Docker 简介

1、集装箱的发展：从不同的运输工具之间流转的时候很麻烦，有了集装箱之后流转方便，国际间也是通行的。“标准化”

2、目前系统开发的麻烦：在开发人员电脑上开发的系统，跑到其他人电脑上或者服务器上的时候需要重新安装软件、安装开发包、系统配置，不同系统上可能操作还不一样；A 电脑运行没问题，B 上运行有问题“我这里没问题呀”；有的公司服务器比较空闲，只运行一个应用服务器太闲，但是运行多个应用又怕互相干扰。

可以用虚拟机，但是性能低，硬件浪费严重。

3、Docker 技术的特点：在操作系统上分出多个个独立的区域（称作“容器”(Container)）。各个容器之间互相“基本隔离”，每个容器可以单独有自己的系统配置、安装的软件、安装的开发包，各个容器之间的软件“基本”不会互相干扰。Docker 上的配置好的容器内容随意随意移动到其他计算机中运行。因此可以很好的解决上面的问题。Docker 性能损耗比虚拟机低很多，都是进程跑在原生操作性系统下，基本没有性能损耗。

Docker 还可以更好的满足对于可伸缩性的要求，以“双十一”举例 Docker 的应用：按需自动扩容，自动启动多个服务器、创建多个容器运行更多集群服务器。京东：2016 年就有 15 万个 Docker。

4、Docker 部署系统的要求：可以“即抛”，不保存状态数据。

5、 Docker 最初是基于 Linux 的 LXC 技术，现在在 Windows、Mac 下也有版本，最好用的还是 Linux。

docker 不支持 32 位系统。Linux3.8 内核以上的操作系统都支持 docker，建议使用 ubuntu。

6、Docker 的几个概念：

1) 镜像（Image）：像软件安装包。我们可以把镜像拉(pull)下来，增加我们的更改，然后发布(push)成新的镜像，别人也可以下载我们发布的镜像，然后继续更改，继续发布；
2) 容器（Container）：镜像安装后运行的实例；同一个镜像可以在同一台服务器上安装成多个容器，同一个镜像可以同时装在多个不同的服务器上。
3) 镜像 layer：对于一个镜像的多次修改 push 就会有多个 layer，一个镜像通常由多个 layer 组成。

7、docker 的命令、参数很多、组合很多，不要去死记硬背命令、参数，要灵活应用。

Docker 安装

docker 安装步骤（目前使用 docker 指令的时候不要忘了 sudo 执行，否则……）：

1、 Ubuntu 中：sudo apt-get install docker.io
2、 执行一下 sudo docker version 看看是否安装成功；
3、 sudo docker info 可以查看 docker 的状态信息
4、 比如想下载 busybox 的镜像（一个简单的 linux 发行版），就执行 sudo docker pull busybox（不要忘了 sudo）。
5、 Docker 相当于 VirtualBox，镜像相当于安装盘，容器相当于安装后的可以运行的虚拟机。
6、 下载速度慢怎么办？加速器（mirror）和镜像市场的区别：一个是代理服务器，最终还是访问官方网站，和官网一致，一个是私服，不和官网一致。
7、 如果下载有问题（别忘了 sudo），可能是网络访问不了国外的 docker 服务器或者网速慢，就要配置镜像，可以用阿里云或者 DaoCloud 等的加速器。以阿里云为例访问：https://cr.console.aliyun.com/ 注册账号后，在【加速器】中获取“专属加速器地址”

按照下面的操作文档配置即可，不要泄露给别人。（使用 sudo docker -v 查看 docker 版本）那个 tee 指令就是用来写入文件的，直接用 vi 写文件也行（一定注意不能拼错了）。国内的加速器的问题仍然不靠谱，如果用了加速器还不行，那就翻墙吧。我用的最新版用 sudo docker info 看不到加速器的信息，有人提 PR 了。

8、 从国内镜像市场下载（https://hub.daocloud.io/等），使用“拉取镜像”中的命令 pull 就可以了。
9、 然后启动 busybox 镜像，并且进入 busybox 的 shell：sudo docker run -it busybox（也可以直接执行 sudo docker run -it busybox，这样如果发现镜像没下载的话会自动 pull 镜像）。后续的操作就是 busybox 这个容器环境中运行了。后续执行的时候一定要区分是在容器

中执行还是在主机环境中执行（通过命令前面的标志来区分是主机中还是容器中）。

10、 容器中执行 ls、pwd 看一下，确认是容器环境，和主机环境的文件不一样；

11、 容器中执行 vi 建个文件保存一下；

12、 容器中执行 apt-get install mysql，执行失败，因为 busybox 这个镜像中没有安装 apt-get

13、 可以到 https://hub.docker.com 上搜索其他的镜像（网速不好的话还是建议翻墙，镜像解决不了上不了 https://hub.docker.com 的问题）

14、 exit 退出 docker 容器中的操作系统

配置每次 docker 免 sudo

每次执行 docker 指令都要 sudo 很麻烦，只要把我们操作的用户加入 docker 用户组就不用那么麻烦。注意要回到主机环境运行。如下：

1） sudo groupadd docker 确保创建了 docker 用户组，应该是显示“docker 用户组已存在”

2） sudo gpasswd -a 当前用户名 docker ：把当前用户名加入 docker 组

3） sudo service docker restart 重启 docker 服务

4） 注销：如果是服务器版（直接 shell 进入）就执行 logout；如果是桌面版，就要在右上角点电源图标选择【注销】

5） 重新登录即可。

docker 常用命令

1、 docker 的帮助文档查看方式： docker --help，文档中一个横线的-是简写参数，两个横线--的是非简写，效果一样；查看子命令的帮助 docker run –help

2、 使用“docker run –it 镜像名字”方式进入容器，一旦退出容器就停止了（相当于关机）；再次执行 docker run 的时候是开启一个新的容器实例。一个镜像（image）可以启动多个容器。试着在不同的容器中创建不同的文件夹就知道。默认会随机给一个名字，可以给一个—name（两个横线）参数设定一个名字：docker run –it –name dev1 busybox

3、 可以使用 docker ps -a 查看所有容器（包含已经停止的。如果不加-a 就只显示运行中的容器），其中有一个 containerId 叫做“容器 id”，用来区分不同的容器；容器 id 是有长 id 和简化 id，都能用。

-q 参数就是只列出容器的 id

可以使用 docker rm $(docker ps -a -q)删除所有容器。

-f 可以添加过滤条件，比如“docker ps -a -f=ancestor=busybox”就是只列出 busybox 这个镜像的容器。因此 docker rm $(docker ps -a -q -f=ancestor=busybox) 就是删除所有 busybox 的容器

每次 docker run 都会启动一个新的镜像，可以通过“docker start 容器 id”来启动已经停止的容器；

可以用“docker run -d 镜像名”的方式在后台运行容器，返回的 id 就是容器的 id，比如

“docker run -d busybox”，如果镜像运行后什么也不干、也不启动后台进程，这个后台容器也就停止了。

4、 可以配置容器运行后台进程。nginx 镜像是一个带后台运行的服务器，我测们试一下：docker run -d nginx" 。busybox 什么的由于执行后没啥事情就“秒退”了，所以不能用 busybox 等测试。

是在后台容器中执行，所以我们看不到打印。再 docker ps -a 试试看。

5、 附加到容器中：docker attach 容器 id；如果容器已经停止了，则首先需要先 docker start 容器 id 启动，再 attach。

6、 docker images 列出所有安装的镜像

7、 “sudo docker rmi 镜像的 ImageId”删除指定的镜像。注意是 rmi（不要丢了代码删除 Image 镜像的 i，rm 是删除容器）。当然删除镜像之前必须把所有这个镜像的容器删除，否则会报错“image is used by stopped container”，比如 docker stop $(docker ps -a -q -f=ancestor=busybox)然后 docker rm $(docker ps -a -q -f=ancestor=busybox)。可以“sudo docker rmi 镜像的 ImageId -f”强制删除，但是容器还在，会有隐患，因此不要-f，还是先删容器，再删镜像。

8、 总结一下常用用法：

a) **“docker run –it 镜像名”以交互模式启动新容器；通过--name 参数设定名字；**

b) **“docker run –d 镜像名” 以后台模式模式启动新容器；**

c) **docker ps –a 查看所有的容器（包含已经停止的）**

d) **如果容器已经停止则“docker start 容器 id”启动容器**

e) **使用“docker attach 容器 id”连接上容器，已经停止的容器无法连接**

9、 可以试验一下 ubuntu 的镜像。第一次很慢。

docker 安装.net core 镜像

1） docker run -it microsoft/dotnet 就会安装带.net core 的 linux 镜像。第一次很慢。

2） 后面就是常规的 dotnet new 创建项目，运行等了，先建立一个控制台程序测试

3） 再建一个 mvc 项目。但是容器中的网络程序，外部是访问不了的，要做端口映射

4） 在启动容器的时候执行：docker run -p ip:hostPort:containerPort 镜像名 这样就把主机的 hostPort 端口映射到容器的 containerPort 上，也就是外部用户通过 hostPort 端口可以访问容器中监听 containerPort 端口的程序。如果省略 ip 则表示绑定主机的所有网卡（0.0.0.0），一定要确保主机的端口没有被占用。还要设置 UseUrls 才可以（下面）。

5） 执行如下的 docker run -it -p :81:5000 microsoft/dotnet 进行端口映射，然后进入容器命令行。

6） 可以通过 docker ps -a 查看端口映射信息

7） 由于 microsoft/dotnet 这个镜像里没有 vi（故意的），所以如果想在容器里写代码，就要自己安装，幸好安装了 apt-get，首先要“apt-get update”更新一下 apt 的源列表信息，然后“apt-get install vim”安装 vim（vi 高级版）。

8） 在容器中/home 下创建一个文件夹，然后 dotnet new mvc 创建一个 mvc 项目，编辑 Program.cs。加入 UseUrls("http://*.5000")，这表示可以监听来自容器外部的请求；

```
namespace web1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] arg
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .UseUrls("http://*:5000")
                .Build();
    }
}
```

9） 然后 restore、run。

10） 在容器外就可以用 http://主机 ip:81 访问了，外网也可以访问

制作自己的镜像

1、为什么要做自己的镜像

如果用 docker 做一个开发用的虚拟机用，那么上面的足够了。但是如果用 docker 做服务器，则不能进入容器后进行操作。因为容器应该是“即抛的”，因此不应该在容器中编写代码，人家没有内置 vi 也是有道理的。怎么办？在外部写代码，调试通过后，发布，然后再创建镜像执行。同样一个镜像可能运行 N 多容器，不可能每个容器都单独配置。

应该制作镜像！

2、制作一个预装 nginx 的镜像

编写一个文件 Dockerfile（不能写错，包括大小写）：

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y nginx
EXPOSE 80
```

FROM 表示这个镜像是继承自那个镜像；RUN 为构建时候执行的指令；EXPOSE 80 表示容器中的服务“可以”暴露 80 端口（后续还要再 run 的时候再绑定一下）

然后执行 docker build -t yzk/nginx .（注意最后的点儿，前面还要有空格），表示根据当前目录下的 Dockerfile 构建一个镜像“yzk/nginx”，然后就可以：docker run -it yzk/nginx 这样使用这个容器了（没有提交到 DockerHub 的时候只能本机使用）

可以搭建自己公司的 DockerHub 私服。

3、制作包含网站的一个镜像

首先在主机上或者 windows 上编写.net core 项目，假如是 web1（别忘了 UseUrls()），然后发布，把发布后的文件放到主机的一个目录下，然后进入这个目录。然后在这个目录下编写 Dockerfile，内容是：

```
FROM microsoft/dotnet
COPY . /publish     #注意.和/之间有空格，表示把主机当前目录内容拷贝到镜像的/publish 目录
WORKDIR /publish    #设定工作目录
EXPOSE 5000/tcp     #暴露 5000 端口
CMD ["dotnet","web1.dll"]
```

然后运行构建：docker build -t yzk/web1 .（不要忘了结尾的.和之前的空格）

然后运行：docker run -d -p :81:5000 yzk/web1（确保没有被其他 docker 或者进程占用，如果其他 docker 占用，要使用 docker stop 先给他停止）

在主机浏览器中就可以 http://127.0.0.1:81 访问了。如果访问不了，看看是不是忘了 UseUrls()、EXPOSE 或者-p

正常开发还是在 windows 中开发，不需要每次都部署到 docker。

每次修改代码或者 Dockerfile 只要重新 build 就可以重新生成镜像，然后再 run（如果端口被占用，要把之前的容器 stop 掉）

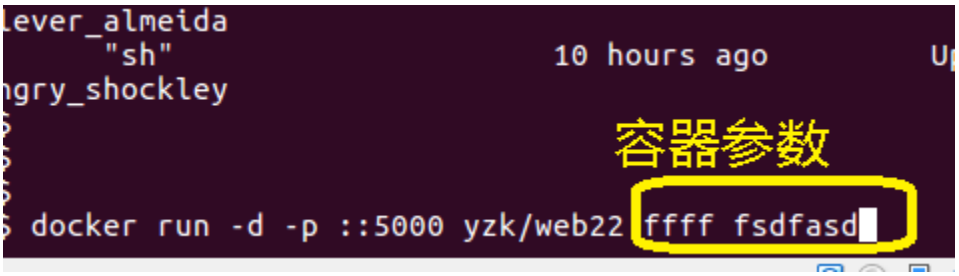
如果不指定主机端口，那么就会随机绑定一个没有被使用的端口“-p ::5000”，可以使用 docker port 命令查看到底被绑定到了哪个端口中。

4、

CMD、RUN 的区别：RUN 是镜像 build 的时候就执行的，用来预装软件和做配置使用，很显然不能 RUN ["dotnet","web1.dll"]；CMD 是镜像启动的时候执行的命令，一般执行服务器启动之类的命令，每个 Dockerfile 只能有一条 CMD 命令；如果指定了多条命令，只有最后一条会被执行。

5、

除了 CMD、RUN 之外，还有一个 ENTRYPOINT 指令，也是只能指定一条，一般不和 CMD 一起用（可以让 CMD 给 ENTRYPOINT 传参数“如果存在 ENTRYPOINT 和 CMD，那么 CMD 就是 ENTRYPOINT 的参数，如果没有 ENTRYPOINT，则 CMD 就是默认执行指令”，意义不大）。



ENTRYPOINT 也是容器启动时候执行的指令，和 CMD 的区别是：如果没有指定容器参数，那么 CMD 和 ENTRYPOINT 是一样的。启动容器时候的指定容器参数（不是-it 那些，而是镜像名后面那些）的影响不同。如果指定了容器参数，那么 CMD 指定的命令被 docker run 传递的容器参数覆盖，而 ENTRYPOINT 会把容器名后面的所有内容都当成参数传递给 ENTRYPOINT 指定的命令（不会对命令覆盖）。

比如 Dockerfile 这样写镜像 yzk/b1

```
FROM busybox
CMD [“/bin/uname”]
```

那么如果 docker run yzk/b1 运行，会显示 Linux 版本；如果执行 docker run yzk/b1 –a，那么会提示不存在-a 命令；如果执行 docker run /yzk/b1 /bin/whoami 就会执行 whoami。

而 Dockerfile 这样写镜像 yzk/b1

```
FROM busybox
ENTRYPOINT[“/bin/uname”]
```

那么如果 docker run yzk/b1 运行，会显示 Linux 版本；如果执行 docker run yzk/b1 –a，那么会执行“uname -a”；如果执行 docker run /yzk/b1 /bin/whoami 就会执行“uname /bin/whoami”提示参数错误。

ENTRYPOINT 用于把容器打包成一个好像可执行程序的东西，比如把 mysqlclient 打包成一个镜像。

6、总结 RUN、CMD、ENTRYPOINT 区别：

RUN 是构建镜像包时候执行的指令，通常用于安装软件、修改配置等初始化的代码，可以执行多条；

CMD 相当于镜像的默认开机指令，只能指定一条 CMD，容器运行参数可以覆盖 CMD；

ENTRYPOINT 用于把镜像打造成可执行程序，容器运行参数作为可执行程序的参数。

7、封装一个 MySql 的镜像

```
FROM ubuntu
RUN echo 'mysql-server mysql-server/root_password password root'|debconf-set-selections
RUN echo 'mysql-server mysql-server/root_password_again password root'|debconf-set-selections
RUN apt-get update
RUN apt-get install -y mysql-server
RUN /etc/init.d/mysql restart &&\
  mysql -uroot -proot -e "grant all privileges on *.* to 'root'@'%' identified by 'root'" &&\
  mysql -uroot -proot -e "show databases;"
EXPOSE 3306
CMD ["/etc/init.d/mysql","restart"]
```

注意：在 Dockerfile 中 RUN echo 'mysql-server mysql-server/root_password password root'|debconf-set-selections 不能写成 debconf-set-selections<<<’ mysql-server mysql-server/root_password password root’

在 Dockfile 中 RUN /etc/init.d/mysql restart 中的这三行不能拆分成三行 RUN。

如果没有命令修改镜像程序的配置，那么可以写好配置文件，然后使用 COPY 指令拷贝覆盖配置文件。

mysql.cnf

```
COPY ./mysql.cnf  /etc/mysql/mysql.cnf
```