行 ssh 服务

（3）禁止将宿主机上敏感目录映射到容器，-v创建数据卷时需要注意。

（4）对 Docker 守护进程、相关文件和目录进行审计，防止有病毒或木马文件生成。

（5）设置适当的默认文件描述符数。（文件描述符∶简称fd，当应用程序请求内核打开/新建一个文件时，内核会返回一个文件描述符用于对应这个打开/新建的文件，文件描述符本质上就是个非负整数，读写文件也是需要使用这个文件描述符来指定待读写的文件的。文件描述符是一个重要的系统资源，理论上系统内存多大就应该可以打开多少个文件描述符，但实际情况，内核会有系统级限制，以及用户级限制，不让某一个应用程序进程消耗掉所有的文件资源，可以使用ulimit -n 查看）

（6）用户权限为root 的 Docker 相关文件的访问权限应该为 644 或者更低权限。

（7）周期性检查每个主机的容器清单，并清理不必要的容器。

3、网络级别

（1）通过 iptables 设定规则实现禁止或允许容器之间网络流量

（2）允许 Docker 修改 iptables

（3）禁止将 Docker 绑定到其他已使用的 IP/Port 或者 Unix Socket

（4）禁止在容器上映射特权端口

（5）容器上只开放所需要的端口

（6）禁止在容器上使用 host网络模式

（7）若宿主机有多个网卡，将容器进入流量绑定到特定的主机网卡上。

docker network create --subnet=172.18.0.0/16 --opt "com.docker.network.bridge.name"="docker1" mynetwork

docker run -itd --net mynetwork --ip l72.18.0.100 centos:7 /bin/bash

iptables -t nat -A POSTROUTING -s 172.18.0.100 -o ens36 -j SNAT --to-source 192.168.150.30

4、镜像级别

（1）创建本地私有镜像仓库服务器

（2）镜像中软件都为最新版本，建议根据实际情况使用对应版本，业务稳定优先

（3）使用可信镜像文件，并通过安全通道下载

（4）重新构建镜像而非对容器和镜像打补丁，销毁异常容器重新构建

（5）合理管理镜像标签，及时移除不再使用的镜像

（6）使用镜像扫描

（7）使用镜像签名

5、容器级别

（1）容器最小化，操作系统镜像最小集

（2）容器以单一主进程的方式运行

（3）禁止--privileged 标记使用特权容器。

（4）禁止在容器上运行 ssh 服务，尽量使用 docker exec 进入容器

（5）以只读的方式挂载容器的根目录系统，-v 宿主机目录:容器目录:ro

（6）明确定义属于容器的数据盘符（将容器放入到专门的系统当中）。

（7）通过设置 on-failure 限制容器尝试重启的次数，容器反复重启容易丢失数据，--restart=on-failure:3

（8）限制在容器中可用的进程数，docker run -m 限制内存的使用，以防止fork炸弹。

（fork炸弹，迅速增长子进程，耗尽系统进程数量例如“. () {.|.&};.”调用一个函数前台后台不断的调用导致内存崩溃

6、其他设置

（1）定期对宿主机系统及容器进行安全审计

（2）使用最少资源和最低权限运行容器，此为 Docker 容器安全的核心思想

（3）避免在同一宿主机上部署大量容器，维持在一个能够管理的数量

（4）监控 Docker 容器的使用，性能以及其他各项指标，比如 zabbix

（5）增加实时威胁检测和事件报警响应功能，比如 zabbix

（6）使用中心和远程日志收集服务，比如 ELK。

更多安全技术可直接参阅 Docker 官方文档，https://docs.docker.com/engine/security/

## 31、https流程（TLS证书访问）

为了防止链路劫持、会话劫持等问题导致 Docker 通信时被中间人攻击，c/s（客户端）两端应该通过 TLS（SSL的加强版）加密方式通讯。

通过在服务端上创建tls密钥证书，再下发给客户端，客户端通过私钥访问容器，这样就保证的docker通讯的安全性。

使用证书访问的工作流程：

（1）客户端发起请求，连接到服务器的进程端口。

（2）服务器必须要有一套数字证书（证书内容有公钥、证书颁发机构、失效日期等）。

（3）服务器将自己的数字证书发送给客户端（公钥在证书里面，私钥由服务器持有）。

（4）客户端收到数字证书之后，会验证证书的合法性。如果证书验证通过，就会生成一个随机的密钥对，用证书的公钥加密。

（5）客户端将公钥加密后的密钥发送到服务器。

（6）服务器接收到客户端发来的密文密钥之后，用自己之前保留的私钥对其进行非对称解密，解密之后就得到客户端的密钥，然后用客户端密钥对返回数据进行对称加密，这样传输的数据都是密文了。

（7）服务器将加密后的密文数据返回到客户端。

（8）客户端收到后，用自己的密钥对其进行对称解密，得到服务器返回的数据。

总结：服务端首先创建一套ca证书，当收到客户端的请求时，服务端将自己的数字证书（公钥，证书失效时间等）发送给客户端，客户端收到验证通过会随机生成一个密钥用服务端的公钥进行加密然后发送给服务端，服务端收到加密密钥用自己留存的私钥进行非对称解密，解密后得到客户端的密钥。服务端使用客户端密钥对数据进行加密传输给客户端，客户端用自己的密钥对传回的数据进行解密得到需要的数据

数据传输使用的是客户端的密钥对进行加密，这里是单向SSL过程，如果是双向的话客户端也要有自己的证书

## 27.在docker容器的生命周期，一个docker容器从产生到运行部署大致分为几个状态：

Dockerfile

用于创建image镜像的模板文件，出于管理和安全的考虑，docker官方建议所有的镜像文件应该由dockerfile来创建，而当前不少用户把docker当虚拟机来使用，甚至容器中安装SSH，从安全的角度，这是不恰当的。

Image

镜像文件，对比PC端的概念，我们可以把它理解为服务器端的可执行软件包。一旦打包生成，如存在安全问题，那这些问题也被一并打包，最后导致安全事件。

Container

运行起来的image文件就是容器了，从外来看就是一个应用，可对外提供服务了。

所以不难发现，docker容器的生命周期，就是一个镜像文件从产生、运行到停止的过程。