Docker 实践与练习

本文旨在通过实际操作让大家对docker有个初步的理解，而相关的原理和命令格式说明并不在本文档内，在阅读本文档过程中，你可以同时查阅相关的命令和概念说明

如 Docker 教程 | 菜鸟教程 (runoob.com) 及Docker 命令大全 Docker run命令详解

更全更完整的命令等说明也可到官网详查 Docker Documentation

本次练习请准备好以下镜像

```
sudo docker pull alpine
sudo docker pull nginx
sudo docker golang
sudo docker
```

安装

对于初步练习和简单使用可以采用以下简单方式进行安装，在ubuntu可使用如下命令

```
$ sudo apt update
$ sudo apt install docker.io
$ sudo docker --version
Docker version 20.10.2, build 20.10.2-0ubuntu1~18.04.2
```

如果进行在生产环境配置与部署请采用官方的推动简方式，以下给出了英文和中文的连接以供产考。

系统	官方指导	中文参考
Windows	Install Docker Desktop on Windows	Windows Docker 安装
Linux	Install Docker Engine on Ubuntu	Ubuntu Docker 安装
Mac	Install Docker Desktop on Mac	MacOS Docker 安装

Docker 镜像加速

由于国内的hub.docker.com访问比较慢，所以通常需要使用代理来加速镜像的下载速度。可以通过修改/etc/docker/daemon.json文件来实现

如果daemon.json文件不存在请创建，下面选项配置就是采用了中科大的代理

```
{
    "registry-mirrors": ["https://docker.mirrors.ustc.edu.cn"]
}
```

修改完毕后需要重启docker服务来生效该配置,如下命令示例了重启与查看状态

```
$ sudo systemctl restart docker && sudo systemctl status docker
```

```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
enabled)
   Active: active (running) since Fri 2021-05-07 10:01:23 UTC; 9ms ago
     Docs: https://docs.docker.com
 Main PID: 1995 (dockerd)
    Tasks: 10
   CGroup: /system.slice/docker.service
           └─1995 /usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock

May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.599243260Z"
level=info msg="[graphdriver] using prior storage driver: overlay2"
May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.608963447Z"
level=warning msg="Your kernel does not support swap memory limit"
May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.608994398Z"
level=warning msg="Your kernel does not support CPU realtime scheduler"
May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.609124041Z"
level=info msg="Loading containers: start."
May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.685308389Z"
level=info msg="Default bridge (docker0) is assigned with an IP address
172.17.0.0/16. Daemon option --bip can be used to set a preferred IP address"
May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.837338326Z"
level=info msg="Loading containers: done."
May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.885076773Z"
level=info msg="Docker daemon" commit="20.10.2-0ubuntu1~18.04.2"
graphdriver(s)=overlay2 version=20.10.2
May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.885124986Z"
level=info msg="Daemon has completed initialization"
May 07 10:01:23 edgetoolkit systemd[1]: Started Docker Application Container
Engine.
May 07 10:01:23 edgetoolkit dockerd[1995]: time="2021-05-07T10:01:23.900779029Z"
level=info msg="API listen on /var/run/docker.sock"
```

Docker 镜像与容器

关于docker镜像与容器的及相关原理概念这里不再介绍，基础概念请到 Docker 镜像使用 Docker 容器使用 学习.

对于一个刚刚配置完的系统是没有任何镜像的如:

```
$ sudo docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

那么接下来然我们下载一个最小的Linux apline

```
$ sudo docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
540db60ca938: Pull complete
Digest: sha256:69e70a79f2d41ab5d637de98c1e0b055206ba40a8145e7bddb55ccc04e13cf8f
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

此时在查看镜像,可以看到 Alpine Linux 其大小为5.61

```
$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
alpine       latest    6dbb9cc54074   3 weeks ago   5.61MB
```

接下来让我们采用交互模式进入Alpine Linux并查看其release信息

```
~$ sudo docker run -i -t alpine /bin/sh
/ # cat /etc/issue
Welcome to Alpine Linux 3.13
Kernel \r on an \m (\l)
/ # exit
```

我们也可以采用直接显示，而无需进入交互模式

```
$ sudo docker run alpine /bin/sh -c "cat /etc/issue"
Welcome to Alpine Linux 3.13
Kernel \r on an \m (\l)
```

参数说明：

- **-i**: 交互式操作。
- **-t**: 终端。
- alpine: alpine镜像。
- /bin/sh：放在镜像名后的是命令

第二种方式的shell方式 -c "cat /etc/issue"就是shell命令的用法，与通常大家在linux操作命令相同。

以上我们运行了两个docker命令，此时如果查看容器信息会显示

```
$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
            PORTS     NAMES
7c01cdd2a8ed   alpine    "/bin/sh -c 'cat /et…"   5 minutes ago   Exited (0) 5
minutes ago             cranky_elbakyan
b7918e7aa0b3   alpine    "/bin/sh"                8 minutes ago   Exited (0) 6
minutes ago             kind_carver
```

通过该显示信息，你应该能够猜到每个容器对应的命令了，而这两个容器都已经退出，如果你不再使用，可以通过 docker rm 或docker container prune进行清除如

```
$ sudo docker rm 7c01cdd2a8ed
7c01cdd2a8ed
$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS
PORTS     NAMES
b7918e7aa0b3   alpine    "/bin/sh"   11 minutes ago   Exited (0) 9 minutes ago
          kind_carver
```

```
$ sudo docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
b7918e7aa0b32153debf5fd2f2e846b94f8a4435ac8874ea76e9c3e779a921c0

Total reclaimed space: 29B
$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
edgetoolkit@edgetoolkit:~$
```

这三个命令的细节请参阅 Docker 命令大全 中 **ps run rm images**

Docker容器执行与查看

本操作 练习一下命令 **ps** , **inspect** , **attach**, **exec**, **start/stop/restart**, **run** , **logs**

我们通过对Nginx来练习

交互模式

首先，启动nginx,由于我们采用的交互模式并给容器指定名为`web`，所以整个终端被在用，为了查看、操作容器我们需要启动另外一个终端(下文称作T1)

```
$ sudo docker run -it --name web -p 80:80 nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to
perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-
default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of
/etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in
/etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-
templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-
processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

打开终端T1并进行一下操作， 可以看到nginx已经启动，容器名称为 nginx,容器端对外暴露80口号被采用。

```
$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS
         PORTS                NAMES
30f5b5e14ac9   nginx     "/docker-entrypoint.…"   About a minute ago   Up About
a minute   0.0.0.0:80->80/tcp   web
```

通过curl命令访问web页面

```
$ curl http://127.0.0.1:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

如果你此时回到启动Nginx的终端就会看到刚才的curl 命令触发了Nginx打印了一条日志

```
172.17.0.1 - - [08/May/2021:01:50:30 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.58.0" "-"
```

后台模式

同样的功能我们采用docker的后台方式运行,web端口号为8080

```
$ sudo docker run -d --name web-8080 -p 8080:80 nginx
edcb4a2f2c9593e78efd51065540b99e8cabe2128a2b43f5459b3a1fa8668534
$ sudo docker ps
```

```
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS
         PORTS                  NAMES
d3658d6e2404   nginx     "/docker-entrypoint.…"   6 seconds ago        Up 5
seconds        0.0.0.0:80->80/tcp     web
edcb4a2f2c95   nginx     "/docker-entrypoint.…"   About a minute ago   Up About
a minute   0.0.0.0:8080->80/tcp   web-8080
$ curl http://127.0.0.1:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
$ sudo docker logs web-8080
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to
perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-
default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of
/etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in
/etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-
templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-
processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
172.17.0.1 - - [08/May/2021:01:59:59 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.58.0" "-"
```

可以看到我们通过通过docker logs 可以查看到相同效果

接下来让我们进入到容器查看

```
$ sudo docker exec -it  web-8080 /bin/sh
```

```
# curl http://127.0.0.1:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

# nginx -h
nginx version: nginx/1.19.10
Usage: nginx [-?hvVtTq] [-s signal] [-p prefix]
             [-e filename] [-c filename] [-g directives]

Options:
  -?,-h         : this help
  -v            : show version and exit
  -V            : show version and configure options then exit
  -t            : test configuration and exit
  -T            : test configuration, dump it and exit
  -q            : suppress non-error messages during configuration testing
  -s signal     : send signal to a master process: stop, quit, reopen, reload
  -p prefix     : set prefix path (default: /etc/nginx/)
  -e filename   : set error log file (default: /var/log/nginx/error.log)
  -c filename   : set configuration file (default: /etc/nginx/nginx.conf)
  -g directives : set global directives out of configuration file

# cat /etc/nginx/nginx.conf

user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}
```

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
# cat /etc/nginx/conf.d/default.conf
server {
    listen       80;
    listen  [::]:80;
    server_name  localhost;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}
```

```
    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}
}
# mkdir -p /usr/share/nginx/html  && echo "Hello
World">/usr/share/nginx/html/index.html
# curl http://127.0.0.1:80
Hello World
# exit
```

- 进入容器 `sudo docker exec -it web-8080 /bin/sh`
- 在容器内部 访问nginx 页面 `curl http://127.0.0.1:80`
- 查看配置 `cat /etc/nginx/nginx.conf`
- 查看默认配置 `cat /etc/nginx/conf.d/default.conf`

  此处可以看到默认页面在 /usr/share/nginx/html (但该目录不存在，实际显示的页面是Nginx自己申城的)
- 创建Hello页面 `mkdir -p /usr/share/nginx/html && echo "Hello World">/usr/share/nginx/html/index.html`
- 再次访问web `curl http://127.0.0.1:80`

  此时的页面变成了Hello World

在容器主机外访问该页面也已经变化

```
$ curl http://127.0.0.1:8080
Hello World
```

停止容器:

```
$ sudo docker stop web-8080
web-8080
$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
            PORTS                  NAMES
d3658d6e2404   nginx     "/docker-entrypoint.…"   53 minutes ago   Up 53 minutes
             0.0.0.0:80->80/tcp   web
edcb4a2f2c95   nginx     "/docker-entrypoint.…"   55 minutes ago   Exited (0) 4
seconds ago                         web-8080
$ curl http://127.0.0.1:8080
curl: (7) Failed to connect to 127.0.0.1 port 8080: Connection refused
```

此时在访问页面就失败了

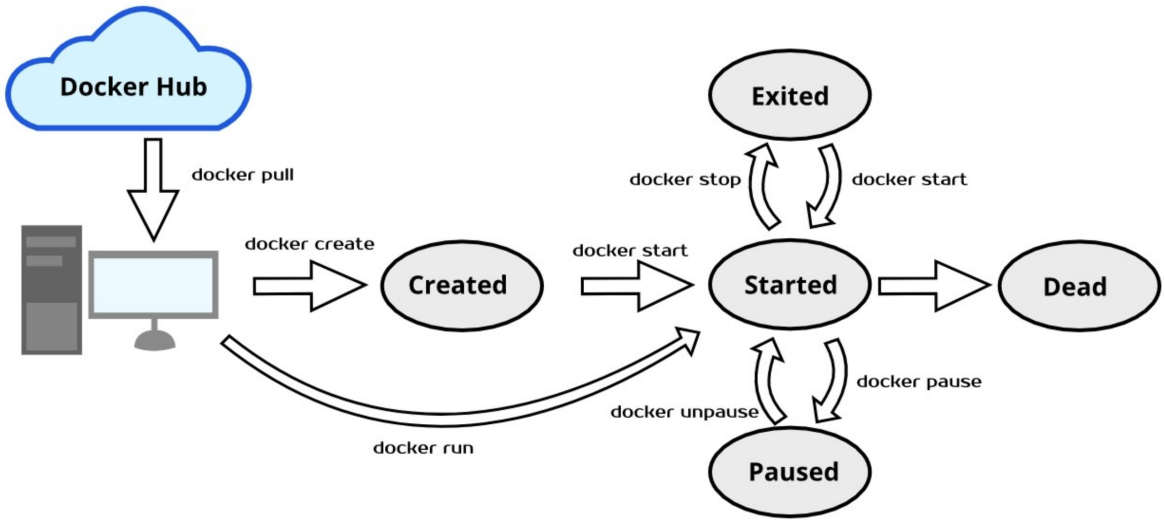
删除容器

```
$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
                 PORTS                NAMES
d3658d6e2404   nginx     "/docker-entrypoint.…"   55 minutes ago   Up 55 minutes
                 0.0.0.0:80->80/tcp   web
edcb4a2f2c95   nginx     "/docker-entrypoint.…"   57 minutes ago   Exited (0)
About a minute ago                          web-8080
$ sudo docker rm web-8080
web-8080
$ sudo docker ps -s
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
PORTS                NAMES     SIZE
d3658d6e2404   nginx     "/docker-entrypoint.…"   56 minutes ago   Up 56 minutes
  0.0.0.0:80->80/tcp   web       1.12kB (virtual 133MB)
```

docker start /restart命令请用户自行练习

Docker容器的状态

docker容器有created（已创建），running（运行中），paused（暂停），exited（停止），dead（死亡）五个状态，器的一个变化关系与命令的关系图如下



根据docker ps 的filter选项 可以看到实际上容器共有7个状态 `created`，`restarting`，`running`，`removing`，`paused`，`exited`，or `dead`

`dead` 状态查到的一个解释如下

```
"dead" is used for a "defunct" container; for example, a container that you
wanted to remove but was only partially removed because resources were kept busy
by an external process. Dead containers cannot be (re)started, only removed. You
can manually attempt to remove a dead container (if the problem causing it to not
be removed in the first attempt failed), and the daemon will automatically
attempt to remove dead containers when it's restarted.
```

Docker的文件系统

下载golang编译环境镜像 `sudo docker pull golang`

进入docker直接编译

```
$ sudo docker run -it --rm  golang /bin/bash
root@fd760394d77a:/go# cat > main.go << EOF
> package main
> import "fmt"
> func main()  {
>     fmt.Println("Hello World")
> }
> EOF
root@fd760394d77a:/go# go build main.go
root@fd760394d77a:/go# ls -l
total 1908
drwxrwxrwx 2 root root    4096 May  6 23:17 bin
-rwxr-xr-x 1 root root 1937623 May  8 03:13 main
-rw-r--r-- 1 root root      74 May  8 03:11 main.go
drwxrwxrwx 2 root root    4096 May  6 23:17 src
root@fd760394d77a:/go# ./main
Hello World
root@fd760394d77a:/go#
```

下面我们采用将主机上的目录映射到docker容器内进行操作

在本机建立目录hello并编辑好main.go文件

```
$ ls hello -l
total 4
-rw-rw-r-- 1 edgetoolkit edgetoolkit 75 May  8 03:17 main.go
```

我们采用以下命令将hello 目录并通过命令方式字节编译

```
sudo docker run --rm -v $PWD/hello:/builds -w /builds golang /bin/bash -c "go
build main.go"
```

-v --volume 将本地目录绑定到了容器的/builds目录下，同过 -w将当前工作目录设置到/builds

此处我们采用了--rm flag也就是容器运行完毕后会自行删除，编译成功后在hello目录中生成了可执行程序，即可执行了

```
$ ls -l hello
total 1900
-rwxr-xr-x 1 root        root        1937623 May  8 03:24 main
-rw-rw-r-- 1 edgetoolkit edgetoolkit      75 May  8 03:17 main.go
$ ./hello/main
Hello World
```

关于docker文件系统，更复杂的应用就会使用到volume, 你必须知道的Docker数据卷(Volume) - EdisonZhou - 博客园 (cnblogs.com)

有更为详尽的说明和联系，在入门阶段可以简单了解即可，在实际工作中有用到时可以深入学习

Docker网络

Docker网络的原理相对比较复杂，本文仅通过简单的案例带大家入个门，关于进本原理可以通过Docker网络详解——原理篇*meltsnow的博客-CSDN博客*docker 网络了解，进一步的建议阅读官方文档或更专业的书籍

准备工作

关于mosquitto/MQTT 请自行百度了解吧 (MQTT消息队列服务介绍和Mosquitto的安装和配置)

```
$ sudo docker pull eclipse-mosquitto
$ mkdir config
$ cat >config/mosquitto.conf<< EOF
> baallow_anonymous true
> listener 1883 0.0.0.0
> EOF
$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:92ff:fec2:14c1  prefixlen 64  scopeid 0x20<link>
        ether 02:42:92:c2:14:c1  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 806 (806.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe3f:e7c9  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:3f:e7:c9  txqueuelen 1000  (Ethernet)
        RX packets 125725  bytes 176369104 (176.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 49104  bytes 3377877 (3.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 274  bytes 22776 (22.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 274  bytes 22776 (22.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vethc807f06: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::f4e5:44ff:feb0:58d9  prefixlen 64  scopeid 0x20<link>
        ether f6:e5:44:b0:58:d9  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

```
        TX packets 12  bytes 976 (976.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

这里可以查看到docker网络地址 172.17.01 和主机物理网络 enp0s3 10.0.2.15

启动中介服务(Broker)

```
$ sudo docker run -it --name mosquitto --rm -p 1883:1883  -v
$PWD/config:/mosquitto/config eclipse-mosquitto
1620614148: mosquitto version 2.0.10 starting
1620614148: Config loaded from /mosquitto/config/mosquitto.conf.
1620614148: Opening ipv4 listen socket on port 1883.
1620614148: mosquitto version 2.0.10 running
```

通过命令启动`订阅者(sub)`侦听消息

```
sudo docker run --rm eclipse-mosquitto /bin/sh -c "mosquitto_sub -h 10.0.2.15 -p
1883 -t topic"
```

此时观察Broker终端可以看到信息

```
1620614200: New connection from 172.17.0.1:35860 on port 1883.

1620614200: New client connected from 172.17.0.1:35860 as auto-BFA3332F-A978-
E519-95DB-C3B8111E2EA2 (p2, c1, k60).
```

`发布者(Pub)`

```
$ sudo docker run --rm eclipse-mosquitto /bin/sh -c "mosquitto_pub -h 172.17.0.1
-p 1883 -t topic -m Hello"
```

订阅者窗口收到消息

```
$ sudo docker run --rm eclipse-mosquitto /bin/sh -c "mosquitto_sub -h 10.0.2.15
-p 1883 -t topic"
Hello
```

以上的操作中我们使用的相当于是桥接模式，所以三方的通信实际上是通过主机的网络来完成的， 如果你去掉Broker中的`-p 1883: 1883`则无法工作。

如果将网络设置为Host模式即用一下命令启动Broker

```
sudo docker run -it --name mosquitto --rm --net=host  -v
$PWD/config:/mosquitto/config eclipse-mosquitto
```

可以达到同样的效果。

Docker容器的资源限制

默认情况下，容器没有资源限制，可以使用主机内核调度程序允许的尽可能多的给定资源。Docker提供了控制容器可以使用多少内存或CPU的方法，设置docker run命令的运行时配置标志。通过docker run --help 可以看到如下部分

```
$ docker run --help
      --cpu-period int                 Limit CPU CFS (Completely Fair Scheduler)
period
      --cpu-quota int                  Limit CPU CFS (Completely Fair Scheduler)
quota
      --cpu-rt-period int              Limit CPU real-time period in
microseconds
      --cpu-rt-runtime int             Limit CPU real-time runtime in
microseconds
  -c, --cpu-shares int                 CPU shares (relative weight)
      --cpus decimal                   Number of CPUs
      --cpuset-cpus string             CPUs in which to allow execution (0-3,
0,1)
      --cpuset-mems string             MEMs in which to allow execution (0-3,
0,1)

  -m, --memory bytes                   Memory limit

      --memory-reservation bytes       Memory soft limit
      --memory-swap bytes              Swap limit equal to memory plus swap:
'-1' to enable unlimited swap
      --memory-swappiness int          Tune container memory swappiness (0 to
100) (default -1)
      --oom-kill-disable               Disable OOM Killer
      --oom-score-adj int              Tune host's OOM preferences (-1000 to
1000)

  ...
```

Docker(二十)-Docker容器CPU、memory资源限制 是一篇非常好的中文文章，大家可以参阅细读一下，这里就不做拷贝工作了。

Linux Top 命令

不进行CPU限制是运行一下命令，然后通过top 和docker stats 命令对比查看

```
sudo docker run --name stress --rm   polinux/stress stress -c 1  -t 20
```

```
sudo docker run --name stress --rm   polinux/stress stress -c 2  -t 20
```

```
sudo docker run --name stress --rm  --cpus 1 polinux/stress stress -c 2  -t 20
```

```
sudo docker run --name stress --rm  --cpus 2 polinux/stress stress -c 4  -t 20
```

内存限制实验

```
 sudo docker run --name stress --rm  polinux/stress stress --vm 2 --vm-bytes 200M
--vm-hang 20 -t 20
```

```
 sudo docker run --name stress --rm  -m 100M  polinux/stress stress --vm 2 --vm-
bytes 20M --vm-hang 20 -t 20
```

```
 sudo docker run --name stress --rm  -m 100M  polinux/stress stress --vm 2 --vm-
bytes 200M --vm-hang 20 -t 20
```

制作Dockerfile

go web 源文件

```
package main

import (
    "fmt"
    "net/http"
)

func Hello(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello Web!\n-- from URL %s\n", r.URL.Path[1:])
}

func main() {
  fmt.Println("Running...\n")
    http.HandleFunc("/docker/", Hello)
    http.ListenAndServe(":8080", nil)
  fmt.Println("Exited!\n")
}
```

编译产生应用程序

```
$ sudo docker run --name golang --rm -v $PWD:/builds -w /builds golang go build
-o hello-web main.go
```

制作镜像 (Dockerfile)

```
#FROM ubuntu:focal
FROM alpine:latest

COPY hello-web /usr/sbin/hello-web

RUN mkdir /lib64 \
    && ln -s /lib/libc.musl-x86_64.so.1 /lib64/ld-linux-x86-64.so.2
EXPOSE 8080

CMD ["hello-web"]
```

```
$ sudo docker build . -t hello-web
$ sudo docker images
REPOSITORY       TAG       IMAGE ID       CREATED          SIZE
hello-web        latest    c9031fb1e07b   5 seconds ago    11.8MB

#启动docker服务
$ sudo docker run -d --rm --name hello -p 80:8080 hello-web
$ curl http://127.0.0.1/docker/hi
Hello Web!
-- from URL docker/hi
```

附

环境安装与准备

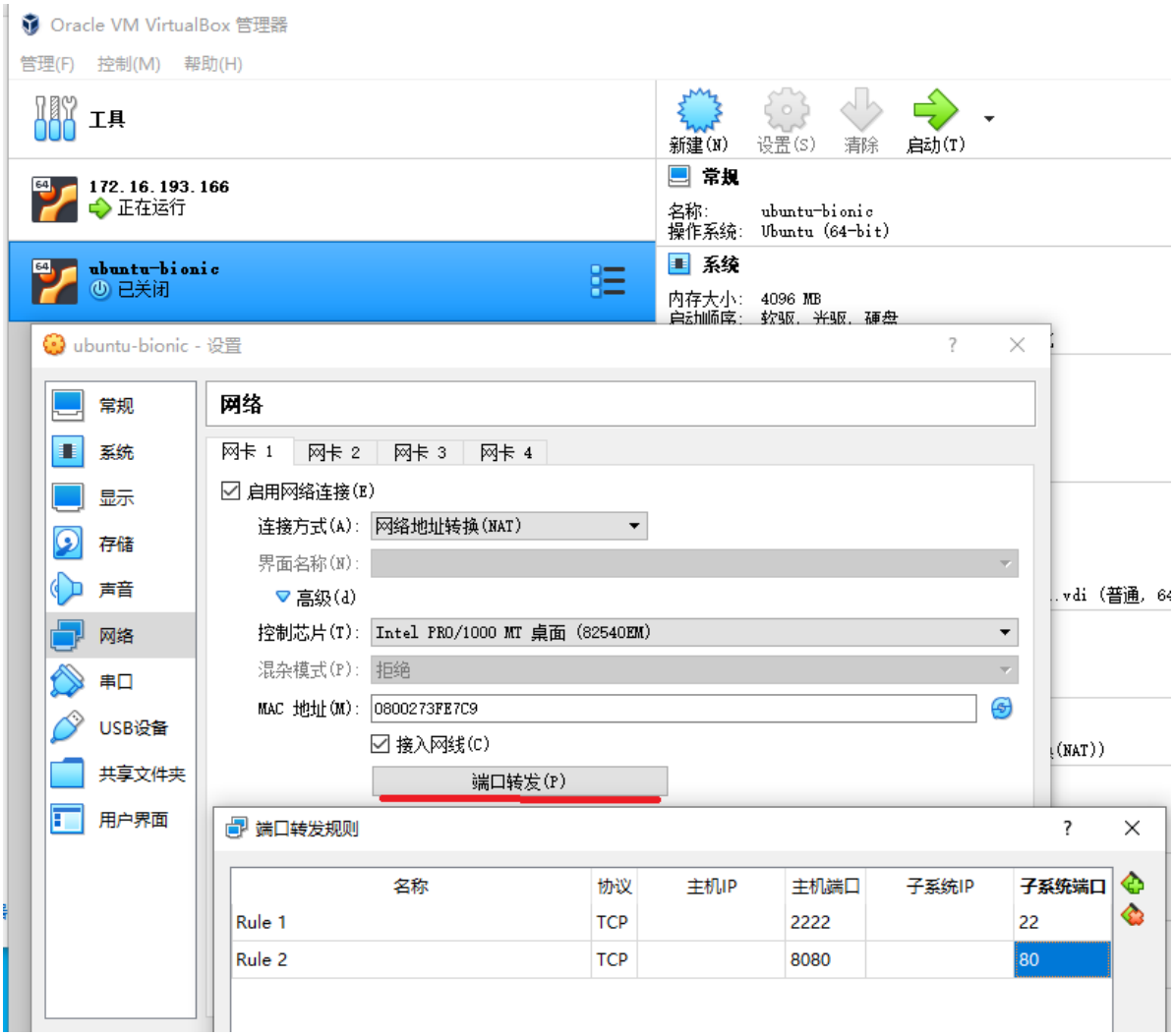
安装virtualbox 下载地址 Downloads – Oracle VM VirtualBox

我已经做好要的虚拟机镜像，可以下载到本地再导入 镜像地址 http://172.16.0.119/mirror/archive/ubuntu-bionic.ova

【管理】=》【导入虚拟电脑】加载ubuntu-boionic.ova

通过【设置】=》【系统】将内存设置为 4096M CPU数为 2

通过【设置】=》配置映射端口号 2222(host) => 22 (guset) 8080(host)=>80(guest)



配置成功后， 启动虚拟机后， 就可以通过 ssh :2222进行操作了。用户名: edgetoolkit 密码: edgetoolkit

```
$ sudo rm /var/lib/dpkg/lock
$ sudo apt update
$ sudo apt install tmux vim docker.io
```

添加进行加速，并下载好备用images

执行apt安装过程中如出现 以下提示 使用`sudo rm /var/lib/dpkg/lock` 命令清除在继续

```
edgetoolkit@edgetoolkit:~$ sudo apt update
E: Could not get lock /var/lib/dpkg/lock-frontend - open (11: Resource
temporarily unavailable)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is
another process using it?
edgetoolkit@edgetoolkit:~$ sudo rm /var/lib/dpkg/lock
```

docker 的实现原理参考文档

Docker 核心技术与实现原理 - DockOne.io

Docker底层原理介绍 - 简书 (jianshu.com)