# DockerNotes



**BV1og4y1q7M4**

## 基本概念

镜像(Image)：相当于模板，提供模板创建容器。Tomcat镜像->Run->tomcat容器

容器(container)：提供镜像来创建，可以理解为简单的linux系统

仓库(repository)：存放镜像的地方，分为公有和私有仓库。

Docker是C/S架构的系统，需要使用Client去连接Docker服务。

## Docker安装

首先设置安装仓库：

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

更新yum软件包索引：

```
yum makecache fast
```

安装 `docker-ce` 社区版

```
sudo yum install docker-ce docker-ce-cli containerd.io
```

启动docker服务：

```
sudo systemctl start docker
```

判断是否安装成功：

```
docker version
```

运行镜像：

```
docker run hello-world
```

查看镜像：

```
$ docker images
REPOSITORY     TAG          IMAGE ID          CREATED           SIZE
hello-world    latest     d1165f221234    4 months ago    13.3kB
```

> Windows下安装docker如果运行出现错误使用 `netsh winsock reset` 命令进行尝试。以及docker的C盘迁移 **LINK**

# WSL迁移

1. 首先关闭docker
2. 关闭所有发行版：
   `wsl --shutdown`
3. 将docker-desktop-data导出到D:\SoftwareData\wsl\docker-desktop-data\docker-desktop-data.tar（注意，原有的docker images不会一起导出）
   `wsl --export docker-desktop-data D:\SoftwareData\wsl\docker-desktop-data\docker-desktop-data.tar`
4. 注销docker-desktop-data：
   `wsl --unregister docker-desktop-data`
5. 重新导入docker-desktop-data到要存放的文件夹：D:\SoftwareData\wsl\docker-desktop-data\：
   `wsl --import docker-desktop-data D:\SoftwareData\wsl\docker-desktop-data\ D:\SoftwareData\wsl\docker-desktop-data\docker-desktop-data.tar --version 2`

# 配置阿里云镜像加速

登陆阿里云控制台=》容器镜像服务=》镜像加速器

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://*.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

> Windows版直接图标Settings的json里加入网址即可。

# Docker命令

## 镜像命令

- `docker images [-a -q]`：查看镜像
- `docker search mysql`：搜索某个镜像

  `docker search mysql --filter=STARS>3000`
- `docker pull pkg`：下载镜像
- `docker rmi [IMAGE ID, IMAGE ID]`：删除镜像

## 容器命令

> 有了镜像才能创建容器，这里使用CentOS镜像

下载CentOS：

```
docker pull centos
```

启动容器命令：

```
docker run [--args] image
docker run --name <container_name> -e <env_var> -d   -p <host_port:con_port> <image>:<tag>
# 参数说明
--name="name"    容器名字 tomcat01 tomcat02 用来区分容器
-d               后台方式启动，相当于nohup
-it              使用交互式运行，进入容器查看内容
-p               指定容器端口 [8080:8080]主机端口，映射到容器端口
-P               随机指定端口(大写)
-e               设置环境变量
```

举例：`docker run --name psql_a -e POSTGRES_USER=root -e POSTGRES_PASSWORD=**** -p 5433:5432 -d postgres:13`

启动CentOS：

```
docker run -it centos /bin/bash
# =============
[root@7c52f9205593 /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr
var
# 镜像CentOS很多命令并不完善
```

列出所有运行的容器：

```
docker ps
docker ps -a        # 将运行过的程序也列出来
docker ps -n=2  # 显示最近创建的n个容器
docker ps -a -q # 只显示容器的ID
```

退出容器：

```
exit            # 停止容器并且退出
Ctrl + p + q     # 不停止容器并且退出
```

删除容器：

```
docker rm 容器ID
docker rm -f [Con ID] # 强制删除运行中的容器
```

启动重启停止杀死容器：

```
docker start [Con ID]
docker restart [Con ID]
docker stop [Con ID]
docker kill [Con ID]
```

**后台启动容器：**

```
docker run -d centos
# 如果使用docker ps -a会发现centos已经停止使用了
```

> 这是常见的问题：docker容器在后台使用，必须有一个前台应用，否则程序会自动停止。

# 其他命令

查看日志：

```
docker logs
```

查看容器内部的进程信息：

```
docker top [ID]
# ======================
C:\Users\admin>docker top b11b4ccb7004
UID                 PID                 PPID                C
        STIME               TTY                 TIME                CMD
root                1218                1197                0
        11:01               ?                   00:00:00            /bin/bash
```

查看镜像元数据：

```
docker inspect conID
```

▶ 详细信息

进入当前正在运行的容器：

```
# 方式一 相当于重新打开一个终端
docker exec -it [Con ID] /bin/bash
# 方式二 进入容器正在运行的终端
docker attach [Con ID]
```

从容器内拷贝到主机上

```
docker cp [ConId]:[Con Path] [Host Path]
```

查看docker容器CPU的情况：

```
docker stats
```

# Docker实例

## 安装nginx

```
docker search nginx
docker pull nginx
docker run -d --name nginx01 -p 3344:80 nginx
docker exec -it nginx01 /bin/bash
docker stop nginx01
```

## 安装tomcat

```
docker run -it --rm tomcat:9.0      # --rm 表示测试容器，用完即删容器（不删除镜像）
docker run -d --name tomcat01 -p 3345:8080 tomcat:9.0
docker exec -it tomcat01
```

## 部署es+kibana

```
# es 暴露端口很多并且十分耗内存
# es 的数据一般要放置到安全目录！ 需要挂载
# --net somenetwork

# docker network create somenetwork
docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node"
elasticsearch:tag
```

## 保存自己的镜像

> 相当于快照

```
docker commit -a="Author" -m="Some msg" <conId> <NewImgId>:<TAG>
```

# 容器数据卷

> 一种数据共享的技术，docker产生的数据会存储到本地。将容器上的目录挂载到本机上+容器间数据共享。

## 挂载：

方式一：

```
docker run --name <name> -v <hostpath>:<conpath> -p <hport>:<cport> <img>
```

MySQL的数据存放在其 `data` 目录下

## 具名挂载和匿名挂载

```
# 匿名挂载 ro->readonly rw->readwrite
docker run --name <name> -v <conpath>:<ro | rw> -p <hport>:<cport> <img>

# 具名挂载  此处name不是目录
docker run --name <name> -v <name>:<conpath> -p <hport>:<cport> <img>

# 查看所有卷的情况
docker volume ls

# 查看某个卷的所在目录：
docker volume inspect 090a2c068e303678c2384df0e5980a4be517c83c71e6e5a82d790f7baf1de531
```

```
[
    {
        "CreatedAt": "2021-07-22T10:03:50Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint":
"/var/lib/docker/volumes/090a2c068e303678c2384df0e5980a4be517c83c71e6e5a82d790f7baf1de531/_data",
        "Name": "090a2c068e303678c2384df0e5980a4be517c83c71e6e5a82d790f7baf1de531",
        "Options": null,
        "Scope": "local"
    }
]
```

## 初步构建dockerfile

> Dockerfile就是用来构建docker镜像的文件

```
FROM centos                    # 从那个镜像而来

VOLUME ["/home"]        # 设置挂载的目录，只写了容器内的目录为匿名挂载

CMD echo "======END======"      # 输出

CMD /bin/bash                  # 进入命令
```

根据dockerfile生成镜像：

```
docker build -f <dockerfile> -t <2mw/centos:tag> .
```

## 数据卷容器

> 多个容器共享数据

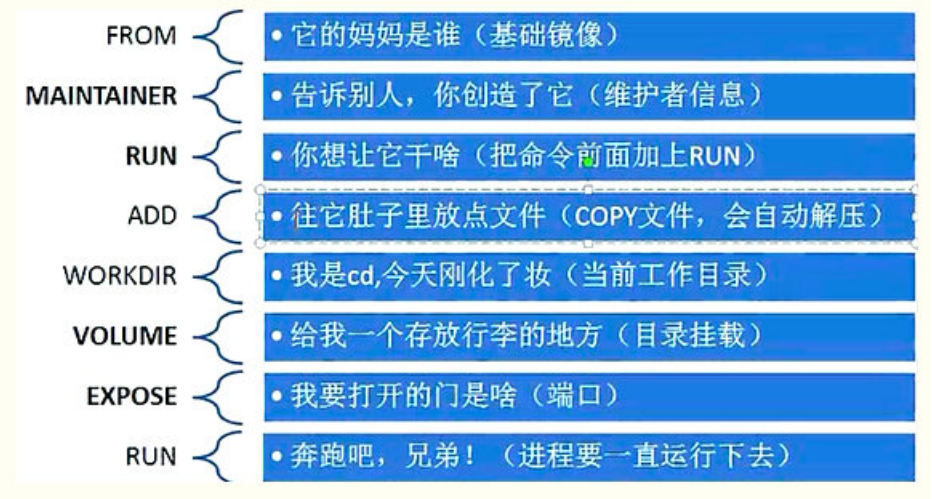
```
docker run --it --name centos02 --volumes-from docker01 <imgID>
# 使用 --volumes-from 来继承同步父容器挂载的内容
```

删除父容器之后，其他容器的数据不会消失。只要有一个容器挂载，数据就不会删除。

# Dockerfile

## 初识dockerfile

- 指令使用大写字母
- 每个指令都会创建一层镜像并且提交




```
FROM            # 基础镜像
MAINTAINER      # 维护者
RUN             # 镜像构建的时候需要运行的命令
ADD             # 添加内容，比如centos镜像中再添加一个tomcat的镜像
WORKDIR         # 进入exec模式的时候的工作目录
VOLUME          # 设置挂载的容器卷
EXPOSE          # 暴露的端口
RUN
CMD             # 指定此容器启动时需要运行的命令，只有最后一个会生效，可被替代
ENTRYPOINT       # 指定此容器启动时需要运行的命令，可以追加命令
ONBUILD          # 当构建一个被继承的 dockerfile的时候，就会运行ONBUILD指令
COPY            # 类似ADD，将文件拷贝到镜像中
ENV             # 构建的时候设置的环境变量
```

### CMD与ENTRYPOINT的区别：

cmd：

```
# dockerfile 文件
FROM centos
CMD ["ls", "-a"]

# run dockerfile，会列出所有目录
docker run <imgId>

# 如果想要追加命令，如 ls -a -l的形式
docker run <imgId> -l       # 会出错，使用-l替换了 ls -a
docker run <imgId> ls -al   # 更正形式
```

### entrypoint

```
# dockerfile 文件
FROM centos
ENTRYPOINT ["ls", "-a"]

# build后run dockerfile，会列出所有目录
docker run <imgId>

# 如果想要追加命令，如 ls -a -l的形式
docker run <imgId> -l       # 相当于 ls -al
```

### 举例：

```
# 构建一个自己的centos
FROM centos
MAINTAINER 2mw<appose@gmail.com>

ENV MYPATH /usr/local   # 设置环境变量
WORKDIR $MYPATH

RUN yum -y install vim
RUN yun -y install net-tools

EXPOSE 80       # 暴露端口

CMD echo $MYPATH
CMD echo "====END===="
CMD /bin/bash
```

## 实战：构建tomcat镜像

### 需要准备tomcat+jdk的安装包

```
FROM centos
MAINTAINER 2mw<appose@gmail.com>

COPY readme.txt /usr/local/readme.txt           # 将主机上的readme.txt拷贝到镜像中
ADD jdk-8u11-linux-x64.tar.gz    /usr/local/    # Add命令会自动解压
ADD apache-tomcat-x64.tar.gz     /usr/local/

RUN yum -y install vim

ENV MYPATH /usr/local
WORKDIR $MYPATH
```

```
ENV JAVA_HOME /usr/local/jdk1.8.0_11
ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
ENV CATALINA_HOME /usr/local/apache-tomcat
ENV CATALINA_BASH /usr/local/apache-tomcat
ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/lib:$CATALINA_HOME/bin

EXPOSE 8080

CMD /usr/local/apache-tomcat/bin/startup.sh && tail -F /usr/local/apache-tomcat/bin/logs/catalina.out
```

构建：

```
# 如果文件名为Dockerfile，使用docker build的时候就不需要指定文件名
docker build -t <name> .
```

## 发布自己的镜像

> dockerhub

1. 登录：

   ```
   docker login -u <account>
   ```

2. Push

   ```
   # 加一个tag（可选）
   docker tag <oldimg> <newimg>:<tag>
   # push
   docker push <name/img>:<tag>
   ```

> 阿里云

1. 创建命名空间
2. 创建容器镜像

# Docker网络

## docker0地址

```
[root@kuangshen tomcat]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:16:3e:30:27:f4 brd ff:ff:ff:ff:ff:ff
    inet 172.17.90.138/20 brd 172.17.95.255 scope global dynamic eth0
       valid_lft 310744886sec preferred_lft 310744886sec
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:bb:71:07:06 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

查看内部容器ip地址：

```
docker exec -it nginx01 ip addr

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

查看Docker网络信息：

```
docker network ls
docker network inspect 9eb8509e1d11
```

```
{
    "Name": "bridge",
    "Id": "9eb8509e1d11e0ff5a52cf9cf5b758cbf1f3017a4aaecd0332e3a98385b71eb8",
    "Created": "2021-08-03T01:50:17.5758794Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
        "Driver": "default",
        "Options": null,
        "Config": [
            {
                "Subnet": "172.17.0.0/16",
                "Gateway": "172.17.0.1"
            }
        ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "07386948c83e25b6ec5a984781f02c70e9bbfcc102afb5fbeda7fa0300364ce2": {
            "Name": "nginx02",
            "EndpointID": "c7ba5f239d7502b348cfb1077f1b0bb05b9c99cc56e8ba7a2095593093cbb6f2",
            "MacAddress": "02:42:ac:11:00:03",
            "IPv4Address": "172.17.0.3/16",
            "IPv6Address": ""
        },
        "10b13848801668474ab7e708a2a29bb675c121b48b095ec8b47b5e004e845ea9": {
            "Name": "nginx01",
            "EndpointID": "76fbe2e051c604f1233c912428a007a205f8574dfd9146de29ac02ac8e7c08bf",
            "MacAddress": "02:42:ac:11:00:02",
            "IPv4Address": "172.17.0.2/16",
            "IPv6Address": ""
        }
```

```
    },
    "Options": {
        "com.docker.network.bridge.default_bridge": "true",
        "com.docker.network.bridge.enable_icc": "true",
        "com.docker.network.bridge.enable_ip_masquerade": "true",
        "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
        "com.docker.network.bridge.name": "docker0",
        "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
}
```


## --link

> 不推荐使用

实现两个主机ping互通

```
docker run -d -P --name nginx01 nginx
docker run -d -P --name nginx02 --link nginx01 nginx
docker exec -it nginx02 ping nginx01   # 可以ping通
#docker exec -it nginx01 ping nginx02  # 不可以ping通，因为没有配置
```

原理是将nginx01的ip放到其 `/etc/hosts` 文件中

## 自定义网络

如果直接启动一个镜像会带一个默认参数：

```
docker run -d -P --name nginx01 --net bridge nginx   # 默认带 --net bridge
```

创建网络：

```
docker network create --driver bridge --subnet 192.168.0.0/16 --gateway 192.168.0.1 <net-name>
# --driver bridge 为指定默认桥接模式，可以忽略
```

- 自定义网络可以使用 `--link` 后的效果。

将不同子网的主机之间相互连通：

```
docker network connect <net-name> <con-name>
```

相当于将此容器加入到这个网络中