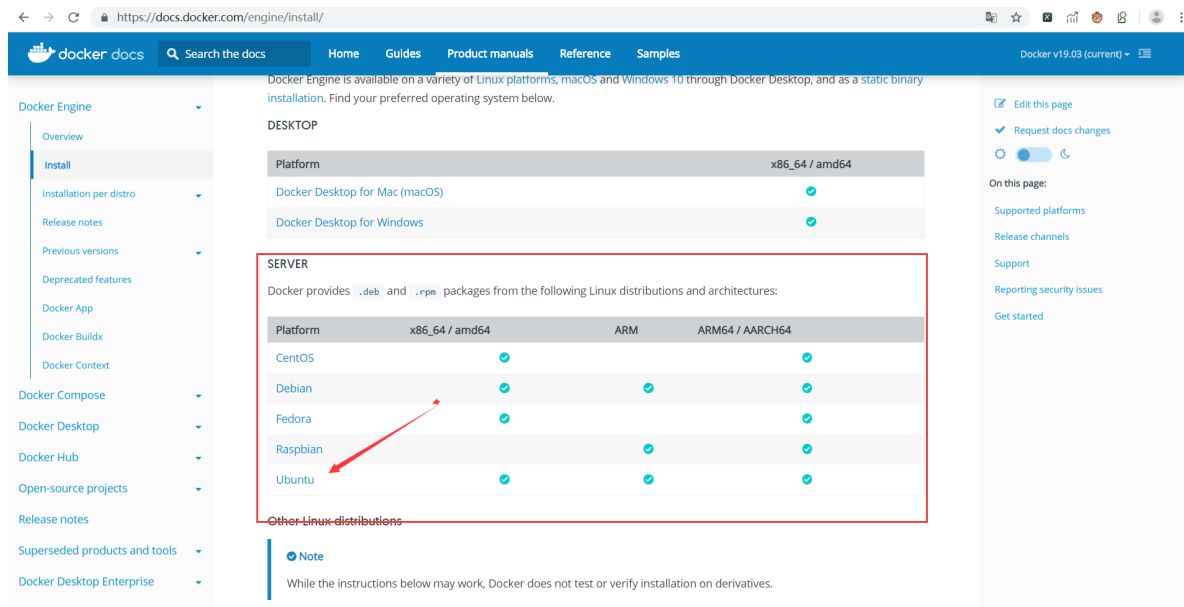


Docker的安装

docker官网下载文档: <https://docs.docker.com/engine/install/>

官网即有详细步骤



```
$ sudo apt-get update

$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common
```

换镜像网站, 阿里云镜像~, 百度即有

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

这样子就下载完毕了~

Docker的常用命令

帮助命令

```
docker version    # 显示docker版本信息
docker info       #显示docker的系统信息，包括镜像和容器的数量
docker 命令 --help #帮助命令
```

帮助文档的地址: <https://docs.docker.com/engine/reference/commandline/>

镜像命令

查看镜像

```
docker images    #查看所有本地的主机上的镜像
```

可添加以下参数

名称, 简写	默认	描述
<code>--all , -a</code>		显示所有图像 (默认隐藏中间图像)
<code>--digests</code>		显示摘要
<code>--filter , -f</code>		根据提供的条件过滤输出
<code>--format</code>		使用Go模板打印漂亮的图像
<code>--no-trunc</code>		不要截断输出
<code>--quiet , -q</code>		仅显示数字ID

```
(Django) ubuntu@VM-0-11-ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest             bf756fblae65       5 months ago       13.3kB
(Django) ubuntu@VM-0-11-ubuntu:~$
```

- REPOSITORY 镜像的仓库源
- TAG 镜像的标签
- IMAGE ID 镜像的ID
- CREATED 镜像创建时间
- SIZE 镜像的大小

搜索镜像

```
docker search    #搜索镜像
```

名称, 简写	默认	描述
<code>--automated</code>		不推荐使用 仅显示自动构建
<code>--filter , -f</code>		根据提供的条件过滤输出
<code>--format</code>		使用Go模板进行漂亮的打印搜索
<code>--limit</code>	25	最多搜索结果数
<code>--no-trunc</code>		不要截断输出
<code>--stars , -s</code>		不推荐使用 仅显示至少x个星标

例子:

```
docker search --filter=STARS=3000 #只显示收藏数大于3000的镜像
```

下载镜像

```
docker pull #下载镜像
```

名称, 简写	默认	描述
<code>--all-tags , -a</code>		下载存储库中所有标记的图像
<code>--disable-content-trust</code>	true	跳过图像验证
<code>--platform</code>		实验 (守护程序) API 1.32+ 如果服务器支持多平台, 则设置平台
<code>--quiet , -q</code>		禁止详细输出

例子如下:

```
docker pull mysql:版本号 #不指定版本就默认最新
```

删除镜像

```
docker rmi -f 镜像id ... #删除指定id的镜像 可以指定多个
```

```
docker rmi -f $(docker images -aq) #删除全部镜像
```

容器命令

下载一个centos容器

```
docker pull centos
```

新建容器并且启动

```
docker run [参数] image
```

#参数

--name="容器名字"

-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 [port] #指定容器端口

-p ip:主机端口:容器端口

-p 主机端口:容器端口（常用）

测试，启动并进入容器

```
(Django) ubuntu@VM-0-11-ubuntu:~$ docker run -it centos /bin/bash
[root@42d35bbf701f /]#
```

退出容器，返回主机

- 停止并退出

```
[root@42d35bbf701f /]# exit
exit
(Django) ubuntu@VM-0-11-ubuntu:~$
```

- 退出不停止

Ctrl+P+Q #退出不停止

查看正在运行的容器

```
docker ps #查看正在运行的容器
docker ps -a #查看曾经运行和正在运行的容器
docker ps -q #只显示容器id
```

删除容器

```
docker rm 容器id #删除指定容器，不能删除正在运行，需要的话加 -f
docker rm -f $(docker ps -aq) #删除所有容器
docker ps -a -q|xargs docker rm #删除所有容器
```

启动和停止容器

```
docker start 容器id      #启动容器
docker restart 容器id    #重启容器
docker stop 容器id       #停止容器
docker kill 容器id       #强制杀掉容器
```

常用其它命令

后台启动容器

```
docker run -d [镜像名字] #必须有前台进程，不然docker会停止这个容器
```

查看日志

```
docker logs 容器id
```

名称, 简写	默认	描述
<code>--details</code>		显示提供给日志的其他详细信息
<code>--follow, -f</code>		跟踪日志输出
<code>--since</code>		显示自时间戳以来的日志 (例如2013-01-02T13: 23: 37) 或相对记录 (例如42m的42分钟)
<code>--tail</code>	all	从日志末尾开始显示的行数
<code>--timestamps, -t</code>		显示时间戳
<code>--until</code>	API 1.35以上	在时间戳 (例如2013-01-02T13: 23: 37) 或相对 (例如42m持续42分钟) 之前显示日志

查看容器中的进程信息

```
docker top 容器id
```

查看容器的元数据

```
docker inspect 容器id
(Django) ubuntu@VM-0-11-ubuntu:~$ docker inspect 42d35bbf701f
[
  {
    "Id":
    "42d35bbf701f860a6c41c317a3f8ba505dd7e51131d032d82972f84c50edf07b",
    "Created": "2020-06-22T09:31:27.601199783Z",
    "Path": "/bin/bash",
    "Args": [],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
```

```
        "Pid": 0,
        "ExitCode": 0,
        "Error": "",
        "StartedAt": "2020-06-22T09:31:28.311467148Z",
        "FinishedAt": "2020-06-22T09:32:58.279179076Z"
    },
    "Image":
    "sha256:831691599b88ad6cc2a4abbd0e89661a121aff14cfa289ad840fd3946f274f1f",
    "ResolveConfPath":
    "/var/lib/docker/containers/42d35bbf701f860a6c41c317a3f8ba505dd7e51131d032d82972f84c50edf07b/resolve.conf",
    "HostnamePath":
    "/var/lib/docker/containers/42d35bbf701f860a6c41c317a3f8ba505dd7e51131d032d82972f84c50edf07b/hostname",
    "HostsPath":
    "/var/lib/docker/containers/42d35bbf701f860a6c41c317a3f8ba505dd7e51131d032d82972f84c50edf07b/hosts",
    "LogPath":
    "/var/lib/docker/containers/42d35bbf701f860a6c41c317a3f8ba505dd7e51131d032d82972f84c50edf07b/42d35bbf701f860a6c41c317a3f8ba505dd7e51131d032d82972f84c50edf07b-
    json.log",
    "Name": "/vigorous_mclaren",
    "RestartCount": 0,
    "Driver": "overlay2",
    "Platform": "linux",
    "MountLabel": "",
    "ProcessLabel": "",
    "AppArmorProfile": "docker-default",
    "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,
        "Capabilities": null,
        "Dns": [],
        "DnsOptions": [],
        "DnsSearch": [],
        "ExtraHosts": null,
        "GroupAdd": null,
        "IpcMode": "private",
        "Cgroup": "",
        "Links": null,
        "OomScoreAdj": 0,
        "PidMode": "",
```

```
"Privileged": false,
"PublishAllPorts": false,
"ReadonlyRootfs": false,
"SecurityOpt": null,
"UTSMode": "",
"UsersnsMode": "",
"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/overlay2/6d069fdf8f59b9f40d9b58a8763be9d175388eb1a2fba96c027c13c3afd14030-init/diff:/var/lib/docker/overlay2/48350764eff43bc5420b124476024e2e711b5af19886a11f68728ff5a276f95a/diff",
        "MergedDir":
"/var/lib/docker/overlay2/6d069fdf8f59b9f40d9b58a8763be9d175388eb1a2fba96c027c13c3afd14030/merged",
        "UpperDir":
"/var/lib/docker/overlay2/6d069fdf8f59b9f40d9b58a8763be9d175388eb1a2fba96c027c13c3afd14030/diff",
        "WorkDir":
"/var/lib/docker/overlay2/6d069fdf8f59b9f40d9b58a8763be9d175388eb1a2fba96c027c13c3afd14030/work"
    },
    "Name": "overlay2"
},
"Mounts": [],
"Config": {
    "Hostname": "42d35bbf701f",
    "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": "20200611",
        "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":
"9cbf4bd004965b0a688a82a3b6c050e40eb237d0e97cf13a7c21513d3d8f80b4",
        "HairpinMode": false,
        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "Ports": {},
        "SandboxKey": "/var/run/docker/netns/9cbf4bd00496",
        "SecondaryIPAddresses": null,
        "SecondaryIPv6Addresses": null,
        "EndpointID": "",
        "Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "MacAddress": "",
        "Networks": {
            "bridge": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": null,
                "NetworkID":
"7e5669cd2425fe9d76f4d09d96c70dfb4d873d3006e7d7fb8f0240f79874aa01",
                "EndpointID": "",
                "Gateway": "",
                "IPAddress": "",
                "IPPrefixLen": 0,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "",
                "DriverOpts": null
            }
        }
    }
}
]

```

进入正在运行的容器

`docker exec -it 容器id /bin/bash` #进入容器开启一个新的终端

`docker attach` #进入容器正在执行的终端

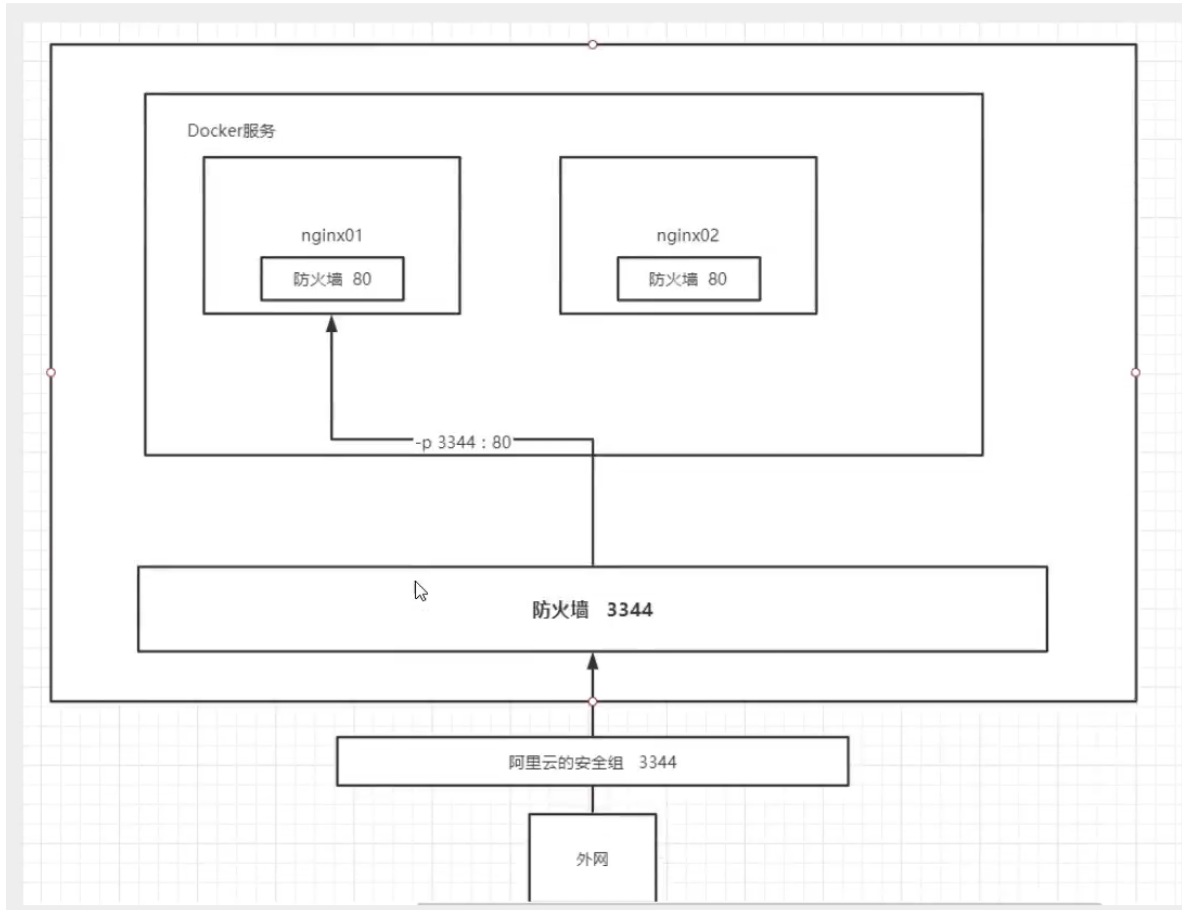
从容器拷贝文件到主机

`docker cp` 容器id:容器内文件路径 要存的主机的路径

Docker使用步骤

安装Nginx

1. 搜索镜像，可以去hub.docker.com查看 或者 docker search nginx
2. 下载 docker pull nginx
3. docker run -d --name nginx01 -p 3344:80 nginx
-d 后台运行 --name 命名 -p: 3344为主机端口，80为容器内端口



安装tomcat

```
docker run -it --rm tomcat:9.0  
--rm #用完即删，可用于测试
```

发现tomcat是被阉割的，Linux命令少了，webapp没了

可视化


- portainer

Docker图形化界面管理工具！提供后台模板供操作

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

直接访问 服务器ip:8088

打开设置密码，创建用户



Please create the initial administrator user.

Username


Password

Confirm password

✖ The password must be at least 8 characters long

[+ Create user](#)

选择本地，点击连接




Connect Portainer to the Docker environment you want to manage.

**Local**
Manage the local Docker environment

**Remote**
Manage a remote Docker environment

**Agent**
Connect to a Portainer agent

**Azure**
Connect to Microsoft Azure ACI

Information

Manage the Docker environment where Portainer is running.

⚠ Ensure that you have started the Portainer container with the following Docker flag:



`-v "/var/run/docker.sock:/var/run/docker.sock"` (Linux).

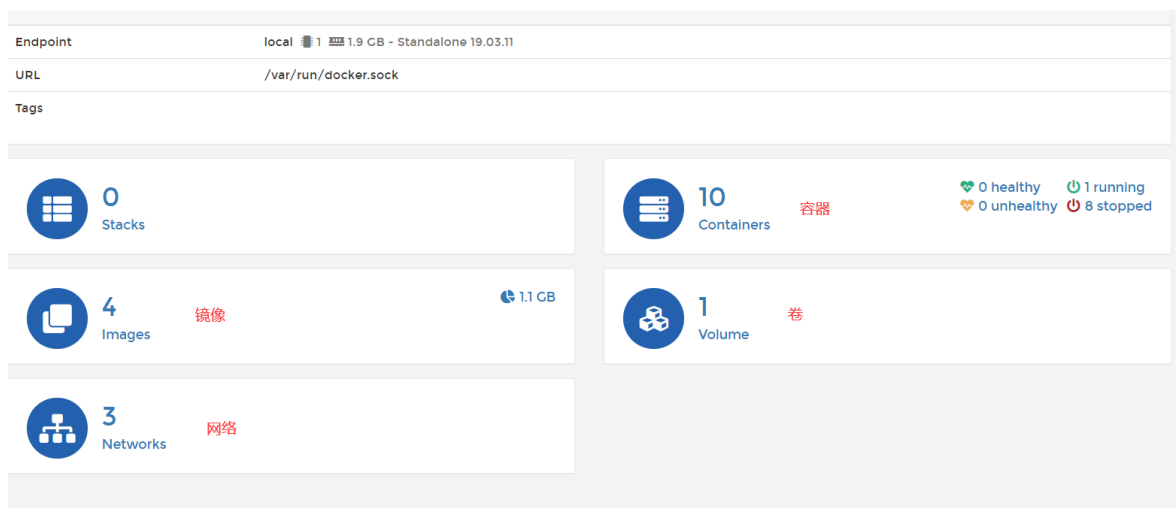
or

`-v \\.\pipe\docker_engine:\\.\pipe\docker_engine` (Windows).

[⚡ Connect](#)

点击进入本地管理面板

Endpoints		
Refresh		
🔍 Search by name, group, tag, status, URL...		
	local up 2020-06-22 20:05:55	Group: Unassigned ✎
	0 stacks 9 containers - 1 up 8 / 0 down 0 1 volume 4 images 1.9 GB - No tags	Standalone 19.03.11 /var/run/docker.sock
	local up 2020-06-22 20:08:41	Group: Unassigned ✎
	0 stacks 9 containers - 1 up 8 / 0 down 0 1 volume 4 images 1.9 GB - No tags	Standalone 19.03.11 /var/run/docker.sock
		Items per page 10



commit镜像

```
docker commit -m="描述" -a="作者" 容器id 目标镜像名:版本号
```

例子:

```
docker commit -m="add webapps" -a="hzh" f96291cf94a4 tomcat02:1.0
```

容器数据卷

容器的持久化和同步操作! 容器之间的数据也可以共享

方式一: 使用命令来挂载 -v

```
docker run -it -v -p 主机目录路径:容器内目录路径
```

测试:

```
docker run -it -v /home/ceshi:/home centos /bin/bash
```

```
{
  "Mounts": [
    {
      "Type": "bind",
      "Source": "/home/ceshi", 主机内的地址
      "Destination": "/home", docker容器内
      "Mode": "",
      "RW": true,
      "Propagation": "rprivate"
    }
  ]
}
```

查看挂载

```
docker volume ls
```

```
docker run -d -P -v 卷名:容器内路径 镜像名
```

具名挂载，如上面这样写，没有指定主机路径的，默认放在主机

```
/var/lib/docker/volume/挂载名/_data
```

```
-v 容器内路径 #匿名挂载  
-v 卷名:容器内路径 #具名挂载  
-v 主机路径:容器内路径 #指定路径挂载
```

扩展：

容器权限管理：ro只能通过主机操作，容器无法修改。

```
docker run -d -P -v 卷名:容器内路径:ro 镜像名 #只读  
docker run -d -P -v 卷名:容器内路径:rw 镜像名 #可读可写！
```

安装MySQL

```
docker run -d -p 3333:3306 -v /home/ubuntu/mysql/conf:/etc/mysql/conf.d -v  
/home/ubuntu/mysql/data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 --name  
mysql01 mysql:5.7
```

```
-d 后台运行  
-p 端口映射  
-v 卷挂载  
-e 环境配置  
--name 容器名字
```

启动后 用服务器3333端口连接MySQL数据库

容器间的数据共享(备份)

```
--volumes-from 父容器 镜像id
```

```

[root@kuangshen /]# docker run -it --name docker03 --volumes-from docker01 kuangshen/centos:1.0
[root@b7481f4ab655 /]# ls -l
total 56
lrwxrwxrwx 1 root root 7 May 11 2019 bin -> usr/bin
drwxr-xr-x 5 root root 360 May 15 12:22 dev
drwxr-xr-x 1 root root 4096 May 15 12:22 etc
drwxr-xr-x 2 root root 4096 May 11 2019 home
lrwxrwxrwx 1 root root 7 May 11 2019 lib -> usr/lib
lrwxrwxrwx 1 root root 9 May 11 2019 lib64 -> usr/lib64
drwx----- 2 root root 4096 Jan 13 21:48 lost+found
drwxr-xr-x 2 root root 4096 May 11 2019 media
drwxr-xr-x 2 root root 4096 May 11 2019 mnt
drwxr-xr-x 2 root root 4096 May 11 2019 opt
dr-xr-xr-x 122 root root 0 May 15 12:22 proc
dr-xr-xr-x 2 root root 4096 Jan 13 21:49 root
drwxr-xr-x 11 root root 4096 Jan 13 21:49 run
lrwxrwxrwx 1 root root 8 May 11 2019 sbin -> usr/sbin
drwxr-xr-x 2 root root 4096 May 11 2019 srv
dr-xr-xr-x 13 root root 0 Mar 23 14:00 sys
drwxrwxrwt 7 root root 4096 Jan 13 21:49 tmp
drwxr-xr-x 12 root root 4096 Jan 13 21:49 usr
drwxr-xr-x 20 root root 4096 Jan 13 21:49 var
drwxr-xr-x 2 root root 4096 May 15 12:20 volume01
drwxr-xr-x 2 root root 4096 May 15 12:18 volume02
[root@b7481f4ab655 /]# cd volume01
[root@b7481f4ab655 volume01]# ls
docker01
[root@b7481f4ab655 volume01]# touch docker03
[root@b7481f4ab655 volume01]# ls
docker01 docker03
[root@b7481f4ab655 volume01]#

```

只要通过它我们就可以实现容器间的数据共享了

容器之间数据共享，删除父容器，其它容器数据依旧存在！其实是复制的共享卷~



多个MySQL实现数据共享

```

docker run -d -p 3333:3306 -v /home/ubuntu/mysql/conf:/etc/mysql/conf.d -v
/home/ubuntu/mysql/data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 --name
mysql01 mysql:5.7

```

```

docker run -d -p 3334:3306 -e MYSQL_ROOT_PASSWORD=123456 --name mysql02 --
volumes-from mysql01 mysql:5.7

```

要直到所有共享卷的容器都关闭，才可能结束数据卷的生命周期

DockerFile

DockerFile介绍

dockerfile是用来构建docker镜像的文件!命令参数脚本!

1. 编写dockfile文件
2. docker build 构建成一个镜像
3. docker run 运行镜像
4. docker push 发布镜像 ==> DockerHub, 阿里云仓库

例子：

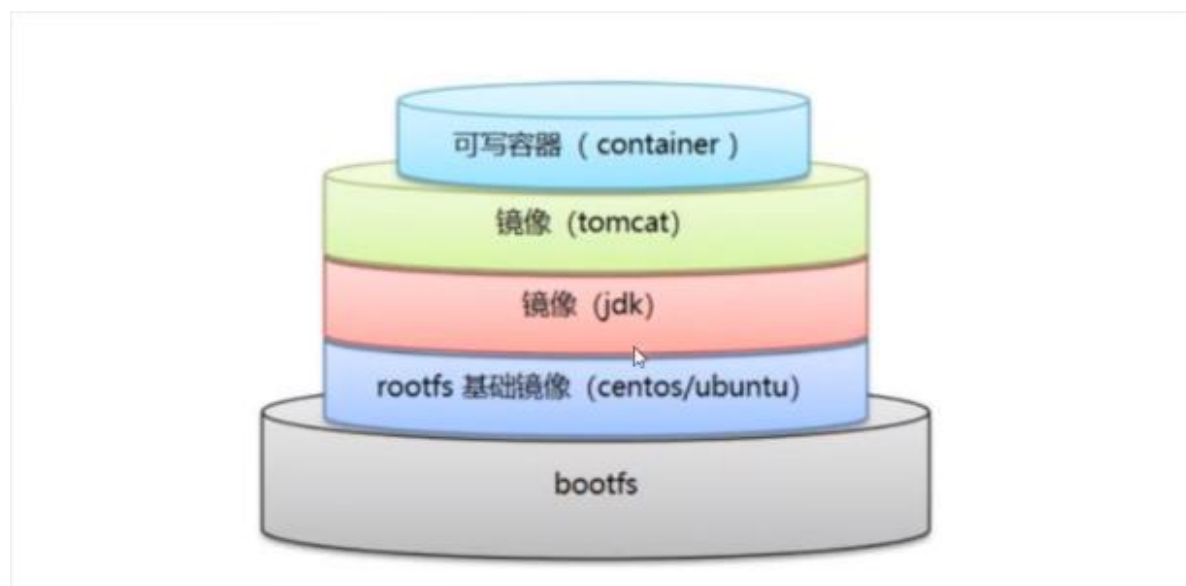
centos 7: <https://github.com/CentOS/sig-cloud-instance-images/blob/b521221b5c8ac3ac88698e77941a2414ce6e778d/docker/Dockerfile>

```
15 lines (13 sloc) | 516 Bytes
Raw Blame History
1 FROM scratch
2 ADD centos-7-x86_64-docker.tar.xz /
3
4 LABEL \
5     org.label-schema.schema-version="1.0" \
6     org.label-schema.name="CentOS Base Image" \
7     org.label-schema.vendor="CentOS" \
8     org.label-schema.license="GPLv2" \
9     org.label-schema.build-date="20200504" \
10    org.opencontainers.image.title="CentOS Base Image" \
11    org.opencontainers.image.vendor="CentOS" \
12    org.opencontainers.image.licenses="GPL-2.0-only" \
13    org.opencontainers.image.created="2020-05-04 00:00:00+01:00"
14
15 CMD ["/bin/bash"]
```

DockerFile构建过程

编写要点

1. 指令关键字都是大写字母
2. 执行从上到下
3. #为注释
4. 每一个指令都是一层镜像，且会提交

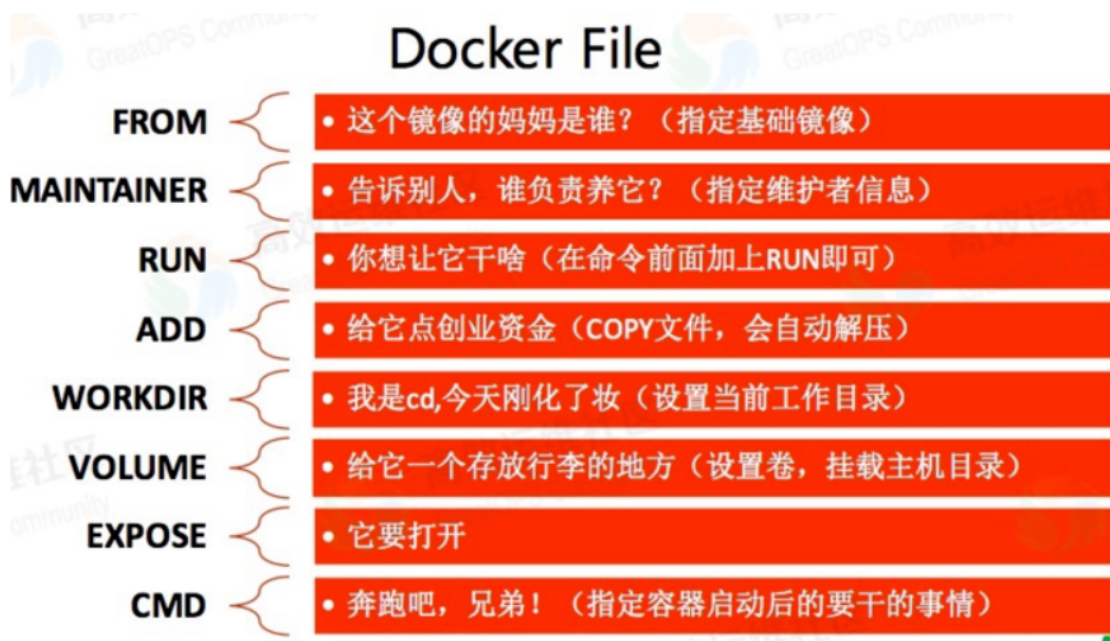


DockerImages就是通过DockFile构建生成的镜像，最终运行和发布的产品。

Docker容器是镜像运行起来的提供服务器

DockerFile的指令

指令	描述
FROM	构建新镜像是基于哪个镜像
MAINTAINER	镜像维护者姓名或邮箱地址
RUN	构建镜像时运行的Shell命令
COPY	拷贝文件或目录到镜像中
ENV	设置环境变量
USER	为RUN、CMD和ENTRYPOINT执行命令指定运行用户
EXPOSE	声明容器运行的服务端口
HEALTHCHECK	容器中服务健康检查
WORKDIR	为RUN、CMD、ENTRYPOINT、COPY和ADD设置工作目录
ENTRYPOINT	运行容器时执行，如果有多个ENTRYPOINT指令，最后一个生效
CMD	运行容器时执行，如果有多个CMD指令，最后一个生效



创建一个自己的centos镜像

```

1 FROM centos
2 MAINTAINER Himit_zh<372347736@qq.com>
3
4 ENV MYPATH /usr/local
5 WORKDIR $MYPATH
6
7 RUN yum -y install vim
8 RUN yum -y install net-tools
9
10 EXPOSE 80
11
12 CMD echo $MYPATH
13 CMD echo "-----end-----"
14 CMD /bin/bash
15

```



```
docker build -f dockerfile文件路径 -t 镜像名:[tag版本号] .
```

centos+tomcat+jdk1.8的镜像

```
FROM centos
MAINTAINER kuangshen<24736743@qq.com>

COPY readme.txt /usr/local/readme.txt

ADD jdk-8u11-linux-x64.tar.gz /usr/local/
ADD apache-tomcat-9.0.22.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-9.0.22
ENV CATALINA_BASH /usr/local/apache-tomcat-9.0.22
ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/lib:$CATALINA_HOME/bin

EXPOSE 8080

CMD /usr/local/apache-tomcat-9.0.22/bin/startup.sh && tail -F /usr/local/apache-tomcat-9.0.22/bin/logs/catalina.out
```

```
[root@kuangshen tomcat]# docker build -t diytomcat .

[root@kuangshen tomcat]# docker run -d -p 9090:8080 --name kuangshentomcat2 -v /home/kuangshen/build/tomcat/test:/usr/local/apac
he-tomcat-9.0.22/webapps/test -v /home/kuangshen/build/tomcat/tomcatlogs:/usr/local/apache-tomcat-9.0.22/logs diytomcat
```

发布镜像

```
docker login -u 账号 -p 密码
```

```
docker push hub账户名/tomcat02:1.0
docker push hub账户名/镜像名:tag
```

镜像名字一定要命名为 hub账户名/镜像名 不然会提交仓库失败!

Docker网络

```

Deleted: sha256:dd4f3b9197241b9de1e2d761300e339e37d03343b1e3de1b71002ad52101ec3
(Django) ubuntu@VM-0-11-ubuntu:~$ 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
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:ae:3d:b6 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.11/20 brd 172.16.15.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:feae:3db6/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:b8:41:51:ed brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:b8ff:fe41:51ed/64 scope link
        valid_lft forever preferred_lft forever
(Django) ubuntu@VM-0-11-ubuntu:~$

```

本机回环地址

腾讯云内网地址

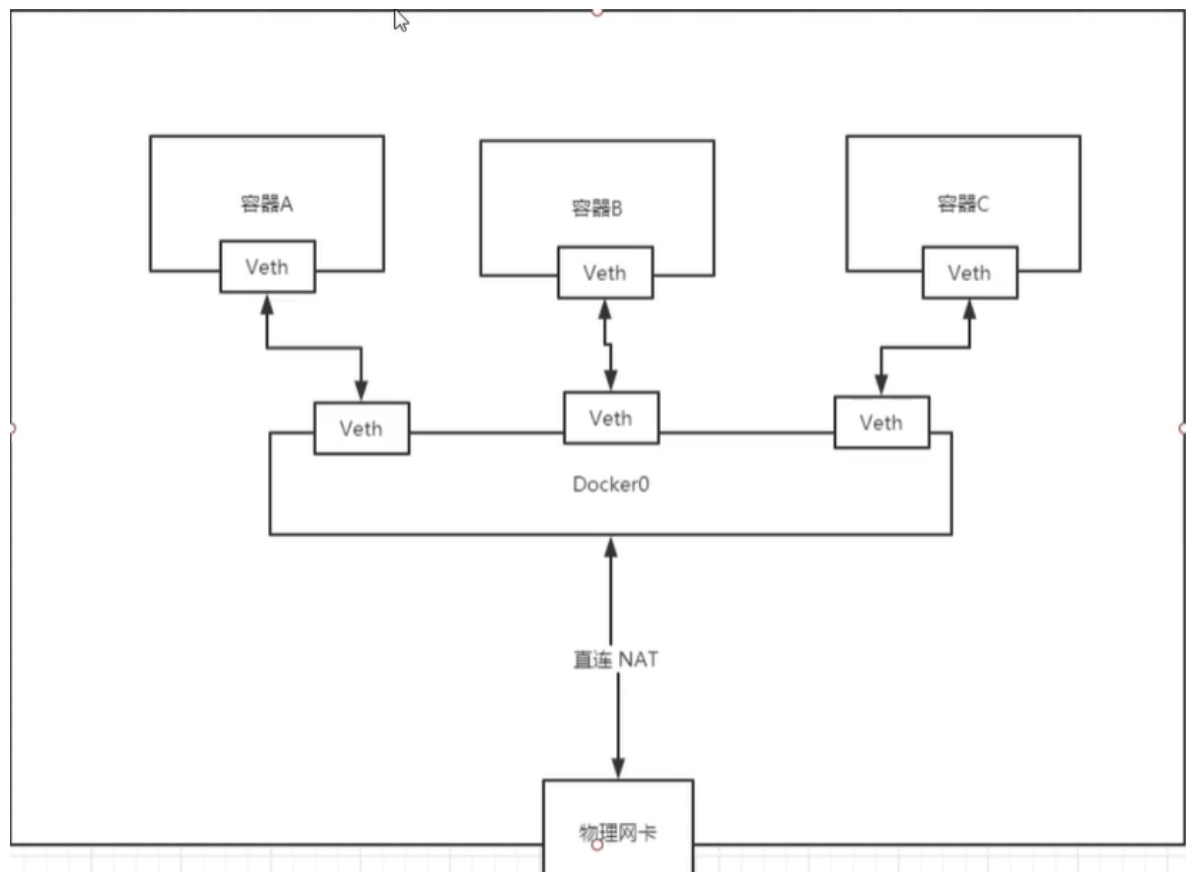
docker0地址

测试

Linux可以ping通docker容器内部网络

原理

1. 我们每启动一个docker容器, docker就会给docker容器分配一个ip ,我们只要安装了docker ,就会有一个网卡docker0, 桥接模式,使用的技术是veth-pair技术!
2. veth-pair 就是一对虚拟设备接口, 成对存在的,一段连着协议, 一段彼此相连。
3. 容器之间也是利用veth-pair技术, 也可以互相ping通的。



容器的转发都是经过linux中的docker0，docker0为网桥~ 可生成65535个容器网卡地址~

--link

```
docker run -d -P --name 新容器名 --link 已存在容器名 镜像名
```

#这样启动，即可用ping 容器名，无需要指定ip，但是只能新容器ping通旧容器，因为旧容器没有绑定新容器~

不支持使用--link

docker0不支持容器名连接

自定义网络

```
(Django) ubuntu@VM-0-11-ubuntu:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
32f4184eb74c        bridge             bridge              local
093d29ffd7c2        host               host                local
dc180a12d863        none               null                local
(Django) ubuntu@VM-0-11-ubuntu:~$
```

网络模式

bridge：桥接 docker (默认)

none：不配置网络

host：和宿主机共享网络

container：容器内网络连通！（用的少！）

```
segment
(Django) ubuntu@VM-0-11-ubuntu:~$ docker network create --driver bridge --subnet 192.168.0.0/16 --gateway 192.168.0.1 mynet
```

```
docker network create --driver bridge --subnet 192.168.0.0/16 --gateway
192.168.0.1 mynet
```

#创建自定义docker网络

--driver 定义模式

--subnet 子网掩码 192.168.0.1 ~192.168.255.255

--gateway 网关地址

mynet 名字

```

    "Containers": {
      "68e6142449d8bbb8a30aaae87bf314bc8322033b9801d6e7d5d5ddcd956088f6": {
        "Name": "tomcat-net-01",
        "EndpointID": "93b3bc5c755b43f433b3ca20e3f3a1a168c6f96abaacd8e655ce1b66b5e311a2",
        "MacAddress": "02:42:c0:a8:00:02",
        "IPv4Address": "192.168.0.2/16",
        "IPv6Address": ""
      },
      "ef21e6310b2f1d99b702f30c7d8df88a8b8a32982f0ca2f0b3ade3132fed4418": {
        "Name": "tomcat-net-02",
        "EndpointID": "d6935f18e9d618c9e002898edf840535cfd617dab3675f979c555f51996aef5a",
        "MacAddress": "02:42:c0:a8:00:03",
        "IPv4Address": "192.168.0.3/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
}

```

```

(Django) ubuntu@VM-0-11-ubuntu:~$ docker exec -it tomcat-net-01 ping tomcat-net-02
PING tomcat-net-02 (192.168.0.3) 56(84) bytes of data.
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=1 ttl=64 time=0.079 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=3 ttl=64 time=0.074 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=4 ttl=64 time=0.064 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=5 ttl=64 time=0.081 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=6 ttl=64 time=0.065 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=7 ttl=64 time=0.060 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=8 ttl=64 time=0.064 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=9 ttl=64 time=0.078 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=10 ttl=64 time=0.067 ms

```

发现自定义网络内的容器可以用容器名ping通

不用使用--link

网络连通

```

(Django) ubuntu@VM-0-11-ubuntu:~$ docker network connect --help
Usage: docker network connect [OPTIONS] NETWORK CONTAINER

Connect a container to a network

Options:
  --alias strings      Add network-scoped alias for the container
  --driver-opt strings driver options for the network
  --ip string          IPv4 address (e.g., 172.30.100.104)
  --ip6 string         IPv6 address (e.g., 2001:db8::33)
  --link list          Add link to another container
  --link-local-ip strings Add a link-local address for the container
(Django) ubuntu@VM-0-11-ubuntu:~$

```

测试 tomcat-01 =>>mynet网络

```

docker run -d -P --name tomcat-01 tomcat
docker network inspect mynet
--net 自定义网络名

```

查看mynet网络信息发现，原来是tomcat-01被加到mynet里面了

```

    "Containers": {
      "15a45199098b2c65039457c1736271526e1cee51ca3e34d0b34db55ea2078265": {
        "Name": "tomcat-01",
        "EndpointID": "4585b2a0dca61ac4fe7a806499ef5fd416ad1e42116c56beb71e
ea9dd9787fad",
        "MacAddress": "02:42:c0:a8:00:04",
        "IPv4Address": "192.168.0.4/16",
        "IPv6Address": ""
      },
    }
  }
}

```

验证

```
(Django) ubuntu@VM-0-11-ubuntu:~$ docker exec -it tomcat-01 ping tomcat-net-01
PING tomcat-net-01 (192.168.0.2) 56(84) bytes of data.
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=1 ttl=64 time=0.086 ms
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=2 ttl=64 time=0.083 ms
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=4 ttl=64 time=0.085 ms
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=5 ttl=64 time=0.065 ms
```

发现docker01网络中的tomcat01成功连通mynet网络中的tomcat-net-01，实现了容器连通网络。

部署Redis集群

```
docker network create redis --subnet 172.38.0.0/16
```

```
# xshell脚本创建
```

```
for port in $(seq 1 6); \
do \
mkdir -p /home/ubuntu/mydata/redis/node-${port}/conf
touch /home/ubuntu/mydata/redis/node-${port}/conf/redis.conf
cat << EOF >/home/ubuntu/mydata/redis/node-${port}/conf/redis.conf
port 6379
bind 0.0.0.0
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
cluster-announce-ip 172.38.0.1${port}
cluster-announce-port 6379
cluster-announce-bus-port 16379
appendonly yes
EOF
done
```

启动redis容器

```
docker run -p 6371:6379 -p 16371:16379 --name redis-1 \
-v /mydata/redis/node-1/data:/data \
-v /mydata/redis/node-1/conf/redis.conf:/etc/redis/redis.conf \
-d --net redis --ip 172.38.0.11 redis:5.0.9-alpine3.11 redis-server
/etc/redis/redis.conf
```

```
docker run -p 6372:6379 -p 16372:16379 --name redis-2 \
-v /mydata/redis/node-2/data:/data \
-v /mydata/redis/node-2/conf/redis.conf:/etc/redis/redis.conf \
```

```
docker run -p 6372:6379 -p 16372:16379 --name redis-2 \
-v /mydata/redis/node-2/data:/data \
-v /mydata/redis/node-2/conf/redis.conf:/etc/redis/redis.conf \
```

```

-d --net redis --ip 172.38.0.12 redis:5.0.9-alpine3.11 redis-server
/etc/redis/redis.conf

docker run -p 6373:6379 -p 16373:16379 --name redis-3 \
-v /mydata/redis/node-3/data:/data \
-v /mydata/redis/node-3/conf/redis.conf:/etc/redis/redis.conf \
-d --net redis --ip 172.38.0.13 redis:5.0.9-alpine3.11 redis-server
/etc/redis/redis.conf

for port in $(seq 1 6); \
do \
docker run -p 637${port}:6379 -p 1637${port}:16379 --name redis-${port} \
-v /mydata/redis/node-${port}/data:/data \
-v /mydata/redis/node-${port}/conf/redis.conf:/etc/redis/redis.conf \
-d --net redis --ip 172.38.0.1${port} redis:5.0.9-alpine3.11 redis-server
/etc/redis/redis.conf
done

```

配置集群

```

redis-cli --cluster create 172.38.0.11:6379 172.38.0.12:6379 172.38.0.12:6379
172.38.0.12:6379 172.38.0.
13:6379 172.38.0.14:6379 172.38.0.15:6379 --cluster-replicas 1

```