# Docker学习

* docker概述
* docker安装
* docker命令
  + 镜像命令
  + 容器命令
  + 操作命令
* docker镜像
* 容器数据卷
* dockerfile
* docker网络原理
* idea整合docker
* docker compose
* docker swarm 简化版的k8s
* CI/CD Jenkins

# Docker概述

## Docker为什么会出现？

运维与开发 环境配置十分麻烦，项目带上环境一起打包！

## Docker的历史

## Docker能干嘛

之前的虚拟机技术

容器化技术

不是模拟一个完整的系统

比较Docker和虚拟机技术的不同

* 传统虚拟机，虚拟出一台硬件运行一个完整的操作系统，然后在这个系统上安装和运行软件
* 容器内的应用直接运行在宿主机的内容，容器没有自己的内核，也没有虚拟我们的硬件，所以就轻便了
* 每个容器见是相互隔离的，每个容器内斗鱼哦属于自己的文件系统，互不影响

DevOps(开发运维)

##### 应用更加快捷的交付和部署

打包镜像发布测试，一键运行

##### 更便捷的升级和扩容

部署就像搭积木

##### 更简单的运维

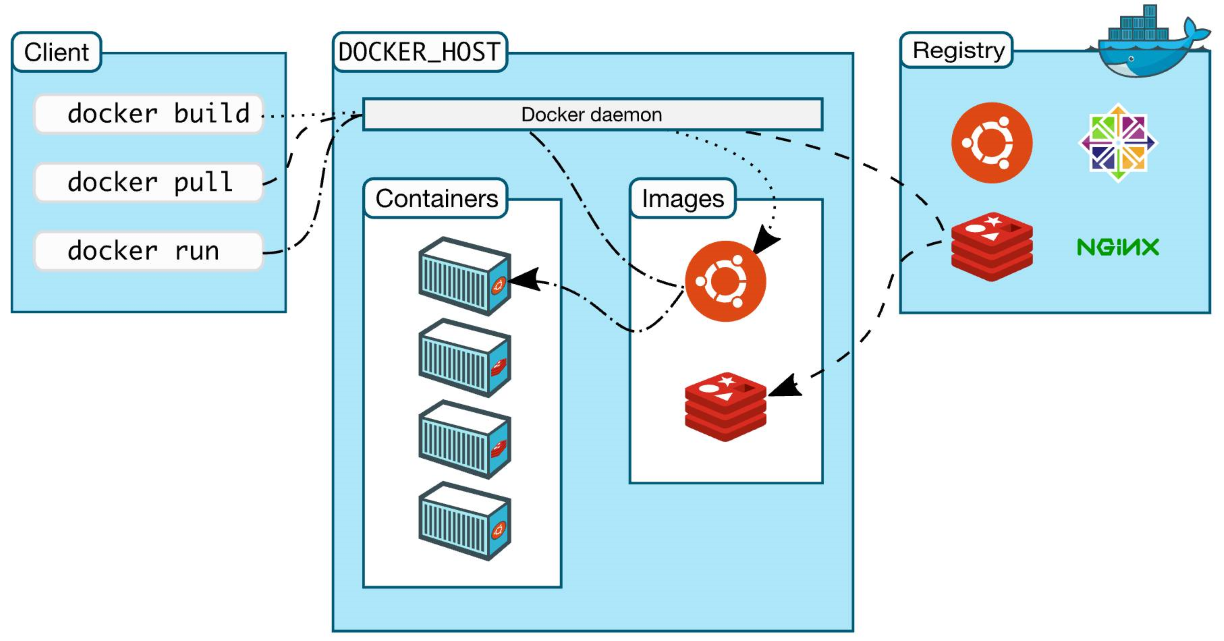
开发测试环境高度一致

##### 更高效的计算机资源的使用

docker是内核级别的虚拟化，可以在一个物理机上运行很多的容器实例。

# Docker安装

## Docker的基本组成



**镜像（image）：** 好比一个模版，可以通过模版来创建容器服务，tomcat镜像=》run=》tomcat容器，通过这个镜像可以创建多个容器（ 最终服务或者项目就在容器中运行）。

**容器**（**container**）：Docker利用容器技术，独立运行一个或者一组应用

启动，停止，删除，基本命令

**仓库**（**repository**）：仓库就是存放镜像的地方 docker hub 阿里云都有容器服务器（配置镜像服务器）

## 安装docker

### Docker 重启

systemctl restart docker.service  
//设置doocker自动重启  
systemctl enable docker

## Docker run的流程

# 底层原理

##### **Docker怎么工作的**

Docker是一个Client-Server结构的系统

##### **Docker为什么比虚拟机快**

1、Docker有这比虚拟机更少的抽象层

GuestOs现在都可以支持了

# Docker的常用命令

## 帮助命令

docker version  
docker info  
docker 命令 --help

Docker帮助文档地址

## 镜像命令

##### **docker images 查看所有的镜像**

wk@WKdeMacBook-Pro ~ % docker images   
REPOSITORY TAG IMAGE ID CREATED SIZE  
docker/getting-started latest 021a1b85e641 3 months ago 27.6MB  
hello-world latest bf756fb1ae65 14 months ago 13.3kB  
  
# 解释  
REPOSITORY 镜像的仓库源  
IMAGE ID 镜像的ID  
CREATED 创建时间  
SIZE 大小  
  
  
# 可选项  
 -a, --all # 列出所有的  
 -q, --quiet # 只显示镜像的ID

##### **docker search 搜索镜像**

-f, --filter filter Filter output based on conditions provided  
 --format string Pretty-print search using a Go template  
 --limit int Max number of search results (default 25)  
 --no-trunc Don't truncate output  
   
# 可选项，通过收藏来过滤  
 --filter=STARS=3000 搜索出来的镜像就是stras大于3000的

##### **docker pull** **下载镜像**

Options:  
 -a, --all-tags Download all tagged images in the repository  
 --disable-content-trust Skip image verification (default true)  
 --platform string Set platform if server is multi-platform capable  
 -q, --quiet Suppress verbose output

wk@WKdeMacBook-Pro ~ % docker pull mysql  
Using default tag: latest #不写tag，默认是下载最新版的  
latest: Pulling from library/mysql  
a076a628af6f: Pull complete #分层下载  
f6c208f3f991: Pull complete   
88a9455a9165: Pull complete   
406c9b8427c6: Pull complete   
7c88599c0b25: Pull complete   
25b5c6debdaf: Pull complete   
43a5816f1617: Pull complete   
1a8c919e89bf: Pull complete   
9f3cf4bd1a07: Pull complete   
80539cea118d: Pull complete   
201b3cad54ce: Pull complete   
944ba37e1c06: Pull complete   
Digest: sha256:feada149cb8ff54eade1336da7c1d080c4a1c7ed82b5e320efb5beebed85ae8c #签名信息   
Status: Downloaded newer image for mysql:latest  
docker.io/library/mysql:latest #真实地址  
  
#等价于它  
docker pull mysql  
docker.io/library/mysql:latest  
  
#指定版本下载  
docker pull mysql:5.7

##### **docker rmi 删除镜像**

docker rmi -f 镜像ID #删除指定的镜像  
docker rmi -f $(docker images -aq) #删除所有的镜像

## 容器命令

**说明：我们有了镜像才可以创建容器**

docker pull centos

##### **新建容器并启动**

docker build [可选参数] image  
#参数说明 （Linux命令中--后面一般接全拼 -后面接缩写）  
-f #指定要使用的Dockerfile路径  
--tag,-t #镜像的名字及标签  
--network #默认default.在构造期间设置RUN指令的网络模式  
docker run [可选参数] image  
#参数说明  
--name "Name"   
-d #后台方式运行  
-it #使用交互方式运行，进入容器查看内容  
-p #指定容器端口  
 -p 主机端口:容器端口（常用）  
 -p 容器端口  
-P #随机指定端口  
  
#测试 启动并进入容器  
wk@WKdeMacBook-Pro ~ % docker run -it centos /bin/bash   
[root@27009aaa79ba /]# ls #很多命令不完善  
bin etc lib lost+found mnt proc run srv tmp var  
dev home lib64 media opt root sbin sys usr  
#从容器中退回主机  
[root@27009aaa79ba /]# exit  
exit

##### **列出所有正在运行的容器**

#docker ps 命令  
-a #列出当前正在运行的容器+历史运行的容器  
-n=? #显示最近创建的容器  
-q #只显示容器的编号  
wk@WKdeMacBook-Pro ~ % docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
cd91e6e3d0fe centos "/bin/bash" 35 minutes ago Up 35 minutes quizzical\_noyce

| 选 项 | 作 用 |
| --- | --- |
| a | 显示一个终端的所有进程 |
| u | 显示进程的归属用户及内存的使用情况 |
| x | 显示没有控制终端的进程 |
| -l | 长格式显示更加详细的信息 |
| -e | 显示所有进程 |

ps 命令中最常用的选项有以下 3 个：

* ps aux 命令可以查看系统中所有的进程。 ps -ef也是查看所有的进程。两者区别就是以不同风格显示出进程的详细信息
* ps -le 命令不仅可以查看系统中全部的进程，而且还能看到进程的父进程 PID 和进程优先级。
* ps -l命令只能查看当前Shell产生的进程。

##### **退出容器**

exit #直接退出容器并停止  
Ctrl+P+Q #容器不停止退出

##### **删除容器**

docker rm 容器ID #删除指定ID的容器，不能删除正在运行的容器  
docker rm -f $(docker ps -aq) #删除所有的容器  
docker ps -a -q|xargs docker rm #删除所有的容器

##### **启动和停止容器的操作**

docker start 容器ID  
docker restart 容器ID  
docker stop 容器ID  
docker kill 容器ID

## 常用其他命令

##### **后台启动容器**

#命令 docker run -d 镜像名  
wk@WKdeMacBook-Pro ~ % docker run -d centos  
  
#问题 docker ps发现centos停止了  
#常见的坑，docker容器使用后台运行，就必须要有一个前台进程，docker发现没有进程，就会自动停止  
#nginx，容器启动后，发现自己没有提供服务，就回立刻停止，没有程序了

##### **查看日志命令**

#自己编一段shell脚本  
docker run -d centos /bin/sh -c "while true;do echo kuangsheng;sleep 1;done"  
docker logs -tf --tail 10 + 容器ID #显示最新的十条数据  
  
  
###利用 "sh -c" 命令，它可以让 bash 将一个字串作为完整的命令来执行，这样就可以将 sudo 的影响范围扩展到整条命令。具体用法如下：  
$ sudo /bin/sh -c 'echo "hahah" >> test.asc'

##### **查看容器中的进程信息**

#命令 docker top 容器ID  
wk@WKdeMacBook-Pro ~ % docker top f6360605cc0b  
UID PID PPID C STIME TTY   
root 6346 6321 0 05:46 ?

##### **查看镜像的元数据**

#命令  
docker inspect 容器ID  
#测试  
wk@WKdeMacBook-Pro ~ % docker inspect f6360605cc0b   
[  
 {  
 "Id": "f6360605cc0bc993b673b3f0bf7d33a573d19bf714d6c0bf96aead04b6b0aa23",  
 "Created": "2021-03-24T05:46:40.577212Z",  
 "Path": "/bin/bash",  
 "Args": [],  
 "State": {  
 "Status": "running",  
 "Running": true,  
 "Paused": false,  
 "Restarting": false,  
 "OOMKilled": false,  
 "Dead": false,  
 "Pid": 6346,  
 "ExitCode": 0,  
 "Error": "",  
 "StartedAt": "2021-03-24T05:46:40.9508973Z",  
 "FinishedAt": "0001-01-01T00:00:00Z"  
 },  
 "Image": "sha256:300e315adb2f96afe5f0b2780b87f28ae95231fe3bdd1e16b9ba606307728f55",  
 "ResolvConfPath": "/var/lib/docker/containers/f6360605cc0bc993b673b3f0bf7d33a573d19bf714d6c0bf96aead04b6b0aa23/resolv.conf",  
 "HostnamePath": "/var/lib/docker/containers/f6360605cc0bc993b673b3f0bf7d33a573d19bf714d6c0bf96aead04b6b0aa23/hostname",  
 "HostsPath": "/var/lib/docker/containers/f6360605cc0bc993b673b3f0bf7d33a573d19bf714d6c0bf96aead04b6b0aa23/hosts",  
 "LogPath": "/var/lib/docker/containers/f6360605cc0bc993b673b3f0bf7d33a573d19bf714d6c0bf96aead04b6b0aa23/f6360605cc0bc993b673b3f0bf7d33a573d19bf714d6c0bf96aead04b6b0aa23-json.log",  
 "Name": "/sharp\_elbakyan",  
 "RestartCount": 0,  
 "Driver": "overlay2",  
 "Platform": "linux",  
 "MountLabel": "",  
 "ProcessLabel": "",  
 "AppArmorProfile": "",  
 "ExecIDs": null,  
 "HostConfig": {  
 "Binds": null,  
 "ContainerIDFile": "",  
 "LogConfig": {  
 "Type": "json-file",  
 "Config": {}  
 },  
 "NetworkMode": "default",  
 "PortBindings": {},  
 "RestartPolicy": {  
 "Name": "no",  
 "MaximumRetryCount": 0  
 },  
 "AutoRemove": false,  
 "VolumeDriver": "",  
 "VolumesFrom": null,  
 "CapAdd": null,  
 "CapDrop": null,  
 "CgroupnsMode": "host",  
 "Dns": [],  
 "DnsOptions": [],  
 "DnsSearch": [],  
 "ExtraHosts": null,  
 "GroupAdd": null,  
 "IpcMode": "private",  
 "Cgroup": "",  
 "Links": null,  
 "OomScoreAdj": 0,  
 "PidMode": "",  
 "Privileged": false,  
 "PublishAllPorts": false,  
 "ReadonlyRootfs": false,  
 "SecurityOpt": null,  
 "UTSMode": "",  
 "UsernsMode": "",  
 "ShmSize": 67108864,  
 "Runtime": "runc",  
 "ConsoleSize": [  
 0,  
 0  
 ],  
 "Isolation": "",  
 "CpuShares": 0,  
 "Memory": 0,  
 "NanoCpus": 0,  
 "CgroupParent": "",  
 "BlkioWeight": 0,  
 "BlkioWeightDevice": [],  
 "BlkioDeviceReadBps": null,  
 "BlkioDeviceWriteBps": null,  
 "BlkioDeviceReadIOps": null,  
 "BlkioDeviceWriteIOps": null,  
 "CpuPeriod": 0,  
 "CpuQuota": 0,  
 "CpuRealtimePeriod": 0,  
 "CpuRealtimeRuntime": 0,  
 "CpusetCpus": "",  
 "CpusetMems": "",  
 "Devices": [],  
 "DeviceCgroupRules": null,  
 "DeviceRequests": null,  
 "KernelMemory": 0,  
 "KernelMemoryTCP": 0,  
 "MemoryReservation": 0,  
 "MemorySwap": 0,  
 "MemorySwappiness": null,  
 "OomKillDisable": false,  
 "PidsLimit": null,  
 "Ulimits": null,  
 "CpuCount": 0,  
 "CpuPercent": 0,  
 "IOMaximumIOps": 0,  
 "IOMaximumBandwidth": 0,  
 "MaskedPaths": [  
 "/proc/asound",  
 "/proc/acpi",  
 "/proc/kcore",  
 "/proc/keys",  
 "/proc/latency\_stats",  
 "/proc/timer\_list",  
 "/proc/timer\_stats",  
 "/proc/sched\_debug",  
 "/proc/scsi",  
 "/sys/firmware"  
 ],  
 "ReadonlyPaths": [  
 "/proc/bus",  
 "/proc/fs",  
 "/proc/irq",  
 "/proc/sys",  
 "/proc/sysrq-trigger"  
 ]  
 },  
 "GraphDriver": {  
 "Data": {  
 "LowerDir": "/var/lib/docker/overlay2/a1982c527266d980dcd77f13ea29e2352edfdee1168cf25e89134c4fceb2f670-init/diff:/var/lib/docker/overlay2/b5bb6a0520d8edf62918d2a03fbcebb5b4238d0c57debb8f3e4e92d472e75519/diff",  
 "MergedDir": "/var/lib/docker/overlay2/a1982c527266d980dcd77f13ea29e2352edfdee1168cf25e89134c4fceb2f670/merged",  
 "UpperDir": "/var/lib/docker/overlay2/a1982c527266d980dcd77f13ea29e2352edfdee1168cf25e89134c4fceb2f670/diff",  
 "WorkDir": "/var/lib/docker/overlay2/a1982c527266d980dcd77f13ea29e2352edfdee1168cf25e89134c4fceb2f670/work"  
 },  
 "Name": "overlay2"  
 },  
 "Mounts": [],  
 "Config": {  
 "Hostname": "f6360605cc0b",  
 "Domainname": "",  
 "User": "",  
 "AttachStdin": true,  
 "AttachStdout": true,  
 "AttachStderr": true,  
 "Tty": true,  
 "OpenStdin": true,  
 "StdinOnce": true,  
 "Env": [  
 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"  
 ],  
 "Cmd": [  
 "/bin/bash"  
 ],  
 "Image": "centos",  
 "Volumes": null,  
 "WorkingDir": "",  
 "Entrypoint": null,  
 "OnBuild": null,  
 "Labels": {  
 "org.label-schema.build-date": "20201204",  
 "org.label-schema.license": "GPLv2",  
 "org.label-schema.name": "CentOS Base Image",  
 "org.label-schema.schema-version": "1.0",  
 "org.label-schema.vendor": "CentOS"  
 }  
 },  
 "NetworkSettings": {  
 "Bridge": "",  
 "SandboxID": "307a7902f4b2710ad4909622aa71adc03969f90cd4a29bb9996b89a7d25c5694",  
 "HairpinMode": false,  
 "LinkLocalIPv6Address": "",  
 "LinkLocalIPv6PrefixLen": 0,  
 "Ports": {},  
 "SandboxKey": "/var/run/docker/netns/307a7902f4b2",  
 "SecondaryIPAddresses": null,  
 "SecondaryIPv6Addresses": null,  
 "EndpointID": "6159fef3b2ee0bc09ca0eacfe3c7ff9d608b084323191958e2ce57cb4a1913f0",  
 "Gateway": "172.17.0.1",  
 "GlobalIPv6Address": "",  
 "GlobalIPv6PrefixLen": 0,  
 "IPAddress": "172.17.0.2",  
 "IPPrefixLen": 16,  
 "IPv6Gateway": "",  
 "MacAddress": "02:42:ac:11:00:02",  
 "Networks": {  
 "bridge": {  
 "IPAMConfig": null,  
 "Links": null,  
 "Aliases": null,  
 "NetworkID": "e2add15336350ea9c4f946eb59a2302d36e9371f77867bdf48af736f4a3b15e1",  
 "EndpointID": "6159fef3b2ee0bc09ca0eacfe3c7ff9d608b084323191958e2ce57cb4a1913f0",  
 "Gateway": "172.17.0.1",  
 "IPAddress": "172.17.0.2",  
 "IPPrefixLen": 16,  
 "IPv6Gateway": "",  
 "GlobalIPv6Address": "",  
 "GlobalIPv6PrefixLen": 0,  
 "MacAddress": "02:42:ac:11:00:02",  
 "DriverOpts": null  
 }  
 }  
 }  
 }  
]

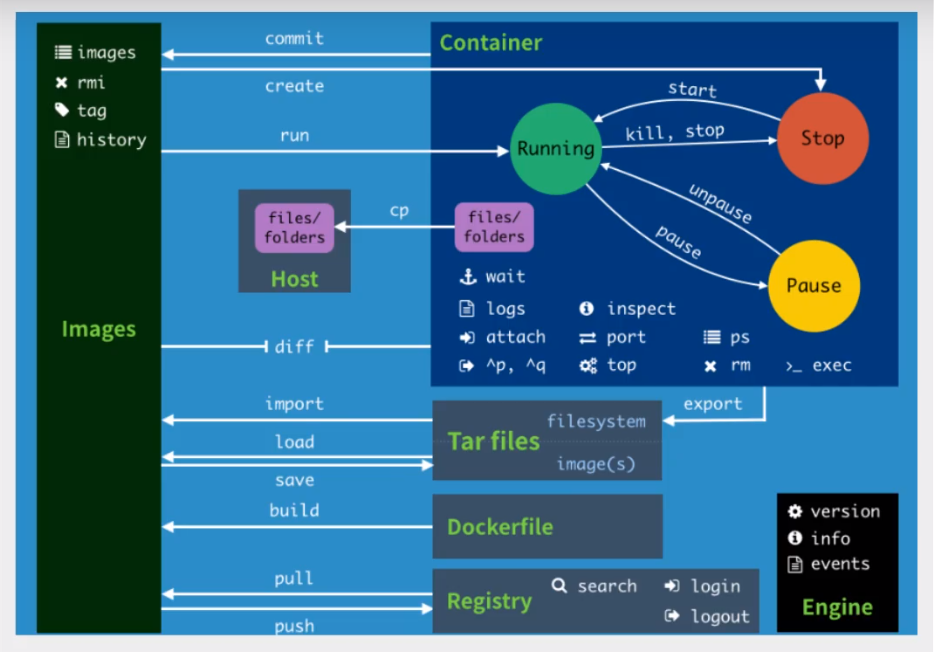
##### **进入正在运行的容器**

#命令  
docker exec -it 容器ID  
#测试, -it以交互的形式尽心，使用/bin/bash的命令进行  
wk@WKdeMacBook-Pro ~ % docker exec -it f6360605cc0b /bin/bash   
[root@f6360605cc0b /]# ps -ef  
UID PID PPID C STIME TTY TIME CMD  
root 1 0 0 05:46 pts/0 00:00:00 /bin/bash  
root 17 0 0 06:00 pts/1 00:00:00 /bin/bash  
root 31 17 0 06:01 pts/1 00:00:00 ps -ef  
#方式二  
#命令  
docker attach 容器ID  
#测试  
wk@WKdeMacBook-Pro ~ % docker attach f6360605cc0b   
[root@f6360605cc0b /]#   
正在执行当前的代码  
   
#docker exec #进入容器后开启一个新的终端，可以在里面进行操作  
#docker attach #进入容器正在执行的终端，不会启动新的进程

##### **从容器内拷贝文件到主机**

#命令  
docker cp 容器ID:容器内路径 目的的主机路径  
#测试  
[root@f6360605cc0b /]# cd home/  
[root@f6360605cc0b home]# ls  
[root@f6360605cc0b home]# touch test.java  
[root@f6360605cc0b home]# ls  
test.java  
[root@f6360605cc0b home]# exit  
exit  
wk@WKdeMacBook-Pro ~ % docker cp f6360605cc0b:/home/test.java /Users/wk   
wk@WKdeMacBook-Pro ~ %   
  
#拷贝是一个手动过程，未来我们使用 -v 卷的技术，可以实现

## 小结



## 作业练习

作业：docker安装nginx

wk@WKdeMacBook-Pro ~ % docker search nginx  
wk@WKdeMacBook-Pro ~ % docker pull nginx  
#运行测试  
# -d 后台运行  
# -p 宿主机端口:容器端口  
wk@WKdeMacBook-Pro ~ % docker run -d -p 9999:80 nginx  
wk@WKdeMacBook-Pro ~ % docker ps   
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
aedc1990d601 nginx "/docker-entrypoint.…" 6 seconds ago Up 4 seconds 0.0.0.0:9999->80/tcp epic\_williams  
#进入容器  
wk@WKdeMacBook-Pro ~ % docker exec -it aedc1990d601 /bin/bash  
root@aedc1990d601:/etc/nginx# ls  
conf.d fastcgi\_params koi-utf koi-win mime.types modules nginx.conf scgi\_params uwsgi\_params win-utf

作业：安装tomcat

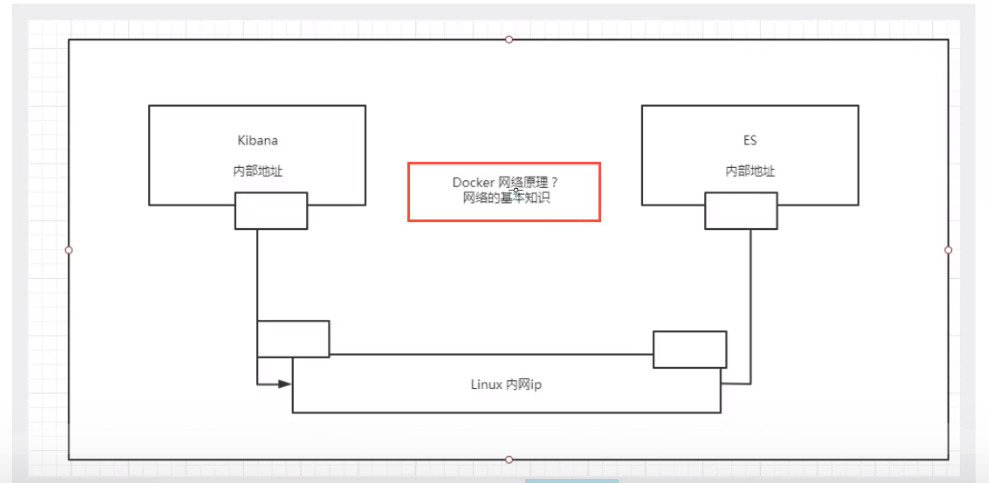
# 官方的使用  
docker run -it --rm tomcat:9.0  
  
#我们之前的启动都是后台，停止了容器之后，容器还是可以查到 docker run -it --rm，一般用来测试，用完即删除  
  
#测试  
wk@WKdeMacBook-Pro ~ % docker run -d -p 8888:8080 --name tomcat 040bdb29ab37  
  
#进入容器  
wk@WKdeMacBook-Pro ~ % docker exec -it tomcat /bin/bash   
#发现问题，没有很多命令，没有webapps 最小镜像，所有不必要的都删除

作业：部署es+kibana

# es 暴露端口很多  
# es 十分占用内存  
# es 的数据一般需要放置到安全目录，挂载  
# --net somenetwork 网络配置  
  
#启动  
$ docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" elasticsearch:7.6.2  
# docker stats 查看docker cpu状态  
#测试es  
wk@WKdeMacBook-Pro ~ % curl localhost:9200  
{  
 "name" : "e06b0d4f2a2d",  
 "cluster\_name" : "docker-cluster",  
 "cluster\_uuid" : "viWwzLxBReSwZbAj2G5M0w",  
 "version" : {  
 "number" : "7.6.2",  
 "build\_flavor" : "default",  
 "build\_type" : "docker",  
 "build\_hash" : "ef48eb35cf30adf4db14086e8aabd07ef6fb113f",  
 "build\_date" : "2020-03-26T06:34:37.794943Z",  
 "build\_snapshot" : false,  
 "lucene\_version" : "8.4.0",  
 "minimum\_wire\_compatibility\_version" : "6.8.0",  
 "minimum\_index\_compatibility\_version" : "6.0.0-beta1"  
 },  
 "tagline" : "You Know, for Search"  
}

#赶紧关闭，增加内存限制，修改配置文件 -e 环境配置修改  
docker run -d --name elasticsearch01 -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" -e ES\_JAVA\_OPTS="-Xms64m -Xmx512m" elasticsearch:7.6.2

作业：使用kibana连接es



## 可视化

* portainer（先用这个）

docker run -d -p 8088:9000 \--restart=always -v /var/run/docker.sock:/var/run/docker.sock --privileged=true portainer/portainer

# Docker镜像讲解

## 如何获取镜像

* 从远程仓库下载
* 拷贝
* 自己制作镜像Dockerfile

## Docker镜像加载原理

UFS联合文件系统

## 分层理解

## commit镜像

docker commit 提交容器成为一个新的副本  
  
#命令和git原理类似  
docker commit -m="提交的描述信息" -a="作者" 容器ID 目标镜像名:[TAG]

##### 实战测试

#启动一个默认的tomcat  
#发现tomcat没有webapps项目，将webapps.disk拷贝  
#将我们操作过的容器进行commit提交为一个镜像，这就是我们自己修改过的镜像  
wk@WKdeMacBook-Pro ~ % docker commit -m="add webapps" -a="wangkai" 594639bdc55c tomcat01:1.0   
sha256:0bd28328a218abf43bda146a5e29eb920ec0baf10217fda0a66502399af4acb2  
wk@WKdeMacBook-Pro ~ % docker images  
REPOSITORY TAG IMAGE ID CREATED SIZE  
tomcat01 1.0 0bd28328a218 5 seconds ago 653MB

学习方式说明：理解概念，但是一定要去实践，最后实践与理论相结合掌握知识

如果想要保存当前容器的状态，就可以通过commit来提交保存

# 容器数据卷

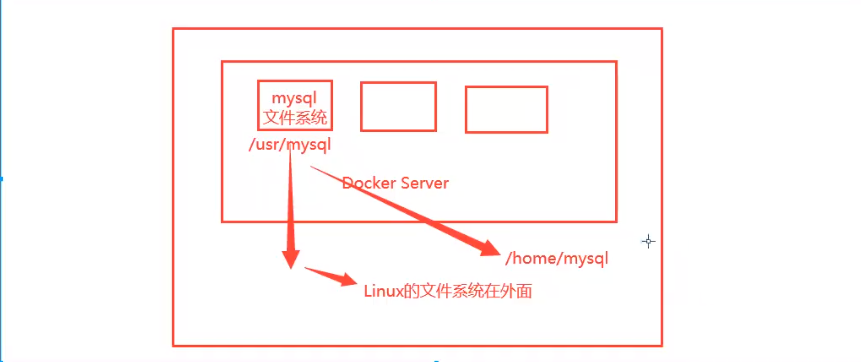
## 什么是容器数据卷

如果数据都存在容器中，那么我们删除容器，数据就会丢失！需求：数据可以持久化

Mysql，容器删了，删库跑路了。

容器之间可以有一个数据共享的技术！Docker容器中产生的数据，同步到本地！

这就是卷技术！目录的挂载，将我们容器内的目录，挂载到LInux上面！！



**总结一句话：容器的持久化和同步操作！容器间也是可以数据共享的！**

## 使用数据卷

方式一：直接使用命令来挂载 -v

docker run -it -v 主机目录:容器内目录  
#测试  
wk@WKdeMacBook-Pro ~ % docker run -it -v /Users/wk/home/test:/home 300e315adb2f /bin/bash   
  
#启动之后通过docker inspect ae522e5ce4a3查看  
#双向的过程，容器关闭依旧有效

好处：我们以后修改只需要在本地修改，容器会自动同步

## 实战：安装MySQL

#获取MySQL  
wk@WKdeMacBook-Pro ~ % docker pull mysql:5.7  
  
#运行容器进行数据挂载  
-d 后台运行  
-p 端口映射  
-v 卷挂载  
-e 环境配置  
--name 容器名字  
k@WKdeMacBook-Pro ~ % docker run -d -p 3333:3306 -v /Users/wk/docker/mysql/conf:/home/mysql/conf.d -v /Users/wk/docker/mysql/data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=123456 --name mysql01 mysql:5.7  
#官方文档  
$ docker run --name some-mysql -v /my/custom:/etc/mysql/conf.d -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:tag  
#使用数据库连接工具成功连接到docker数据库

假设我们将容器删除，发现我们挂载到本地的数据卷依旧没有消失，这就实现了容器数据卷的持久化功能！

## 具名挂载和匿名挂载

#匿名挂载  
 -v 容器内路径  
 docker run -d -P --name nginx01 -v /etc/nginx nginx  
   
 #查看所有的本地卷  
 wk@WKdeMacBook-Pro ~ % docker volume ls  
 #这里发现都是匿名的，我们没有写容器  
 #使用inspect查看卷的详细情况  
 wk@WKdeMacBook-Pro ~ % docker volume inspect d200a97f10a6a9d5f4a54482aa5990c83cc375e56ce3c9cc1e3fdb02754e6675  
[  
 {  
 "CreatedAt": "2021-03-26T13:38:57Z",  
 "Driver": "local",  
 "Labels": null,  
 "Mountpoint": "/var/lib/docker/volumes/d200a97f10a6a9d5f4a54482aa5990c83cc375e56ce3c9cc1e3fdb02754e6675/\_data",  
 "Name": "d200a97f10a6a9d5f4a54482aa5990c83cc375e56ce3c9cc1e3fdb02754e6675",  
 "Options": null,  
 "Scope": "local"  
 }  
]

# 如何让确定是具名挂载还是匿名挂载  
-v 容器内路径 #匿名挂载  
-v 卷名:容器内路径 #具名挂载  
-v /宿主机路径:容器内路径 #指定路径挂载

扩展：

#通过 -v 容器内路径:ro 控制读写权限  
docker run -d -P --name nginx01 -v juming-nginx:/etc/nginx:ro nginx  
docker run -d -P --name nginx01 -v juming-nginx:/etc/nginx:rw nginx

运行以后就只能去配置文件里面挂载镜像，因此建议停止容器并删除容器，再次挂载

## 初识DockerFile

DockerFile就是用来构建docker镜像的构建文件！命令脚本，先体验一下

通过这个脚本可以生成镜像，镜像是一层一层的，脚本是一个一个命令，每个命令都是一层。

方式二：创建一个dockerfile文件  
#文件中的指令都是大写的  
FROM centos  
VOLUME ["volume01","volume02"]  
  
CMD echo "----end----"  
CMD /bin/bash  
#生成自己的镜像，注意后面的 .  
wk@WKdeMacBook-Pro docker-test-volume % docker build -f /Users/wk/docker-test-volume/dockerfile1 -t wk/centos:1.0 .

查看卷的挂载路径

## 数据卷容器

两个MySQL同步数据

# 启动三个镜像，通过我们刚才自己写的镜像启动  
  
#只要使用 --volumes-from 各个容器之间的数据就是共享的  
  
#只要有一个容器还在使用，数据就是存在的，不是共享，类似于备份

成功同步

**容器之间配置信息的传递，数据卷容器的生命周期一直持续到没有容器使用为止。**

**但是一旦持久化到了本地，这个时候本地数据是不会被删除的**

# DockerFile

## DockerFile介绍

dockerfile是用来构建docker镜像的文件！命令参数脚本！

构建步骤：

1、编写一个dockerfile文件

2、docker build构建成为一个镜像

3、docker run 运行发布镜像

4、docker push 发布镜像（DockerHub、阿里云镜像仓库）

## DockerFile构建过程

**基础知识**

1、每个保留关键字（指令）都必须是大写的

2、指令的执行是从上到下的

3、#表示注释

4、每一个指令都会创建提交一个新的镜像层

dockerfile是面向开发的，我们以后要发布项目，做镜像，都需要编写dockerfile文件，这个文件十分简单

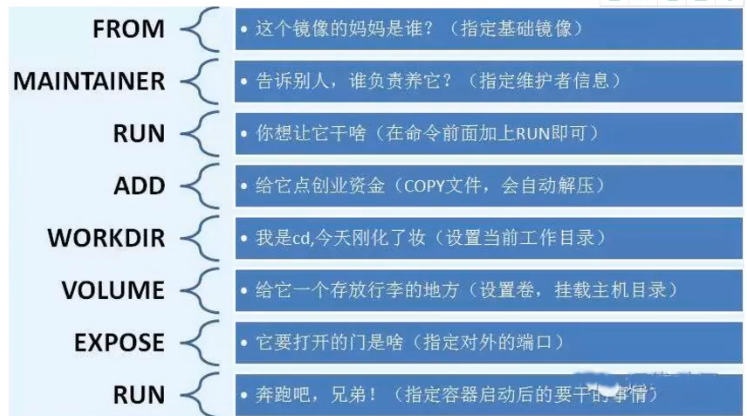
Docker镜像 逐渐成为了企业交付的标准

DockerFile：构建文件，定义了一切步骤，源代码

DockerImages：通过DockerFile构建生成的镜像，最终发布和运行的产品

Docker容器： 容器就是镜像运行起来提供服务

## DockerFile的指令



FROM #基础镜像，一切从这里开始构建  
MAINTAINER #镜像是谁写的，姓名+邮箱  
RUN #镜像运行的时候需要运行的命令  
ADD #步骤，tomcat镜像，这个tomcat压缩包！添加内容  
WORKDIR #镜像的工作目录  
VOLUME #设置容器卷，挂载的目录  
EXPOSE #指定暴露端口  
CMD #指定容器启动的时候需要运行的命令 只有最后一个会生效，可被替代  
ENTRYPOINT #指定容器启动的时候需要运行的命令，可以追加命令  
ONBUILD #当构建一个被继承DockerFile 这个时候就会运行ONBUILD命令，触发指令  
CP #类似ADD命令   
ENV #构建的时候设置环境变量

创建自己的centos

#1、编写dockerfile文件  
FROM centos  
MAINTAINER wangkai<424018092@qq.com>  
ENV MYPATH /usr/local  
WORKDIR $MYPATH  
RUN yum -y install vim  
run yum -y install net-tools  
EXPOSE 80  
CMD echo $MYPATH  
CMD echo "---end---"  
CMD /bin/bash  
#2、通过文件构建对象  
#命令 docker build -f mydockerfile-centos -t mycentos:0.1 .  
# 3、测试运行

测试，进来之后直接进入/usr/local路径

增加了vim与net相关的命令

我们可以列出本地进行的变更历史

CMD和EMTRYPOINT的区别

CMD #指定容器启动的时候需要运行的命令 只有最后一个会生效，可被替代  
ENTRYPOINT #指定容器启动的时候需要运行的命令，可以追加命令

测试CMD

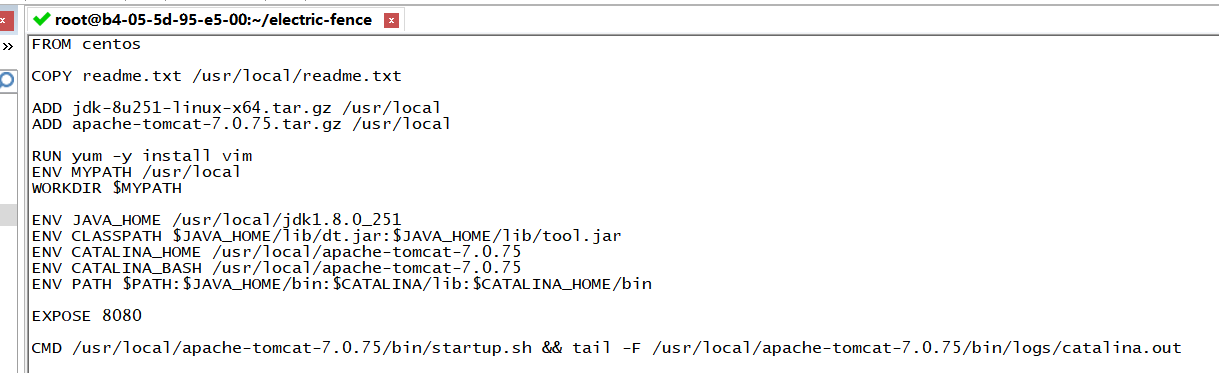
#编写docker file 文件  
 wk@WKdeMacBook-Pro  ~/docker/dockerfile/mydockerfile  cat dockertest-CMD  
 FROM centos  
CMD ["ls","-a"]  
#构建镜像  
 wk@WKdeMacBook-Pro  ~/docker/dockerfile/mydockerfile  docker build -f dockertest-CMD -t cmdtest .  
#run运行，发现直接运行 ls -a 命令生效  
 wk@WKdeMacBook-Pro  ~/docker/dockerfile/mydockerfile  ll   
total 16  
-rw-r--r-- 1 wk staff 30B 3 29 20:50 dockertest-CMD  
-rw-r--r-- 1 wk staff 201B 3 29 20:31 mydockerfile-centos  
#想加一个命令时  
 wk@WKdeMacBook-Pro  ~/docker/dockerfile/mydockerfile  docker run -it be8cf7de8763 -l   
docker: Error response from daemon: OCI runtime create failed: container\_linux.go:370: starting container process caused: exec: "-l": executable file not found in $PATH: unknown.  
#CMD的情况下 -l 替换了CMD ["ls","-a"]命令，-l不是命令，所以报错

## 实战，创建tomcat镜像

1、下载对应的镜像包 tomcat压缩包、jdk压缩包

2、编写dockerfile文件 官方命名Dockerfile在build的时候就会自己去寻找

1 FROM centos  
 2 MAINTAINER wangkai<424018092@qq.com>  
 3   
 4 COPY readme.txt /usr/local/readme.txt  
 5   
 6 ADD jdk-8u281-linux-x64.tar /usr/local  
 7 ADD apache-tomcat-8.5.64.tar.gz /usr/local  
 8   
 9 RUN yum -y install vim  
 10 ENV MYPATH /usr/local  
 11 WORKDIR $MYPATH  
 12   
 13 ENV JAVA\_HOME /usr/local/jdk1.8.0\_281  
 14 ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tool.jar  
 15 ENV CATALINA\_HOME /usr/local/apache-tomcat-8.5.64  
 16 ENV CATALINA\_BASH /usr/local/apache-tomcat-8.5.64  
 17 ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA/lib:$CATALINA\_HOME/bin  
 18   
 19 EXPOSE 8080  
 20   
 21 CMD /usr/local/apache-tomcat-8.5.64/bin/startup.sh && tail -F /usr/local/apache-tomcat-8.5.64/bin/logs/catalina.out



打包指令：tar -zcvf apache-tomcat-7.0.75.tar.gz apache-tomcat-7.0.75/

拆包指令：tar -zxvf jdk-8u251-linux-x64.tar.gz

apache-tomcat-8.5.65.tar.gz

jdk-8u281-linux-x64.tar.gz

### 3、构建镜像

docker build -t electricfence\_img .  
  
docker run -d -P --name fileserver\_contain (image\_id)  
  
docker run -d -p 8090:8080 --name app-electric-fence -v /root/tomcat/test:/usr/local/apache-tomcat-7.0.75/webapps/test -v /root/tomcat/tomcatlogs/:/usr/local/apache-tomcat-7.0.75/logs -v /root/tomcat/ROOT:/usr/local/apache-tomcat-7.0.75/webapps/ROOT electricfence\_img

## 实战，电子围栏项目部署

FROM tomcat:8-jdk8  
MAINTAINER LYL  
  
LABEL version="1.0" \  
 description="elec-fence in docker"  
  
ENV TZ=Asia/Shanghai  
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone \  
 rm -rf /usr/local/tomcat/webapps/\*  
  
#将target下的xx.war拷贝到/usr/local/tomcat/webapps/下  
ADD ./\*\*.war /usr/local/tomcat/webapps/  
  
#端口  
EXPOSE 8080  
#设置启动命令  
ENTRYPOINT ["/usr/local/tomcat/bin/catalina.sh","run"]  
  
#进入到此dockerfile目录然后生成镜像  
#docker build -t elec-fence .  
#docker run -d --restart=always --name ef1 -p 8080:8080 elec-fence

学习一下这个Dockerfile

* 将电子围栏项目的war部署在当前Dockerfile目录下面

**注意**：

* 部署war包的坑：自己解压好的war包直接放进服务器中不能直接运行，会找不到路径所以再Dockerfile中加Add命令，将war通过ADD命令进行解压
* Linux ln （link files）它的功能是为某一个文件在另外一个位置建立一个同步的链接。连接分为硬链接和软链接。
  + 硬链接的意思是一个档案可以有多个名称
  + 软链接则是产生一个特殊的档案，该档案可以指向另一个档案的位置。
  + 硬链接是存在同一个文件系统中，而软链接却可以跨越不同的文件系统。
  + 不论是硬链接或软链接都不会将原本的档案复制一份，只会占用非常少量的磁碟空间。

${JAVA\_HOME} 取变量值

## 实战，文件服务器

FROM java:8  
COPY \*.jar /app.jar  
CMD ["--cxwserver.port=8090"]  
EXPOSE 8090  
ENTRYPOINT ["java","-jar","/app.jar"]  
  
#docker build -t fileserver\_img .  
#docker run -d -p 8090:8080 --name OTA -v /home/fileserver/data:/home/fileserver/data/ fileserver\_img

#### 大Bug:

远程升级程序的时候需要在docker容器中挂载相应的路径程序，才能远程升级。

## 实战，基站app项目部署

FROM java:8  
VOLUME /tmp  
ADD basestation.jar app.jar  
EXPOSE 8110  
EXPOSE 50110  
EXPOSE 50111  
EXPOSE 47500  
RUN bash -c 'touch /app.jar'  
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]  
# docker build -t electric\_fence\_img .

## 发布自己的镜像

1、在DockeHub注册自己的账号

2、在服务器上提交自己的镜像

[root@instance-m4k4qp22 tomcat]# docker login --help  
  
Usage: docker login [OPTIONS] [SERVER]  
  
Log in to a Docker registry.  
If no server is specified, the default is defined by the daemon.  
  
Options:  
 -p, --password string Password  
 --password-stdin Take the password from stdin  
 -u, --username string Username  
[root@instance-m4k4qp22 tomcat]# docker login -u tenten91  
Password:   
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.  
Configure a credential helper to remove this warning. See  
https://docs.docker.com/engine/reference/commandline/login/#credentials-store  
  
Login Succeeded

**阿里云镜像服务器**

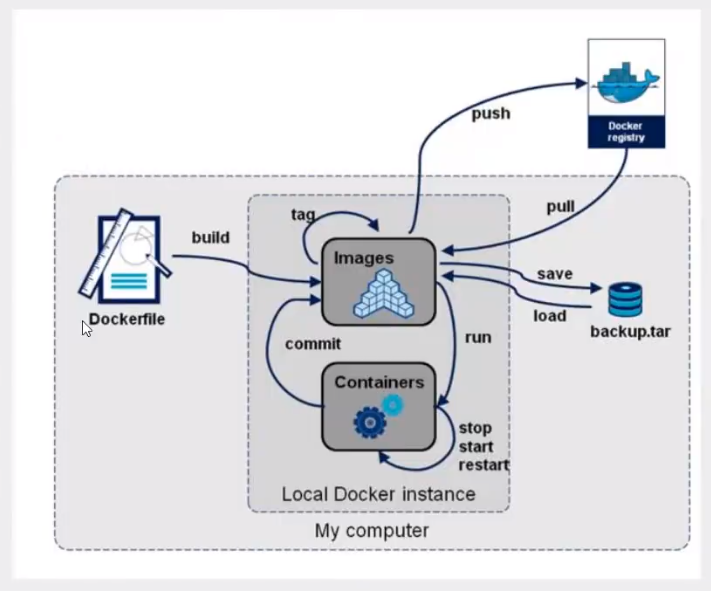
1、登陆阿里云

2、找到容器镜像服务

3、创建命名空间

4、创建

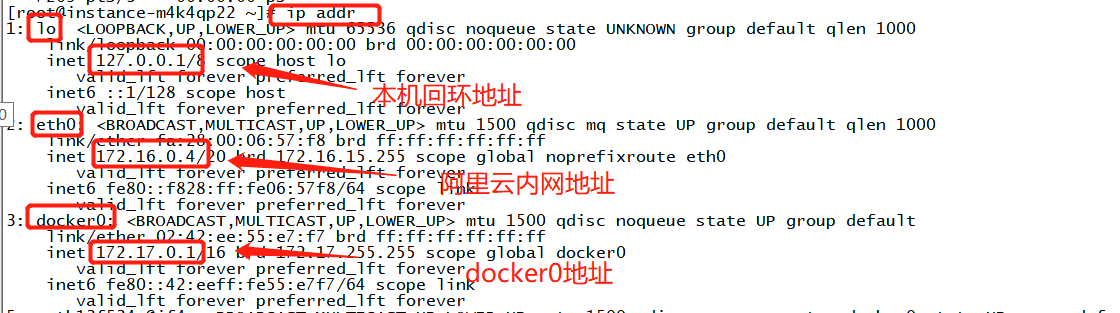
### **小结：**



# Docker网络

### 理解Docker0

测试



三个网络

#问题： docker是如何处理容器网络访问的？



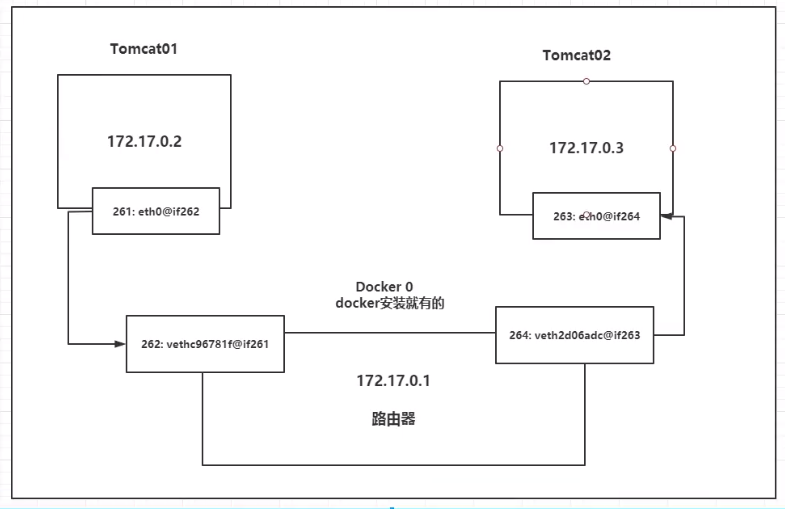
#查看容器的内部网络地址 ip addr  
  
#docker exec -it tomcat01 ip addr  
#linux可以ping通docker容器内部

原理

#每启动一个docker容器，docker就会给docker容器分配一个ip，我们只要安装了docker，就会又一个网卡docker0桥接模式，使用的技术是veth-pair技术! veth-pair充当一个桥梁，连接各种虚拟网络设备的协议   
# veth virtual ethernet 虚拟网卡  
#

3、我们来测试一下tomcat01和tomcat02

# 容器和容器之间是可以相互ping通的!



结论：tomcat01和tomcat02是共用一个路由器，docker0

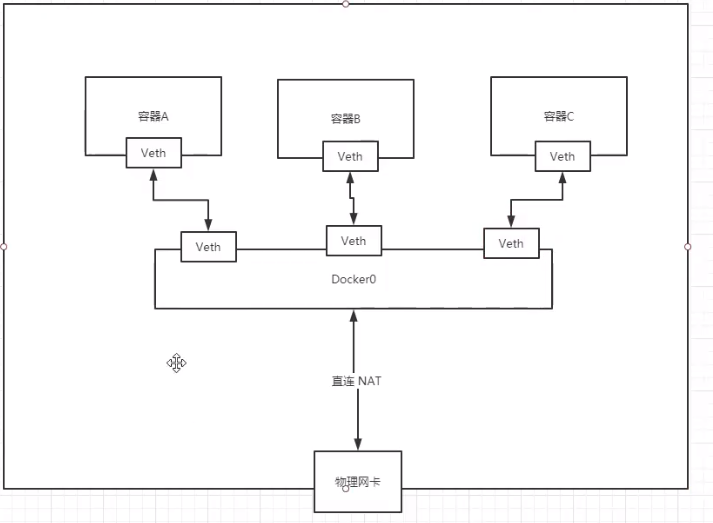
所有的容器不指定网络的情况下，都是docker0路由的，docker会给我们的容器分配一个默认的可用ip

### --link

思考一个场景，我们编写了一个微服务，database url=ip; 项目不重启，数据库ip换掉了，我们希望处理这个问题，可以名字来进行访问容器？

### 小结

Docker使用的是Linux的桥接，宿主机中是一个Docker容器的网桥 docker0



Docker中的所有的网络接口都是虚拟的。虚拟的转发效率高！

只要容器删除，对应网桥一对就没了！

思考一个场景，我们编写了一个微服务

### Docker Compose

#### 实战，楼宇系统的搭建

version: '3'  
  
services:  
  
 mysql:  
 image: mysql  
 container\_name: louyu\_mysql  
 restart: always  
 command: --default-authentication-plugin=mysql\_native\_password  
 networks:  
 - app\_net  
 volumes:  
 # 数据库  
 - ./docker/mysql/source:/docker-entrypoint-initdb.d  
 environment:  
 MYSQL\_ROOT\_PASSWORD: 123456  
 MYSQL\_DATABASE: louyu  
  
 redis:  
 # 指定镜像  
 image: redis  
 container\_name: louyu\_redis  
 restart: always  
 networks:  
 - app\_net  
 command:  
 # 执行的命令  
 redis-server  
  
 serv:  
 depends\_on:  
 - mysql  
 - redis  
 build:  
 context: ./  
 dockerfile: Dockerfile  
 container\_name: louyu\_serv  
 restart: always  
 ports:  
 - 8088:8088  
 - 1081:1081  
 networks:  
 - app\_net  
 nginx:  
 image: nginx  
 container\_name: louyu\_show  
 depends\_on:  
 - serv  
 restart: always  
 networks:  
 - app\_net  
 ports:  
 - 8089:8089  
 volumes:  
 - ./docker/nginx:/etc/nginx/conf.d/  
 - ./dist:/usr/share/nginx/html/dist  
  
networks:  
 app\_net:  
 driver: bridge  
#compose以后台模式运行加-d选项  
#docker-compose up -d  
#加防火墙,查看端口开放的命令  
#firewall-cmd --list-port  
#打开防火墙  
#firewall-cmd --zone=public --add-port=8089/tcp --permanent  
#systemctl start firewalld   
#查看防火墙的状态  
#systemctl status firewalld  
#firewall-cmd --reload

# 添加JAVA启动的必要镜像  
FROM openjdk:8-jdk-alpine  
# 作者  
MAINTAINER yd-tao <yd\_tao@outlook.com>  
# 环境  
ENV PARAMS="" TZ=Asia/Shanghai  
# 工作目录  
WORKDIR /project/lyksh/louyu  
# 端口list  
EXPOSE 8088  
# 将jar加入docker，并重命名  
ADD target/wisdom\_monitor-\*.jar wisdom\_monitor.jar  
# 同步时间  
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone  
# 运行jar包  
CMD ["java", "-jar", "./wisdom\_monitor.jar"]

**启动docker-compose**

docker-compose up -d

#### 实战，部署雷楼宇老系统

version: '3'  
  
services:  
  
 monitor\_mysql:  
 image: mysql:5.7  
 container\_name: monitor\_mysql  
 restart: always  
 command: --default-authentication-plugin=mysql\_native\_password  
 networks:  
 - app\_net  
 volumes:  
 # 数据库  
 - ./mysql/source:/docker-entrypoint-initdb.d  
 environment:  
 MYSQL\_ROOT\_PASSWORD: xidianiot@123  
 MYSQL\_DATABASE: monitor  
  
 monitor\_redis:  
 # 指定镜像  
 image: redis  
 container\_name: monitor\_redis  
 restart: always  
 networks:  
 - app\_net  
 command:  
 # 执行的命令  
 redis-server  
  
 serv:  
 depends\_on:  
 - monitor\_mysql  
 - monitor\_redis  
 build:  
 context: ./  
 dockerfile: Dockerfile  
 container\_name: monitor\_serv  
 restart: always  
 ports:  
 - 8080:8080  
 - 8090:8090  
 networks:  
 - app\_net  
  
networks:  
 app\_net:  
 driver: bridge

### Docker Swarm

### CI/CD之Jenkins

#### Docker问题

1、启动docker发现错误



最后解决问题方法是：  
进入 /etc/docker，没有daemon.json文件就自己新建一个：

cd /etc/docker

编辑daemon.json文件：  
加入这段代码：

{  
 "registry-mirrors": ["https://registry.docker-cn.com"]  
}

然后重启docker：

systemctl restart docker.service

### linux下Docker 设置自动开机

#### 1、设置docker自动开机

systemctl enable docker

#### 2、设置容器自动重启

docker run -d --restart=always --name 设置容器名 使用的镜像  
（上面命令 --name后面两个参数根据实际情况自行修改）  
 --restart具体参数值详细信息：  
       no　　　　　　　　容器退出时，不重启容器；  
       on-failure　　 只有在非0状态退出时才重新启动容器；  
       always　　　　　 无论退出状态是如何，都重启容器；

#### 3、修改已有容器，使用update

docker update --restart=always 容器ID(或者容器名)  
   
docker container update --restart=unless-stopped <容器名字>  
（容器ID或者容器名根据实际情况修改）

Linux命令：

ln:它的功能是为某一个文件在另外一个位置建立一个同不的链接，这个命令最常用的参数是-s, 具体用法是：ln -s 源文件 目标文件。

ln -s 源文件 目标文件 。 软连接： 它只会在你选定的位置上生成一个文件的镜像，不会占用磁盘空间

ln 源文件 目标文件。 硬连接：硬链接ln \*\* \*\*,没有参数-s, 它会在你选定的位置上生成一个和源文件大小相同的文件，无论是软链接还是硬链接，文件都保持同步变化。

[root@VM-16-12-centos ~]# vim readme.txt  
[root@VM-16-12-centos ~]# ln -s /root/readme.txt /home/readme.txt  
[root@VM-16-12-centos ~]# cd /home/  
[root@VM-16-12-centos home]# ls  
basestation.jar fileserver ignite logs nohup.out readme.txt  
[root@VM-16-12-centos home]# ll  
total 138676  
-rw-r--r-- 1 root root 91593201 Jun 21 11:32 basestation.jar  
drwxr-xr-x 3 root root 4096 Jun 29 15:53 fileserver  
drwxr-xr-x 3 root root 4096 Jun 21 11:22 ignite  
drwxr-xr-x 3 root root 4096 Jun 21 11:20 logs  
-rw------- 1 root root 50387395 Jun 29 20:42 nohup.out  
lrwxrwxrwx 1 root root 16 Jun 29 20:42 readme.txt -> /root/readme.txt

桥接的话，容器网络和主机网络是并列的，各占一个ip。就用宿主机的网卡

企业实战

Docker Compose

Docker Swarm

CI/CD Jenkins了流水线！