# Docker

## 为什么要使用docker，简单介绍一下docker

Docker 是一个开源的容器引擎，可以轻松的为任何应用程序，创建一个轻量级的，可移植的，自己自足的容器镜像。解决了软件程序跨环境/平台的迁移问题。

传统的软件更新发布部署效率低下，过程非常繁琐需要大量人工维护。

开发，测试，运维环境难以一致。

不同环境之间迁移数据成功太高。

有了docker可以解决传统软件程序上线，测试，维护所带来的大部分问题。特别是有了微服务和DevOps概念之后，docker更是成为了必不可少的基层环境部署工具。

Docker构建容易部署简单，隔离应用服务解耦，自动测试，快速部署，自动部署，持续部署。

### 为什么把docker当成虚拟机来用

虚拟化的核心是对宿主机资源的抽象，目的是在一个宿主机上运行多个系统或者应用，从而提高系统资源利用率。

虚拟化分为很多种，而docker所代表的是容器虚拟化技术，属于系统级虚拟化，通过系统内核来创建多个虚拟的操作系统来隔离不同的进程，这个虚拟的系统中只存在软件工作所需要的库资源和设置。

传统的硬件级虚拟化技术，是模拟出一套硬件，在其上运行一个完整的系统。容器虚拟化以共享kernel的方式实现，直接运行在宿主机之上的一个进程，几乎没有性能损耗。

这也是为什么docker快的原因，因为一个容器的增删改全是由主机内核直接操作。

## docker 组件

C/S 架构，包含如下:

Docker-client 客户端 //客户端执行docker提供的各种命令

Docker-server服务端 //服务端运行着容器上所有的镜像容器，通过守护进程启停。

Docker-daemon 守护进程 //服务组件，是docker核心的后台进程，负责docker的启停，

加载配置项，响应客户端发送的指令。

images 镜像 //用于创建docker容器的模板。

Registry 存储仓库 //用来保存镜像，代码仓库。

Container 容器 //在docker上运行的一组或一个应用。

## docker镜像组成原理

Docker 镜像是由特殊的文件系统叠加而成。

1.最底层是bootfs，并使用宿主机的bootfs，容器与宿主机间内核通用

2.第二层是rootfs文件系统，称为 base images，也就是rootfs基础镜像

3.再往上可以叠加其他的镜像文件。可以是自己的软件环境。

bootfs：包含bootloader(引导加载程序) 和 kernel(内核)组成

rootfs：root文件系统，包含的就是典型的linux 系统中 /dev/ /etc/ /proc等标准目录或文件。

每一层的镜像都有一个指向父层的指针，如果没有这个指针说明它在最底层。

#### 3-1.为什么不同程序镜像的大小不一样

拿 centos 和 tomcat举例

centos中的ISO镜像包含了bootfs 和rootfs，而docker中的centos镜像会复用真实机的bootfs，只有rootfs和其他镜像层，所以大小只有200M

由于docker中的镜像是分层的，tomcat本身虽然只有70mb，但是他需要很多依赖的父镜像以及基础镜像，所以整个对外暴露的tomcat镜像大小是500MB

#### 3-2.联合文件系统

联合文件系统是一种分层的轻量级文件系统，它可以把多个目录内容联合挂载到同一目录下，从而形成一个单一的文件系统，这种特性可以让使用者像是使用一个目录一样使用联合文件系统。

对于docker 来说，联合文件系统就是镜像和容器的基础。联合文件系统使得docker把镜像做成了分层结构。使得docker的每一层数据都能被共享，节省大量的存储空间。

Docker底层所支持的文件系统共有三种：AUDFS\Devicemapper\overlay2

Docker自18版本之后，默认使用overlay2作为文件存储驱动。

#### 3-3、dockerfile 指令

只记录几个关键的

From //指定基础镜像

Run //执行/bin/sh 命令

Volume //指定managed数据卷，将物理机目录挂载到容器镜像中

Cmd //启动容器时执行的命令

Entrypoint //入口命令

Copy //build时会复制本地文件到容器中

Add //build时会添加文件到容器中

Env //环境变量

Workdir //进入容器后的默认工作

注意事项：

1. 当CMD 与 ENTRYPOINT 同时出现，entrypoint的处理优先级比cmd要高，如果同时出现会先执行 entrypoint。且cmd的命令会变成字符串直接输出。
2. 当设置ENV时，直接调用不会生效，需要手动shell解析 /bin/sh -c 才会生效。
3. CMD 和 entrypoint 最好使用 exec 格式，RUN 命令使用 shell 格式，没有为什么
4. 如果一个 dockerfile 中出现多条 From，在编排时会为每一个from添加一个id号。默认从0开始。

如果第二个，第三个from的镜像，想要从第一个上传from镜像中获取数据，可以通过 --from=0 的方式，来指定fromid。

1. Volume 指定的数据卷一定是 managed 类型的。

#### 3-4、缓存机制

官网也没有给出具体的缓存原理

构建镜像时，docker会逐步执行dockerfile中的指令。在检查每条指令时，Docker会在其缓存中寻找一个可以重用的现有镜像，而不是创建一个新的镜像。

如果构建镜像时有 run，copy，add 这三个指令，会创建临时的中间镜像，会检查它们各自镜像中的文件内容，并为每一个文件内容生成一个校验和。在缓存查找期间，将校验和与现有的镜像校验和做一个对比，如果文件中的内容发生了更改，则缓存无效。

## 容器的运行原理

容器的定义和镜像几乎一样，也是一堆镜像层的统一视角，唯一区别在于容器是可写的。

当执行 docker create 或者 docker start 命令之后，就会在镜像的基础上运行一个可写层，这个可写层是运行在CPU上的一个进程。有多种不同的状态，比如运行状态，退出状态。

当真正 docker run 运行容器后就会进入运行状态。

同一个镜像可以运行多个容器，容器启动之后互相是隔离的，针对一个容器的更改只会局限于它本身。

## Docker 容器状态切换

Create(创建) ---> start(开启) ---> running (运行) ---> stoped(停止) ---> distory(销毁)

| |

pause(挂起) Restart(重启)

| |

Unpause(恢复挂起) running

create 创建容器 ，会显示 created 状态，经过start命令后到达 start状态，才能runing(up)

run 运行容器，会经过start状态，到达 running(up)

rm 删除容器，到达 distory 毁灭状态，直接删除

stop 停止，会到达 stopd exited 状态，停滞状态。

kill 删除，会先通过 stop 在进入到 stopd exited 状态，停滞状态。

restart 重启，会通过 start 重新到达 running(up)

pause 挂起，会到达 paused挂起状态，通过 unpause重新恢复到 running(up) 状态

特殊状态：

OCM：容器报错，内存不够，CPU不足，导致的程序结束

## Namespace 隔离

Namespace 属于linux内核的隔离机制，保证linux两个进程之间不会出现冲突，互不影响。

在linux的/proc 中包含每一个pid进程文件，这些进程文件中有linux原生进程，也有docker容器启动的进程。每个pid 文件中有一个ns目录，这个目录下包含了6种隔离机制。

六种隔离机制

PID：进程编号

UTS：主机名以及域名

IPC：信号量，消息队列，共享内存

NET：网路设备，网络协议栈(网关)，端口

MNT：挂载点，文件系统

Use：用户和组

进入docker运行的容器，在proc下也有这样的目录，包含着docker自己的进程文件。

因为容器依赖与镜像，而镜像的组成又是多个联合文件系统，所以容器内的数据其实也是挂载了一部分宿主机的数据，类似于复制。

举例说明：

1.PID： 首先是进程隔离，在宿主机上运行容器其实也是一个进程，比如是2345，到了docker内部就是1，经过比对之后可以发现 2345 与 1 的隔离编号完全相同。

2.MNT：文件隔离，docker内部执行ls 与 宿主机执行ls看到的结果是相同的，进入/var/lib/docker/overlay2/ID/merged 可以看到该目录里面的内容与容器目录是一样的，在这个目录下随便创建点什么，会同步挂载到容器中，这就是文件隔离。

3.Net：网络隔离是最特殊的，它不在程序内部隔离，而是需要在宿主机上创建一个独立的空间来实现隔离，创建网络空间之后，容器内网卡两块网卡间通过一个 @if 符号相关联。

1. Use：如果不指定隔离用户，进入容器后会是root用户，容器内创建的用户，宿主机上没有。
2. IPC：共享内存ipcs，在容器中创建的ipcs队列，在宿主机是看不见的

## 容器在隔离之后遇到的问题？docker提权

因为docker运行的容器中1号进程是 /bin/bash，它没有执行systemctl系统命令的权限，所以如果想要执行 systemctl 需要进行提权。提权的方式有两种：

第一种方法:

--privileged 提权命令，打开/sbin/init ，此时进入容器运行的 1号进程就是 /sbin/init。

但是这样做是不对的，因为/sbin/init 是root用户的权限，如果容器和宿主机都是root权限，就会出现一个权限占用的问题。

第二种方法：

通过启动命令来启动程序，找到一个服务systemd 启动程序时的Execstart 文件，比如apache服务，ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND。其实这个Execstart文件也是通过绝对路径执行一个服务的启动文件，如果知道该服务的启动文件在哪里，也可以进入该服务的启动文件目录手动启动。

## 八、Cgroup 资源限制

Namespace解决环节隔离问题，Cgroup解决及计算机资源使用上的隔离问题。

Cgroup 也是linux内核的一个功能，主要用来限制，控制，分离一个进程组的资源（CPU,内存，硬盘输入输出）。

Cgroup实现的功能

进程组的 Memory 内存容量使用上限。

进程组的 CPU 优先控制，为进程分配cpu的优先级。

进程组的 Disk-I/O 硬盘控制，bps,iops 控制每秒读写数据量，读写次数。

Linux将cgroup 以系统文件的方式都实现好了。

Docker中使用Cgroup限制资源也是需要指定不同的资源限制参数，通过内核进行系统调用。

将docker运行的进程ID加入到Cgroup-Tasks任务中，再分配给具体的资源控制组来实现。

## 九、数据卷持久化

#### 持久化小问题

1、docker 容器删除后，在容器中产生的数据还在吗？

****—答：如果设置了持久化数据卷，则产生的数据会在数据卷中保存。****

2、docker 容器和外部机器可以直接交换文件吗？

****—答：通过数据卷挂载可以进行交换文件，共享文件。****

3、容器之间想要进行数据交互？

****—答：一个数据卷可以被多个容器挂载，实现数据交互。****

1. 数据卷权限

****—答：ro，rw。 Bind 类型默认支持ro，rw。Managed仅支持rw。****

1. 一个容器可以同时使用bind 和 managed 吗？

****—答：可以，因为它们的挂载点并不冲突****

#### 都有哪些持久化的方式

****针对单一容器：****

bind mount：将物理机目录 挂载到 容器目录

managed volume：将容器目录 挂载到 物理机目录/docker/\_data 目录当中

tmpfs：容器目录挂载至内存当中

****针对集群容器：****

volume container：容器共享，所有容器挂载共享容器的目录

volume nfs：挂载nfs目录，创建容器共享，所有容器挂载共享容器目录

data-packed volume container：创建dockerfile实现镜像迁移，通过镜像创建容器共享，所有容器挂载共享容器目录

**注意细节：**

1.bind mount 这种数据卷的挂载方法会将 容器内初始内容全部清空。此数据卷挂载方法只适用于 容器创建时使用。一旦容器已经运行一段时间后将不再适用。

2.使用managed volume 必须要提前使用 docker volume创建出一个 managed 卷

3.使用 tmfps 需要限制容量大小，如果容器删除则里面数据立刻清空。

#### 什么是容器共享

**容器共享是一种特殊的容器，而不是数据卷。**

**凡是使用容器共享的容器，比如nginx，其挂载的数据都是一样的。假如开了3台nginx，这3个nginx的挂载页面都是一样的。**

#### 如何进行数据迁移

通过Dockefile 实现数据迁移

1. 通过NFS进行创建持久卷，将要迁移的数据目录打包，并放到 managed volume中。
2. 将要迁移的镜像进行commit形成一个新的镜像，或者本来就有一个新的镜像。
3. 使用Dockerfile 将打包好的的备份数据文件，ADD拷贝到新的容器目录上。
4. 可以手动进入容器解压数据，也可以在dockerfile中通过 volume指令将存在备份数据的目录共享到本地，然后解压。

#### 如何进行数据备份

通过共享容器实现备份

1.使用NFS将备份服务器的硬盘挂载到容器目录下，生成共享目录。

2.使用docker volume 命令创建一个 managed持久卷，然后根据这个持久卷创建一个共享容器。

3.通过 --volumes-from 将数据卷挂载到要备份的主机上。

4.接着通过tar 命令，将容器中所有想要备份的目录制成 tar包。然后放到managed目录上，创建一个定时任务，到一定时间就备份一次。

## Docker网络

分为单主机网络和跨主机网络

#### 1.Docker 网络基础

Docker 的网络隔离通过namespace实现，网络的转发通过 iptables实现。

Docker 的网络需要在宿主机上创建 虚拟网卡，DNS，路由表，通过nemespace可以自动实现，这种网卡也是容器的内部网络栈。

Docker的默认网络类型bridge，网卡是docker0 网桥模式连接宿主机。所有未指定网络类型容器都会连接到这块docker0上。在容器中网络通过interface 接口来连接docker0。

#### 2.Docker 网络模型？有什么局限？

**单主机通信模型(none,host,joine,bridge)：**

1、none ： 封闭模式，需要安全性较高，且不需要联网的环境下使用，如：生成随机码，生成二维码。

2、host ：连接docker与主机共享的网络(完全同步)

优点：性能好，对网络性能要求较高可以使用host，但是会导致灵活性降低。比如要考虑端口冲突问题，Docker host 上已经使用的端口就不能再用了

3、joine 容器：容器和容器使用完全相同的网络栈，两个容器使用相同的网络，相同的网卡。

1. bridge：应用最广泛也是默认的 bridge 网络，docker 0，由真实机与容器通过两块网卡形成桥接网络进行通信。

**多主机网络模型（overlay、macvlan、flannel）**

#### 3.Docker跨主机实现容器间通信？

Docker 原生的跨主机通信：overlay 和 macvlan

第三方通信方案：flannel，weave

Docker 通过libnetwork 以及 CNM技术将上述的各种方案与docker集成在一起。Libnetwork 是 docker 容器网络库，最核心的内容是其定义的Container Network Model，这个模型对容器网络进行了抽象，由三类组件组成：

1.SandBox

Sandbox 是容器的网络栈，包含容器的interface，路由表和DNS设置。Network的namespace就是通过sandbox实现的。Sandbox 可以包含来自不同Network的Endpoint，也就是说Sandbox 将一个容器与另外一个容器通过 Namespace进行隔离，一个容器包含一个sandbox，每个sandbox可以有多个endopint连接不同的网络。

1. Endpoint

Endpoint的作用是将Sandbox 接入Network，Endpoint的典型实现的 Vethpair。一个Endpoint只能属于一个网络。

1. Network

一个网络中包含多个 Endpoint，同一网络下的Endpoint可以互相通信。

什么是覆盖网络：

覆盖网络是运行在应用层的网络，不依靠ip地址来传递消息，而是采用一种映射机制，通过ip地址映射来进行资源定位。将TCP数据包，装在另一种网络包里面进行路由转发和通信。

##### Overlay

Overlay的组成：

Vxlan ：负责通讯，将二层数据封装到UDP层进行传输，提供了和vlan相同二层以太网服务，但是拥有更强的扩展性和灵活性。

key-value：非关系型数据库，比如：consul etcd zookeeper 保存如下信息：

sandbox : 容器的网络栈 interface 路由表 DNS

endpoint：将sandbox接入到 network veth peer

network：包含一组sandbox和endpoint。相同的network可以通讯

Overlay 主要用在 swarm集群中，属于swarm默认的分布式网络。当初始化集群或将docker主机加入集群时，会在该Docker主机上创建两个新网络：ingress，docker\_gwbridge

Ingress：覆盖网络，用于处理与集群服务相关的数据流量。如果创建群集服务时没有定义具体的连接网络，默认会将它连接到 ingress网络中。

Docker\_gwbridge 桥接网络，连接每一个docker启动的容器，替代了docker0

##### Macvlan

不能在云端主机上使用，只能在物理机使用，甚至虚拟机也不可以，因为它是通过vlanMAC地址进行通信的。

##### Flannel

flannel使用etcd来维护分配的子网到实际的ip地址之间的映射。对于数据路径，flannel 使用udp来封装ip数据报，转发到远程主机。默认的节点通讯方式是udp。

Flannel 如何分配网络实现通信：

首先，flannel利用Kubernetes API或者etcd用于存储整个集群的网络配置，其中最主要的内容为设置集群的网络地址空间。例如，设定整个集群内所有容器的IP都取自网段“10.1.0.0/16”。

数据从源容器中发出后，经由所在主机的docker0虚拟网卡转发到flannel0虚拟网卡，这是个P2P的虚拟网卡，flanneld服务监听在网卡的另外一端。

Flannel通过Etcd服务维护了一张节点间的路由表，该张表里保存了各个节点主机的子网网段信息。

源主机的flanneld服务将原本的数据内容UDP封装后根据自己的路由表投递给目的节点的flanneld服务，数据到达以后被解包，然后直接进入目的节点的flannel0虚拟网卡，然后被转发到目的主机的docker0虚拟网卡，最后就像本机容器通信一样的由docker0路由到达目标容器

1. **UDP封装是怎么回事？**

在UDP的数据内容部分其实是另一个ICMP（也就是ping命令）的数据包。原始数据是在起始节点的Flannel服务上进行UDP封装的，投递到目的节点后就被另一端的Flannel服务还原成了原始的数据包，两边的Docker服务都感觉不到这个过程的存在。

1. **为什么每个节点上的Docker会使用不同的IP地址段？**

这个事情看起来很诡异，但真相十分简单。其实只是单纯的因为Flannel通过Etcd分配了每个节点可用的IP地址段后，偷偷的修改了Docker的启动参数。  
 在运行了Flannel服务的节点上可以查看到Docker服务进程运行参数（ps aux|grep docker|grep "bip"），例如“--bip=182.48.25.1/24”这个参数，它限制了所在节点容器获得的IP范围。这个IP范围是由Flannel自动分配的，由Flannel通过保存在Etcd服务中的记录确保它们不会重复。

1. **为什么在发送节点上的数据会从docker0路由到flannel0虚拟网卡，在目的节点会从flannel0路由到docker0虚拟网卡？**

例如现在有一个数据包要从IP为172.17.18.2的容器发到IP为172.17.46.2的容器。根据数据发送节点的路由表，它只与172.17.0.0/16匹配这条记录匹配，因此数据从docker0出来以后就被投递到了flannel0。因为它们属于同一个网段。

同理在目标节点，由于投递的地址是一个容器，因此目的地址一定会落在docker0对于的172.17.46.0/24这个记录上，自然的被投递到了docker0网卡。

##### ping包转换流程：

1. 发送icmp包到www.baidu.com
2. 通过网桥连接到docker0，docker0收到ping包，发现是访问外网的，则通过NAT转换端口号，到ens33进行处理。
3. 此时的源IP地址换成了 ens33的ip和port。然后与外网通信。

## 十一、简单讲下compose和Swarm

使用Docker compose可以用YAML文件来定义一组需要启动的容器，以及容器运行时的属性。

docker-compose用来对这一组容器进行操作。

docker swarm 原生的Docker集群管理工具，依赖docker本身，很多重要功能依赖团队二次开发。

社区不够活跃，一般公司生产环境会选择k8s，个人项目或者容器数量较少可选swarm,只需要docker

即可完成，相对较轻。