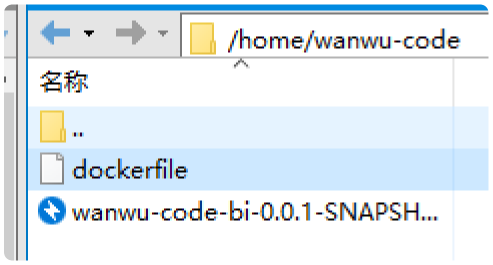
常用的Docker启动命令🌹

docker单机部署Java小技巧

在同级目录下准备一个dockerfile文件



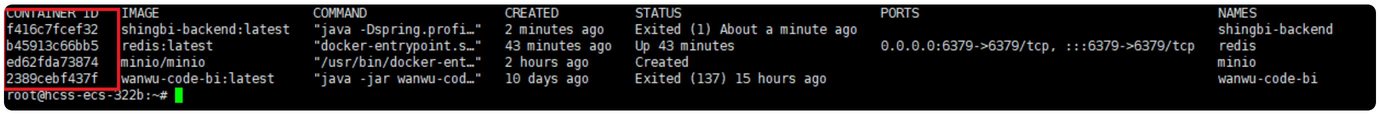
```shell
FROM openjdk:8
ENV workdir=/home/wanwuCode/wanwu-code-bi
COPY . ${workdir}
WORKDIR ${workdir}
EXPOSE 8807
CMD ["java","-jar","wanwu-code-bi-0.0.1-SNAPSHOT.jar"]
```

> workdir={jar包放置的文件夹}
>
> CMD ["java","–jar","{jar包名称}"]

```shell
//先到文件夹目录下面
cd /home/wanwu-code
//进行项目打包  -t后面跟的是你项目打包的名字
docker build -f ./dockerfile -t wanwu-code .
//进行容器的运行 --name 后面是你想取的容器名字 紧跟着后面是 你上面打包的名字+:latest
//-p后面是你项目的端口号 如果不通过nginx代理出去想通过地址使用，就要到服务器防火墙进行放行操作
docker run -d -p 8806:8806 --name "wanwu-code" wanwu-code:latest
//查看容器列表
docker ps -a
//如果查看容器是否启动成功或者失败 看控制台 容器id不用打全，前几位数就行
docker logs -f 容器id

//部署失败怎么删除容器
//找到容器id
//删除容器 容器id不用打全，前几位数就行
docker rm -f f416
```

红色框住的是容器id



```
CONTAINER ID   IMAGE                    COMMAND                  CREATED          STATUS                        PORTS                                       NAMES
f416c7fcef32   shingbi-backend:latest   "java -Dspring.profi…"   2 minutes ago    Exited (1) About a minute ago                                             shingbi-backend
b45913c66bb5   redis:latest             "docker-entrypoint.s…"   43 minutes ago   Up 43 minutes                 0.0.0.0:6379->6379/tcp, :::6379->6379/tcp   redis
ed62fda73874   minio/minio              "/usr/bin/docker-ent…"   2 hours ago      Created                                                                   minio
2389cebf437f   wanwu-code-bi:latest     "java -jar wanwu-cod…"   10 days ago      Exited (137) 15 hours ago                                                 wanwu-code-bi
root@hcss-ecs-322b:~#
```

docker 的安装（Centos）

```
yum安装gcc相关环境
yum -y install gcc
yum -y install gcc-c++
#安装yum的工具
yum install -y yum-utils
#添加阿里云docker镜像地址
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
#更新yum软件包索引
yum makecache
#安装docker
yum install --allowerasing docker-ce docker-ce-cli containerd.io         #不要用yum install docker-ce docker-ce-cli containerd.io安装，centos8内置了podman容器会冲突（是一个跟docker差不多的容器，因为centos8放弃了对docker的支持，代替方案就是podman）--allowerasing命令用于将要安装的包替代冲突的包
#启动docker
systemctl start docker
#开机自启
systemctl enable docker
```

docker 的安装（Ubuntu）

Shell

```shell
apt-get remove docker docker-engine docker.io containerd runc

sudo apt update
sudo apt upgrade
apt-get install ca-certificates curl gnupg lsb-release
curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] http://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
apt-get install docker-ce docker-ce-cli containerd.io

sudo usermod -aG docker $USER

systemctl start docker

apt-get -y install apt-transport-https ca-certificates curl software-properties-common
service docker restart
```

docker的停止

systemctl stop docker

docker 安装 WireGuard

Plain Text

```
docker run -d \
 --name=wg-easy \
 -e WG_HOST=43.138.246.140 \
 -e PASSWORD=oo771901874 \
 -v /usr/local/src/wireguard:/etc/wireguard \
 -p 51820:51820/udp \
 -p 51821:51821/tcp \
 --cap-add=NET_ADMIN \
 --cap-add=SYS_MODULE \
 --sysctl="net.ipv4.conf.all.src_valid_mark=1" \
 --sysctl="net.ipv4.ip_forward=1" \
 weejewel/wg-easy
```

docker 启动rabbitmq

Shell

```shell
docker run -di --name myrabbit -e RABBITMQ_DEFAULT_USER=admin -e RABBITMQ_DEFAULT_PASS=admin -p 15672:15672 -p 5672:5672 -p 25672:25672 -p 61613:61613 -p 1883:1883 rabbitmq:management
```

docker启动mysql

PowerShell

```powershell
# --name指定容器名字 -v目录挂载 -p指定端口映射  -e设置mysql参数 -d后台运行
docker run --name mysql -v /usr/local/mysql/data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=lhc010516  -p 3306:3306 -d mysql:8.0.27
#进入docker里面mysql的目录
docker exec -it mysql /bin/bash
```

docker 启动 redis服务运行容器

Plain Text

```
# 启动redis服务运行容器
docker run --name redis  -v /usr/local/redis/data:/data  -v /usr/local/redis/redis.conf:/usr/local/etc/redis/redis.conf -p 6379:6379 -d redis:latest redis-server /usr/local/etc/redis/redis.conf
```

docker 安装启动Elasticsearch、Kibana

```powershell
# 存储和检索数据
docker pull elasticsearch:7.4.2

# 可视化检索数据
docker pull kibana:7.4.2
# 创建配置文件目录
mkdir -p /mydata/elasticsearch/config

# 创建数据目录
mkdir -p /mydata/elasticsearch/data

# 将/mydata/elasticsearch/文件夹中文件都可读可写
chmod -R 777 /mydata/elasticsearch/

# 配置任意机器可以访问 elasticsearch
echo "http.host: 0.0.0.0" >/mydata/elasticsearch/config/elasticsearch.yml

#启动Elasticsearch
docker run --name elasticsearch -p 9200:9200 -p 9300:9300 \
-e  "discovery.type=single-node" \
-e ES_JAVA_OPTS="-Xms64m -Xmx128m" \
-v /mydata/elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml \
-v /mydata/elasticsearch/data:/usr/share/elasticsearch/data \
-v  /mydata/elasticsearch/plugins:/usr/share/elasticsearch/plugins \
-d elasticsearch:7.4.2



● -p 9200:9200 -p 9300:9300：向外暴露两个端口，9200用于HTTP REST API请求，9300 ES 在分布式集群状态下 ES 之间的通信端口；
● -e  "discovery.type=single-node"：es 以单节点运行
● -e ES_JAVA_OPTS="-Xms64m -Xmx128m"：设置启动占用内存，不设置可能会占用当前系统所有内存
● -v：挂载容器中的配置文件、数据文件、插件数据到本机的文件夹；
● -d elasticsearch:7.6.2：指定要启动的镜像
访问 IP:9200 看到返回的 json 数据说明启动成功。


#设置 Elasticsearch 随Docker启动
# 当前 Docker 开机自启，所以 ES 现在也是开机自启
docker update elasticsearch --restart=always

#启动可视化Kibana

```

```
docker run --name kibana \
-e ELASTICSEARCH_HOSTS=http://110.42.254.129:9200 \
-p 5601:5601 \
-d kibana:7.4.2

#设置 Kibana 随Docker启动
# 当前 Docker 开机自启，所以 kibana 现在也是开机自启
docker update kibana --restart=always
```

docker启动minio

```shell
docker run -d   -p 9000:9000   -p 9001:9001   --name minio   --restart=always   --privileged=true   -v /home/minio/config:/root/.minio   -v /home/minio/data:/data   -e "MINIO_ROOT_USER=miniocong"   -e "MINIO_ROOT_PASSWORD=miniocong"   minio/minio server /data --console-address ":9001" --address ":9000"
```

docker启动Nacos

```shell
docker pull nacos/nacos-server:2.0.3



创建application.properties文件并放入/mydata/nacos/conf/中


=================================配置文件=======================================================

server.servlet.contextPath=/nacos
### Default web server port:
server.port=8848

#*************** Config Module Related Configurations ***************#
### If use MySQL as datasource:
spring.datasource.platform=mysql

### Count of DB:
db.num=1

### Connect URL of DB:
db.url.0=jdbc:mysql://192.168.180.128:3306/nacos_config?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&useUnicode=true&useSSL=false&serverTimezone=UTC
db.user.0=root
db.password.0=root123

### Connection pool configuration: hikariCP
db.pool.config.connectionTimeout=30000
db.pool.config.validationTimeout=10000
db.pool.config.maximumPoolSize=20
db.pool.config.minimumIdle=2

nacos.naming.empty-service.auto-clean=true
nacos.naming.empty-service.clean.initial-delay-ms=50000
nacos.naming.empty-service.clean.period-time-ms=30000


management.metrics.export.elastic.enabled=false
#management.metrics.export.elastic.host=http://localhost:9200

### Metrics for influx
management.metrics.export.influx.enabled=false
#management.metrics.export.influx.db=springboot
```

```

#*************** Access Log Related Configurations ***************#
### If turn on the access log:
server.tomcat.accesslog.enabled=true

### The access log pattern:
server.tomcat.accesslog.pattern=%h %l %u %t "%r" %s %b %D %{User-Agent}i %{Request-Source}i

### The directory of access log:
server.tomcat.basedir=

### The ignore urls of auth, is deprecated in 1.2.0:
nacos.security.ignore.urls=/,/error,/**/*.css,/**/*.js,/**/*.html,/**/*.map,/**/*.svg,/**/*.png,/**/*.ico,/console-ui/public/**,/v1/auth/**,/v1/console/health/**,/actuator/**,/v1/console/server/**

### The auth system to use, currently only 'nacos' and 'ldap' is supported:
nacos.core.auth.system.type=nacos

### If turn on auth system:
nacos.core.auth.enabled=false

### The token expiration in seconds:
nacos.core.auth.default.token.expire.seconds=18000

### The default token:
nacos.core.auth.default.token.secret.key=SecretKey012345678901234567890123456789012345678901234567890123456789

### Turn on/off caching of auth information. By turning on this switch, the update of auth information would have a 15 seconds delay.
nacos.core.auth.caching.enabled=true

### Since 1.4.1, Turn on/off white auth for user-agent: nacos-server, only for upgrade from old version.
nacos.core.auth.enable.userAgentAuthWhite=false

### Since 1.4.1, worked when nacos.core.auth.enabled=true and nacos.core.auth.enable.userAgentAuthWhite=false.
### The two properties is the white list for auth and used by identity the request from other server.
nacos.core.auth.server.identity.key=serverIdentity
nacos.core.auth.server.identity.value=security

#*************** Istio Related Configurations ***************#
### If turn on the MCP server:
```

```
nacos.istio.mcp.server.enabled=false
=============================================配置文件结束===================================================


# 创建logs目录
mkdir -p /mydata/nacos/logs/

# 创建配置文件目录
mkdir -p /mydata/nacos/conf/

#授予权限
chmod 777 /mydata/nacos/logs

chmod 777 /mydata/nacos/conf






docker run --name nacos-server \
-p 8848:8848 \
-p 9848:9848 \
-p 9849:9849 \
--privileged=true \
--restart=always \
-e JVM_XMS=128m \
-e JVM_XMX=128m \
-e MODE=standalone \
-e PREFER_HOST_MODE=hostname \
-v /mydata/nacos/logs:/home/nacos/logs \
-v /mydata/nacos/conf/application.properties:/home/nacos/conf/application.properties \
-d nacos/nacos-server:2.0.3
```

docker启动nginx

```shell
docker run --privileged=true -d --name nginx -p 443:443 -p 80:80 -v /usr/local/nginx/conf/nginx.conf:/etc/nginx/nginx.conf -v /usr/local/nginx/logs:/var/log/nginx -v /usr/local/nginx/html:/usr/share/nginx/html -v /usr/local/nginx/conf.d:/etc/nginx/conf.d -v /home/treehouse:/usr/local/nginx/data nginx
```

删除镜像

docker rmi –f (image id)

批量删除镜像

docker rmi –f $(docker images)

#docker ps 命令

#列出当前正在运行的容器

–a#列出当前正在运行的容器+带出历史运行的容器

–n=? #列出最近创建的容器

–q #只显示容器的编号

退出容器

exit *直接退出容器*

Ctrl + P + Q *容器不停止退出*

删除容器

docker rm 容器id #删除指定的容器，不能删除正在运行的容器，强制运行 rm –f

docker rm –f $(docker ps –aq) #删除所有容器

启动和停止容器的操作

```
docker start 容器id # 启动容器
docker restart 容器id # 重启容器
docker stop 容器id # 停止当前正在运行的容器
docker kill 容器id #强制停止当前的容器
```

常用的命令

后台启动容器

Plain Text

```
docker run -d 镜像名称 #问题docker ps 发现 后台镜像停止了
#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发现没有应用就会停止
#nginx 容器启动后，发现自己没有提供服务，就会立刻停止，就是没有程序了
```

查看日志

Plain Text

```
#显示日志
-tf  #显示日志
-n/--tail number #显示日志的条数
-f #保持打印窗口持续打印
-t #日志加时间
docker logs -tf -n 10 容器id
```

查看容器中进程信息

docker top 容器id

查看镜像元数据

```
#命令 docker inspect 容器id


#测试
docker inspect a1bd0008d904
[
    {
        "Id": "a1bd0008d904f20d5b7025cd05d47b35362406d2fa2fef8a72d6b9ff03dce587",
        "Created": "2021-10-27T09:07:47.833653838Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "redis-server",
            "/usr/local/etc/redis/redis.conf"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 11902,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2021-10-27T09:09:57.155577898Z",
            "FinishedAt": "2021-10-27T09:09:56.791361947Z"
        },
        "Image": "sha256:7faaec68323851b2265bddb239bd9476c7d4e4335e9fd88cbfcc1df374dded2f",
        "ResolvConfPath": "/var/lib/docker/containers/a1bd0008d904f20d5b7025cd05d47b35362406d2fa2fef8a72d6b9ff03dce587/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/a1bd0008d904f20d5b7025cd05d47b35362406d2fa2fef8a72d6b9ff03dce587/hostname",
        "HostsPath": "/var/lib/docker/containers/a1bd0008d904f20d5b7025cd05d47b35362406d2fa2fef8a72d6b9ff03dce587/hosts",
        "LogPath": "/var/lib/docker/containers/a1bd0008d904f20d5b7025cd05d47b35362406d2fa2fef8a72d6b9ff03dce587/a1bd0008d904f20d5b7025cd05d47b35362406d2fa2fef8a72d6b9ff03dce587-json.log",
        "Name": "/redis",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": [
            "2bbe0f50434b250da9a83ebef9218d276d72564d1eb4d7770ae635ae6ea97946"
        ],
        "HostConfig": {
            "Binds": [
                "/usr/local/redis/data:/data",
                "/usr/local/redis/redis.conf:/usr/local/etc/redis/redis.conf"
            ],
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {
                "6379/tcp": [
                    {
                        "HostIp": "",
                        "HostPort": "6379"
                    }
                ]
            },
            "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/overlay2/11f69cce2beeeb58ef4812ffd5072fc0e517eb99d31fefd08010cfc72d711c76-init/diff:/var/lib/docker/overlay2/748ef5a20ad62f0c128a1697f9ce5f0250fdeecc1546484aa84a9f726cbf8c51/diff:/var/lib/docker/overlay2/0fc31f056976345d445f474756586df4cb36d8389503df58f591720a21a841b6/diff:/var/lib/docker/overlay2/dbfc1f1c36b270653d0abedd4d5f88f466454cfcd5ce5f7ca5096c3ccadc76ee/diff:/var/lib/docker/overlay2/05ae818be6b435fdeee92ddd729ba80a98a996960591dca0136a98d1a1317f03/diff:/var/lib/docker/overlay2/689212ed08c74ca383706529458744a1f95e021ff13fc559e42e85a8dcb90325/diff:/var/lib/docker/overlay2/0c20e25349bc0f04e0cfff725fbe00ce6935015a09ea0408fd577e11e9faf2da/diff",
                "MergedDir": "/var/lib/docker/overlay2/11f69cce2beeeb58ef4812ffd5072fc0e517eb99d31fefd08010cfc72d711c76/merged",
                "UpperDir": "/var/lib/docker/overlay2/11f69cce2beeeb58ef4812ffd5072fc0e517eb99d31fefd08010cfc72d711c76/diff",
                "WorkDir": "/var/lib/docker/overlay2/11f69cce2beeeb58ef4812ffd5072fc0e517eb99d31fefd08010cfc72d711c76/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [
            {
                "Type": "bind",
                "Source": "/usr/local/redis/data",
                "Destination": "/data",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            },
            {
                "Type": "bind",
                "Source": "/usr/local/redis/redis.conf",
                "Destination": "/usr/local/etc/redis/redis.conf",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
        "Config": {
            "Hostname": "a1bd0008d904",
            "Domainname": "",
```

进入当前正在运行的容器

```
# 我们通常容器都是使用后台方式进行的，需要进入容器，修改一些配置

#命令
#方式一
docker exec -it容器id /bin/bash
#方式二
docker attach 容器id#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上

```
docker cp 容器id:容器内路径 目的主机路径
#test

[root@VM-0-4-centos ~]# docker exec -it mysql /bin/bash
root@0cd4debcb955:/# cd home
root@0cd4debcb955:/home# ls
root@0cd4debcb955:/home# touch cong.java
root@0cd4debcb955:/home# exit;
exit
[root@VM-0-4-centos ~]# doucker ps
-bash: doucker: command not found
[root@VM-0-4-centos ~]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS         PORTS                                                  NAMES
a1bd0008d904   redis:latest   "docker-entrypoint.s…"   26 hours ago   Up 26 hours   0.0.0.0:6379->6379/tcp, :::6379->6379/tcp              redis
0cd4debcb955   mysql:8.0.27   "docker-entrypoint.s…"   26 hours ago   Up 26 hours   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql
[root@VM-0-4-centos ~]# docker cp 0cd4debcb955:/home/cong.java /home
[root@VM-0-4-centos ~]# cd /home
[root@VM-0-4-centos home]# ls
cong  cong.java
```

docker 安装tomcat

```
docker run -it --rm tomcat:9.0
#我们之前启动的都是后台，停止了容器还是可以查到，docker run -it --rm 用来测试，用完就删除
```

docker 安装ES镜像

```
docker pull nshou/elasticsearch-kibana
#最后咱们把镜像启动为容器就可以了，端口映射保持不变，咱们给这个容器命名为eskibana，到这里ES和Kibana就安装配置完成了！容器启动后，它们也就启动了，一般不会出错，是不是非常方便？节省大把时间放到开发上来，这也是我一直推荐docker的原因。

docker run -d -p 9200:9200 -p 9300:9300 -p 5601:5601 -e ES_JAVA_OPTS="-Xms64m -Xmx512m" --name eskibana  nshou/elasticsearch-kibana



#咱们还需要安装ElasticSearch Head，它相当于是ES的图形化界面，这个更简单，它是一个浏览器的扩展程序，直接在chrome浏览器扩展程序里下载安装即可：


#打开chrome浏览器，在扩展程序chrome应用商店那里，搜索elasticsearch：

#-e 环境的修改
```

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]
```

容器数据卷

```
#使用命令来加载 -v
docker run -it -v 主机目标：容器内目录
#通过docker inspect 容器id 查看详细信息

"Mounts": [
            {
                "Type": "bind",
                "Source": "/usr/local/nginx/conf/nginx.conf",
                "Destination": "/etc/nginx/nginx.conf",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            },
            {
                "Type": "bind",
                "Source": "/usr/local/nginx/logs",
                "Destination": "/var/log/nginx",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            },
            {
                "Type": "bind",
                "Source": "/usr/local/nginx/html",
                "Destination": "/usr/share/nginx/html",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
#如何确定是具名挂载还是匿名挂着，还是指定路径挂载
-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
```

DockerFile

Dockerfile就是用来构建docker镜像的构建文件 --命令脚本

通过这个脚本可以生成镜像，镜像是一层层的，脚本一个个的命令，每个命令都是一层

```
docker build -f 脚本路径 -t 镜像id .
#test
[root@VM-0-4-centos ~]# docker build -f /root/dockerfile1 -t cong/docker:1.0 .
Sending build context to Docker daemon  45.06kB
Step 1/4 : FROM nginx
 ---> 87a94228f133
Step 2/4 : VOLUME ["volume01","volume02"]
 ---> Running in ce644543049b
Removing intermediate container ce644543049b
 ---> 5dd60f570667
Step 3/4 : CMD echo "----end----"
 ---> Running in b254c1bc9348
Removing intermediate container b254c1bc9348
 ---> d67eef784cbe
Step 4/4 : CMD /bin/bash
 ---> Running in c17b47fab41f
Removing intermediate container c17b47fab41f
 ---> b147d870887b
Successfully built b147d870887b
Successfully tagged cong/docker:1.0
[root@VM-0-4-centos ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
cong/docker   1.0       b147d870887b   3 seconds ago    133MB
mysql         8.0.27    ecac195d15af   11 days ago      516MB
redis         latest    7faaec683238   2 weeks ago      113MB
nginx         latest    87a94228f133   2 weeks ago      133MB
```

两个容器实现数据同步

```
docker run ..... -- volums -from 容器名称 镜像名称
容器间的信息传递  数据卷容器的生命周期一直持续到没有容器使用为止。
```

DockerFile构建过程

基础知识：

1 每个保留关键字（指令）都是必须是大写字母

2 执行从上到下顺序执行

3 #表示注释

4 每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，就需要编写dockerfile文件

```
FROM        #基础镜像，一切从这里开始构建
MAINTAINER  #镜像是谁写的，名字+邮箱
RUN        #镜像构建的时候需要运行的命令
ADD        #步骤：tomcat镜像，这个tomcat的压缩包！添加内容
WORKDIR     #镜像的工作目录
VOLUME      #挂载的目录
EXPOSE      #暴露端口配置
CMD        #指定容器启动的时候要运行的命令，只有最后一个会生效，可被替代
ENTRYPOINT  #指定这个容器启动的时候要运行的命令，可以直接追加命令
ONBUILD     #当构建一个被继承 DockerFile这个时候就会运行 ONBUILD 的指令。触发指令
COPY       #类似ADD，将我们文件拷贝到镜像中
ENV        #构建的时候设置环境变量
```

实战测试

Docker Hub 中99%镜像都是从这个基础镜像过来的FROM scratch，然后配置需要的软件和配置

```
#1  编写dockerfile文件
[root@VM-0-4-centos dockerfile]# vim mydockerfile
[root@VM-0-4-centos dockerfile]# cat mydockerfile
FROM centos
MAINTAINER cong<771901874@qq.com>

EVA 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
docker build -f mydockerfile -t mycentos:0.1 .
```

cmd 和 entrypointd的区别

```
CMD        #指定容器启动的时候要运行的命令，只有最后一个会生效，可被替代
ENTRYPOINT   #指定这个容器启动的时候要运行的命令，可以直接追加命令
```

实战：Tomcat

1 准备镜像文件 tomcat压缩包，jdk的压缩包

2 编写dockerfile文件，官方命名DockerFile

发布自己的镜像

DockerHub

1地址：https://hub.docker.com/注册自己的账号

2 确定这个账号可以启动

3 在我们的服务器上提交镜像

4登录完毕后就可以提交镜像，就是一步 docker push

```
[root@VM-0-4-centos dockerfile]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: lhcong
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




[root@VM-0-4-centos dockerfile]# docker push cong/docker:1.0
The push refers to repository [docker.io/cong/docker]
9959a332cf6e: Preparing
f7e00b807643: Preparing
f8e880dfc4ef: Preparing
788e89a4d186: Preparing
43f4e41372e4: Preparing
e81bff2725db: Waiting
denied: requested access to the resource is denied #拒绝

#解决方法
[root@VM-0-4-centos ~]# docker tag b147d870887b lhcong/docker:1.0
[root@VM-0-4-centos ~]# docker rmi cong/docker:1.0
Untagged: cong/docker:1.0
[root@VM-0-4-centos ~]# docker iamges
docker: 'iamges' is not a docker command.
See 'docker --help'
[root@VM-0-4-centos ~]# docker images
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
lhcong/docker   1.0       b147d870887b   19 hours ago   133MB
mysql           8.0.27    ecac195d15af   12 days ago    516MB
redis           latest    7faaec683238   2 weeks ago    113MB
nginx           latest    87a94228f133   2 weeks ago    133MB
#易错点：docker push 后面必须加的是用户id/文件名字:tag 不然会报错
要用docker tag 命令来修改
```

阿里云镜像服务

1登录阿里云

2找到容器镜像服务

3创建命名空间

4创建容器镜像

```
1docker login --username=congdoraemon registry.cn-beijing.aliyuncs.com
2docker tag [ImageId] registry.cn-beijing.aliyuncs.com/lhcong-docker/cong-test:[镜像版本号]
3 docker push registry.cn-beijing.aliyuncs.com/lhcong-docker/cong-test:[镜像版本号]
```

Docker网络

理解Docker

```
#查看容器的内部网络地址 ip addr

docker exec -it 125227f233a9 ip addr

#若有报错在容器内执行下面的代码

apt update && apt install -y iproute2

#我们发现这个容器带来的网卡，都是一对对的
#ve th-pair 就是一对的虚拟设备接口，他们都是成对出现的，一段连着协议，一段彼此相连
```

docker使用的是linux的桥接

--link

自定义网络

查看所有的docker网络

[root@VM-0-4-centos ~]# docker network lsNETWORK ID NAME DRIVER SCOPE3fd008702217 bridge bridge local2e08eb3e1445 host host localf1063c44c7b8 none null local

网络模式

bridge :桥接 docker（默认，自己的创建也是用bridge模式）

none ： 不配置网络

host ： 和宿主机共享网络

container：容器网络连同！ （用的少！局限很大）

```
#我们直接启动的命令 默认自带--net bridge ,而这个就是我的docker0

docker0特点：默认，域名不能访问， --link可以打通连接

docker network create --driver bridge
#test


[root@VM-0-4-centos ~]# docker network create --driver bridge --subnet 192.168.0.0/16 --gateway 192.168.0.1 mynet
f16708ce675cc5b9e3836c0052fba7a78b68ca71ad2d9d67feb4a5a4646921c3


[root@VM-0-4-centos ~]# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3fd008702217   bridge    bridge    local
2e08eb3e1445   host      host      local
f16708ce675c   mynet     bridge    local
f1063c44c7b8   none      null      local
```

好处：

redis -不同集群使用不同的网络，保证集群是安全和健康的

mysql-不同集群使用不同的网络，保证集群是安全和健康的

网络连通

#一个容器两个ip，打通。阿里云服务器： 公网ip 私网ip

redis集群部署

SpringBoot微服务打包Docker镜像

```
[root@VM-0-4-centos idea]# docker build -t cong666 .
Sending build context to Docker daemon  17.49MB
Step 1/5 : FROM java:17
manifest for java:17 not found: manifest unknown: manifest unknown
[root@VM-0-4-centos idea]# docker build -t cong666 .
Sending build context to Docker daemon  17.49MB
Step 1/5 : FROM java:8
8: Pulling from library/java
5040bd298390: Pull complete
fce5728aad85: Pull complete
76610ec20bf5: Pull complete
60170fec2151: Pull complete
e98f73de8f0d: Pull complete
11f7af24ed9c: Pull complete
49e2d6393f32: Pull complete
bb9cdec9c7f3: Pull complete
Digest: sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66b7f8c0e9d
Status: Downloaded newer image for java:8
 ---> d23bdf5b1b1b
Step 2/5 : COPY *.jar /app.jar
 ---> dfddcc85166d
Step 3/5 : CMD ["--server.port=8080"]
 ---> Running in 8a08f7a296f4
Removing intermediate container 8a08f7a296f4
 ---> 04f26d611dda
Step 4/5 : EXPOSE 8080
 ---> Running in 41a1d76e0b20
Removing intermediate container 41a1d76e0b20
 ---> fbaa288a1dfc
Step 5/5 : ENTRYPOINT ["java","-jar","/app.jar"]
 ---> Running in ad8c17c24c1c
Removing intermediate container ad8c17c24c1c
 ---> a8a9c7dce586
Successfully built a8a9c7dce586
Successfully tagged cong666:latest
[root@VM-0-4-centos idea]# docker images
REPOSITORY                                               TAG       IMAGE ID       CREATED          SIZE
cong666                                                  latest    a8a9c7dce586   45 seconds ago   661MB
registry.cn-beijing.aliyuncs.com/lhcong-docker/cong-test   1.0       b147d870887b   29 hours ago     133MB
lhcong/docker                                            1.0       b147d870887b   29 hours ago     133MB
```

```
mysql                                                         8.0.27    ecac1
95d15af   12 days ago      516MB
redis                                                         latest    7faae
c683238   2 weeks ago      113MB
nginx                                                         latest    87a94
228f133   2 weeks ago      133MB
java                                                          8         d23bd
f5b1b1b   4 years ago      643MB
[root@VM-0-4-centos idea]# docker run -d -P --name cong-springboot-web con
g666
a21577023b12dba18ffd4aa2ed3e254d5f628f76343df6892784067c3b4dba11
[root@VM-0-4-centos idea]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
  STATUS              PORTS
a21577023b12   cong666        "java -jar /app.jar …"   About a minute ago
  Up About a minute   0.0.0.0:49153->8080/tcp, :::49153->808
125227f233a9   nginx          "/docker-entrypoint.…"   2 days ago
  Up 2 days           0.0.0.0:8080->80/tcp, :::8080->80/tcp
a1bd0008d904   redis:latest   "docker-entrypoint.s…"   4 days ago
  Up 4 days           0.0.0.0:6379->6379/tcp, :::6379->6379/
0cd4debcb955   mysql:8.0.27   "docker-entrypoint.s…"   4 days ago
  Up 4 days           0.0.0.0:3306->3306/tcp, :::3306->3306/
[root@VM-0-4-centos idea]# curl localhost:49153
{"timestamp":"2021-10-31T13:34:09.790+00:00","status":404,"error":"Not Fou
nd","path":"/"}[root@VM-0-4-centos idea]#
[root@VM-0-4-centos idea]# curl localhost:49153/hello
Hello cong
```

Docker Compose

简介

Docker Compose来轻松高效的管理容器，定义运行多个容器

docker-compose up来启动

Cmpose: 重要的概念

服务service,容器。应用。

项目project。一组关联的容器

Compose的安装

```
#国外的地址比较慢
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
#国内的源比较快
sudo curl -L https://get.daocloud.io/docker/compose/releases/download/1.29.2/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
#赋予权限
chmod +x docker-compose
```

体验

1、应用 app.py

2、Dockerfile 应用打包为镜像

3、Docker-compose yaml文件（定义整个服务，需要的环境。web、redis）完整的上线服务

4、启动compose项目（docker-compose up）

5 、停止项目 docker-compose down ctrl+c

小结

1、Docker镜像。 run=》容器

2、DockerFile构建镜像（服务打包）

3、docker-compose启动项目（编排、多个微服务、环境）

yaml规则

docker-compose.yaml 核心。！

```
#3层!
version: ''#版本
services: #服务
  服务1: web
  #服务配置
  images
  build
  network
  ....
  服务2: redis
  ....
  #其他配置  网络/卷、全局规则
  volumes:
  networks:
  configs:
```

博客

1、下载项目（docker-compose.yaml）

2、如果需要文件。dockerfile

3、文件准备齐全（直接一键启动项目）

总结

工程、compose、容器

项目compose：三层

- 工程project
- 服务 服务
- 容器 运行实例！ docker k8s 容器

Docker Swarm

docker swarm init --advertisr-addr ip

步骤

1 、生成主节点

2、加入（管理者、worker）

Raft协议

双主双从：假设一个节点挂了，其他节点是否可以用

Raft：保证大多数节点存活才可以用。只要>1,集群至少大于三台

```
docker run 容器启动！不具有扩缩容
docker service 服务！ 具有扩容容器，滚动更新（需要做swarm集群）
```

服务、集群中任意节点都可以访问。服务可以有多个副本动态扩容实现高可用

概念总结

swarm

集群的管理和编号。docker可以初始化一个swarm集群，其他节点可以加入。（管理，工作者）

Node

就是一个docker节点。多个节点就组成了一个网络集群（管理 ，工作者）

Service

任务，可以在管理节点或者工作节点来运行。核心！用户访问！

Task

容器内的命令，细节任务