**01-容器的介绍**

**1. 虚拟化和容器技术**

* 虚拟化技术

Type1: 不安装宿主机操作系统，而直接在硬件基础上安装虚拟化管理软件

Type2: 安装宿主机操作系统，在宿主机操作系统之上安装虚拟化管理软件

* 容器技术

Container 是在宿主机操作系统上使用Cgroups,Namespaceses技术创建出来具有边界的特殊进程。

* 区别

虚拟化技术会虚拟出多个内核，每个虚拟机拥有一个自己的内核，彼此之间隔离性非常好。并且在创建虚拟机之初就定义好了虚拟机的资源限制，如CPU核心数，内存大小，磁盘大小等。

容器技术是直接运行在宿主机之上的，多个容器共用一个内核，因此隔离效果比较差，但是性能更好。

图表, 表格

描述已自动生成

**2. 容器化的基础**

**2.1. Chroot**

如果需要在一个宿主机上运行多个容器，且容器之间相互个离，那么第一个就需要系统库文件的依赖，对于一个容器而言，需要将其需要的系统文件单独复制出来一份，放到指定目录，并且需要让进程认为这就是根目录，而不是去调用宿主机系统上的库文件。Chroot就是一个切换根目录的方式。

**2.2. NameSpaces**

为了让多个容器以沙盒的方式在宿主机上运行，就需要提前定义好各个容器能看到的边界。由于各个容器都是直接运行在宿主机系统上，因此需要内核对各个容器的上下文进行修改，让他们看上去是一个独立的操作系统。比如，指定PID为1的进程，指定网卡设备，指定文件系统挂载，指定用户等等。

Linux操作系统内核从底层实现了为各个进程创建独立用户空间的功能，不同用户空间似于一个个独立的虚拟机系统，用户空间内部进程不能感知到其它用户空间中的进程状态。内核提供了六种Namespaces:

|  |  |  |  |
| --- | --- | --- | --- |
| UTS | hostname and domainname | 主机名和域名隔离 | 内核版本：2.6.19 |
| User |  | 用户隔离。运行进程的用户和组 | 内核版本：3.8.x |
| Mount |  | 挂载点隔离。即挂载点隔离，主要指根目录 | 内核版本：2.4.19 |
| IPC | Inter-process-connection | 进程间通信隔离。消息队列、共享内容、信号量 | 内核版本：2.6.19 |
| Pid | Process ID | PID隔离 | 内核版本：2.6.24 |
| Net | Network | 网络隔离。网络设备、协议栈、端口 | 内核版本：2.6.29 |

**2.3. Cgroups**

Namespaceses通过障眼法实现了用户空间的隔离，但是没办法对硬件资源进行限制，当一个容器进行CPU密集型操作时，会消耗掉整个宿主机的CPU资源，进而影响了其它容器的正常运行。

因此在Namespaceses之上，还需要对各个容器实现硬件资源限制，比如CPU，Memory,diskio等等。

Cgroups技术针对进程而言的，在centos7系统上，可以通过以下方式来实现对进程的资源限制:

[root@centos-82 ~]# while :;do :;done &

[1] 2136

[root@centos-82 ~]# pidstat -u -p 2136 2 ## 未加cgroups限制下，跑满单个CPU核心

11:09:54 AM UID PID %usr %system %guest %CPU CPU Command

11:09:56 AM 0 2136 99.50 0.00 0.00 99.50 4 bash

11:09:58 AM 0 2136 100.00 0.00 0.00 100.00 4 bash

11:10:00 AM 0 2136 100.00 0.00 0.00 100.00 4 bash

[root@centos-82 ~]# mount -t cgroup ## 查看当前cgroups路径

cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release\_agent=/usr/lib/systemd/systemd-cgroups-agent,name=systemd)

cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)

cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)

cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpuacct,cpu)

cgroup on /sys/fs/cgroup/net\_cls,net\_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net\_prio,net\_cls)

cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)

cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)

cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)

cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)

cgroup on /sys/fs/cgroup/perf\_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf\_event)

cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)

[root@centos-82 ~]# mkdir /sys/fs/cgroup/cpu/loop

[root@centos-82 ~]# cat /sys/fs/cgroup/cpu/loop/cpu.cfs\_quota\_us

-1

[root@centos-82 ~]# cat /sys/fs/cgroup/cpu/loop/cpu.cfs\_period\_us

100000

[root@centos-82 ~]# echo 10000 >/sys/fs/cgroup/cpu/loop/cpu.cfs\_quota\_us ## CPU时间片限制在20%

[root@centos-82 ~]# echo 2136 > /sys/fs/cgroup/cpu/loop/tasks ## 指定限制的进程PID

[root@centos-82 ~]# pidstat -u -p 2136 2

Linux 3.10.0-862.el7.x86\_64 (centos-82) 03/02/2019 \_x86\_64\_ (8 CPU)

11:16:37 AM UID PID %usr %system %guest %CPU CPU Command

11:16:39 AM 0 2136 10.50 0.00 0.00 10.50 4 bash

11:16:41 AM 0 2136 10.00 0.00 0.00 10.00 4 bash

11:16:43 AM 0 2136 9.50 0.00 0.00 9.50 4 bash

11:16:45 AM 0 2136 9.95 0.00 0.00 9.95 4 bash

**3. Docker架构**

**3.1. Docker组件**

图示

描述已自动生成

Docker服务有三个部分组成，分别是Client，Docker Host，Registry。当创建新的容器时，会向Docker Daemon发送指令，Docker Daemon通过本地镜像文件创建容器，当本地不存在镜像时，将从Registry下载镜像。

Registry由两个部分组成：

* Repostitory
* 由特定的docker镜像的所有迭代版本组成一个镜像仓库
* 一个Registry可以包括多个Repostitory
* Repostitory包含顶层仓库和用户仓库
* 顶层仓库: 仓库名:标签, nginx:latest
* 用户仓库: 用户名/仓库名:标签, heyang/nginx:1.4.2
* 一个镜像可以有多个标签，如最新版的nginx,可以是nginx:latest,nginx:1.4.2
* Index
* 提供用户认证、镜像检索功能

**3.2. Docker镜像和容器**

**3.2.1. Docker镜像**

镜像(Image)是一堆只读层(read-only layer)的统一视角。如下图所示:

图形用户界面, 应用程序

描述已自动生成

图形用户界面

描述已自动生成

左边的是多个只读层，他们相互堆叠在一起。除了最下层之外，其它每一层都会有一个指针指向下一层。这些层是Docker内部的实现细节，并且能够在宿主机的文件系统上访问到。

统一文件系统（union file system,aufs）技术(新版用overlay2)能够将不同的层整合成一个文件系统，为这些层提供了一个统一的视角，这样就隐藏了多层的存在，在用户的角度看来，只存在一个文件系统。我们可以在图片的右边看到这个视角的形式。

每一层都包含了当前层的ID，Metadata，Pointer（指向上一层）三层，最底层不包含Pointer。

**3.2.2. Docker容器**

Docker容器包含静止状态和运行状态两种，这两种状态下的层级不一样。

静态状态的容器仅仅是在镜像状态下增加一个可读写的层级，运行状态中的容器包含了进程和对应的进程空间：

图形用户界面, 应用程序

描述已自动生成

图片包含 文本

描述已自动生成

**4. Docker安装**

**4.1. Docker安装配置**

Docker在2017年以前时使用大版本号+小版本号来名，在2017年之后，采用YY.MM.N-xx格式，如 19.03.1-ce表示2019年3月份的第2个ce版本。以CentOS 7安装docker-ce版本为例

[root@docker-24-20 ~]# uname -r # 确认内核版本，要求大于3.8

3.10.0-862.el7.x86\_64

[root@docker-24-20 ~]# wget -O /etc/yum.repos.d/docker-ce.repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

[root@docker-24-20 ~]# yum install -y docker-ce # 安装docker-ce

[root@docker-24-20 ~]# vim /etc/docker/daemon.json # 初始化配置

{

"graph": "/data/docker",

"storage-driver": "overlay2",

"insecure-registries": ["registry.access.redhat.com","quay.io"],

"registry-mirrors": ["https://q2gr04ke.mirror.aliyuncs.com"],

"bip": "172.24.20.1/24",

"exec-opts": ["native.cgroupdriver=systemd"],

"log-opts": {"max-size":"32M", "max-file":"2"},

"live-restore": true

}

[root@docker-24-20 ~]# mkdir -p /data/docker

[root@docker-24-20 ~]# systemctl start docker && systemctl enable docker

[root@docker-24-20 ~]# ip addr show dev docker0 # 确认IP地址

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default

link/ether 02:42:e7:6b:2d:f6 brd ff:ff:ff:ff:ff:ff

inet 172.24.20.1/24 brd 172.24.20.255 scope global docker0

valid\_lft forever preferred\_lft forever

[root@docker-24-20 ~]# docker container run --rm hello-world # 测试docker是否运行正常

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

1b930d010525: Pull complete

Digest: sha256:4fe721ccc2e8dc7362278a29dc660d833570ec2682f4e4194f4ee23e415e1064

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

(amd64)

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/

# daemon.json 配置介绍

{

"graph": "/data/docker",

"storage-driver": "overlay2",

"insecure-registries": ["registry.access.redhat.com","quay.io"],

"registry-mirrors": ["https://q2gr04ke.mirror.aliyuncs.com"],

"bip": "172.24.20.1/24",

"exec-opts": ["native.cgroupdriver=systemd"],

"live-restore": true

}

# 配置项注意点：

# graph: 该关键字未来将被弃用，可以采用 "data-root" 替代

# storage-driver: 存储驱动，即分层文件系统

# insecure-registries: 不安全的docker registries，即使用http协议推拉镜象

# registry-mirrors: 加速站点，一般可以使用阿里、网易云、docker中国(https://registry.docker-cn.com)的地址

# bip: 指定docker bridge地址(不能以.0结尾)，生产中建议采用 172.xx.yy.1/24,其中xx.yy为宿主机ip后四位，方便定位问题

# 若启动失败，查看 /var/log/message 日志排错

[root@docker-24-20 ~]# docker info # Docker 信息查看

Client:

Debug Mode: false

Server:

Containers: 0 # 容器数量

Running: 0

Paused: 0

Stopped: 0

Images: 1 # 镜像数量

Server Version: 19.03.5 # server 版本

Storage Driver: overlay2

Backing Filesystem: xfs # 宿主机上的底层文件系统

Supports d\_type: true

Native Overlay Diff: true

Logging Driver: json-file

Cgroup Driver: systemd # Cgroups 驱动

Plugins:

Volume: local

Network: bridge host ipvlan macvlan null overlay

Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog

Swarm: inactive

Runtimes: runc

Default Runtime: runc

Init Binary: docker-init

containerd version: b34a5c8af56e510852c35414db4c1f4fa6172339

runc version: 3e425f80a8c931f88e6d94a8c831b9d5aa481657

init version: fec3683

Security Options:

seccomp

Profile: default

Kernel Version: 3.10.0-862.el7.x86\_64 # 宿主机的相关信息

Operating System: CentOS Linux 7 (Core)

OSType: linux

Architecture: x86\_64

CPUs: 2

Total Memory: 3.685GiB

Name: docker-24-20

ID: ASZO:NO2O:EJ2W:WGN5:MWAL:MBMP:TURI:WKNC:YUPS:E4E2:MATG:MXV3

Docker Root Dir: /data/docker # docker 数据存储目录

Debug Mode: false

Registry: https://index.docker.io/v1/ # registry 地址

Labels:

Experimental: false

Insecure Registries:

quay.io

registry.access.redhat.com

127.0.0.0/8

Registry Mirrors: # 加速站点

https://q2gr04ke.mirror.aliyuncs.com/

Live Restore Enabled: true

WARNING: bridge-nf-call-iptables is disabled # 当前两个warn需要设置内核模块

WARNING: bridge-nf-call-ip6tables is disabled

**4.2. Docker pull配置代理**

部分场景中，下载国外镜像非常慢，因此可以配置相关代理进行加速，方法如下：

**4.2.1. 配置宿主机代理**

1. 找到 Lantern 的代理服务器的http(s)监听地址，以当前宿主机 127.0.0.1:9768 为例

2. 配置windows端口的转发，将所有10.4.7.1的10080端口转发至 127.0.0.1:9768

netsh interface portproxy add v4tov4 listenaddress=10.4.7.1 listenport=10080 connectaddress=127.0.0.1 connectport=9768

3. 配置windows防火墙策略，允许 10.4.7.0/24 网段访问 10.4.7.1:10080

4. 配置Linux机器的http和https代理

[root@centos-7-50 ~]# export http\_proxy="10.4.7.1:10080"

[root@centos-7-50 ~]# export https\_proxy="10.4.7.1:10080"

5. 测试

[root@centos-7-50 ~]# curl -I https://www.google.com

HTTP/1.1 200 OK

Date: Friday, 20-Mar-20 09:55:18 CST

Keep-Alive: timeout=58

Content-Length: 0

# 实验环境的虚拟机使用的是 10.4.7.0/24 的NAT网络，宿主机地址为 10.4.7.1

**4.2.2. 配置docker代理**

root@ubuntu-7-61:~# mkdir /etc/systemd/system/docker.service.d

root@ubuntu-7-61:~# vim /etc/systemd/system/docker.service.d/http-proxy.conf

[Service]

Environment="HTTP\_PROXY=http://10.4.7.1:10080"

root@ubuntu-7-61:~# systemctl daemon-reload

root@ubuntu-7-61:~# systemctl restart docker

root@ubuntu-7-61:~# docker info 2>&1 | grep -i proxy

HTTP Proxy: http://10.4.7.1:10080

**02-Docker镜像**

**1. Docker镜像**

Docker 容器的运行是基于宿主机的内核，通过linux的namespaces来实现隔离，相对于虚拟机而言降低了硬件资源的性能损耗，且具备一定程度上的应用隔离效果。

另外，通过最小化的镜像方式运行docker 容器，相对于虚拟机而言大幅度降低非生产力进程的资源消耗。

最后使用打包好的镜像，统一了应用程序的运行环境，降低了开发和测试人员的负担，但是增加了运维工作的复杂度。

图示

描述已自动生成

Docker 客户端启动一个容器时，通过TCP或者unix socket连接docker daemon。如果需要启动的容器镜像本地不存在，则从registry来拉取到本地，然后根据传递的CMD或者默认的CMD来启动容器。

**Docker 镜像的命名方式： ${registry\_name}/${repository\_name}/${image\_name}:${tag\_name}。其中默认的registry\_name为docker.io，repository\_name默认值为library，默认tag\_name为lastest。lastest为一个指针，可以指向任意一个版本，一般都是指向最新的版本。非官方镜像必须要带上${repository\_name}，非docker-hub镜像必须带上 ${registry\_name}/${repository\_name}。**

**2. Docker 镜像相关操作**

图示

描述已自动生成

* 基于运行中的容器，手动制作镜像
* 将运行中的容器暂停(docker commit 默认暂停容器)，然后将当前的状态制作成镜像，注意需要指定repository,可以只要-a添加作者信息，-c修改其它字段信息，常修改CMD信息。
* 导出(export)当前容器的文件系统信息，导出后的格式为.tar，.tar文件是一个中间状态的文件，需要使用import导入并且设置CMD等相关信息才是一个完整的镜像。
* 自动化制作镜像 Dockerfile

Dockerfile 能解决两个问题：

* 制作过程自动化：原理也是先运行容器，然后再基于运行的容器制作镜像，但是将制作过程使用dockerfile预先指定好
* 可以引入变量：在制作镜像时手动传参，根据传递的参数制作不同的镜像；在创建容器时手动传参，根据传递的参数修改进程的配置
* 镜像导出和导入

这并不是制作镜像，而是共享或传输镜像到其它服务器的一种方式，如果局域网中存在registory，这种方式意义不大。

如果局域网中没有registory，通过公网下载镜像会变得非常慢，因此将需要的镜像在一台机器上打包好，分发到各台服务器上会更加节约时间。

**3. docker commit**

**3.1. 指令**

docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

docker container commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

----

-a,--author string Author (e.g., "John Hannibal Smith <hannibal@a-team.com>")

-c,change list Apply Dockerfile instruction to the created image

-m,--message string Commit message

-p,--pause Pause container during commit (default true)

**3.2. 案例**

**3.2.1. 针对运行中的容器创建镜像**

[root@centos-82 ~]# docker run --name img01 -it quay.io/quay/busybox

/ # echo 'busybox' > /tmp/a.txt

[root@centos-82 ~]# docker container commit -a "Heyingsheng <29492492@qq.com>" img01 ## 未指定仓库和标签的镜像

[root@centos-82 ~]# docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

<none> <none> 2fcb2d26d167 7 seconds ago 2.49MB

nginx latest f09fe80eb0e7 3 weeks ago 109MB

quay.io/quay/busybox latest c8cb1f026eeb 5 years ago 2.49MB

[root@centos-82 ~]# docker container commit -a "Heyingsheng <29492492@qq.com>" img01 devops/busybox:1.0.1 ## 指定仓库和标签

REPOSITORY TAG IMAGE ID CREATED SIZE

devops/busybox 1.0.1 a29845416384 3 seconds ago 2.49MB

<none> <none> 2fcb2d26d167 44 seconds ago 2.49MB

nginx latest f09fe80eb0e7 3 weeks ago 109MB

quay.io/quay/busybox latest c8cb1f026eeb 5 years ago 2.49MB

[root@centos-82 ~]# docker run --name busybox01 -it devops/busybox:1.0.1  ## 运行并查看

/ # cat /tmp/a.txt

busybox

**3.2.2. 创建镜像并修改CMD参数**

[root@centos-82 ~]# docker image inspect busybox:latest | grep -n CMD

31: "CMD [\"sh\"]"

[root@centos-82 ~]# docker container run --name busybox02 -it busybox

/ # mkdir -p /opt/webiste

/ # echo "<h1>Hello World</h1>" > /opt/webiste/index.html

/ # exit

[root@centos-82 ~]# docker container commit -a "Heyingsheng <33202492034@qq.com>" -c 'CMD ["/bin/httpd","-f","-h","/opt/website"]' busybox\_01 devops/apache:1.0.1 ## Every element in list must delemeted by ",".

[root@centos-82 ~]# docker container run --name web\_1.0.1 -d devops/apache:1.0.1

[root@centos-82 ~]# docker container inspect web\_1.0.1 | grep IPAddress

"SecondaryIPAddresses": null,

"IPAddress": "172.17.0.2",

"IPAddress": "172.17.0.2",

[root@centos-82 ~]# curl 172.17.0.2

<p>Hello World</p>

**4. image save & load**

**4.1. 指令**

docker [image] save [OPTIONS] IMAGE [IMAGE...]

Options:

-o, --output string Write to a file, instead of STDOUT

# ---

docker [image] load [OPTIONS]

Options:

-i, --input string Read from tar archive file, instead of STDIN

-q, --quiet Suppress the load output

**4.2. 案例**

**4.2.1 image save**

[root@centos-82 ~]# docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

devops/apache 1.0.1 d5fd027d1066 30 minutes ago 1.2MB

busybox latest d8233ab899d4 2 weeks ago 1.2MB

nginx latest f09fe80eb0e7 3 weeks ago 109MB

[root@centos-82 ~]# docker image save d5fd027d1066 d8233ab899d4 f09fe80eb0e7 -o deveops.tar.gz ## doesn't gzip

[root@centos-82 ~]# docker save d5fd027d1066 d8233ab899d4 f09fe80eb0e7 -o deveops2.tar ## no tag and repository

[root@centos-82 ~]# ls -lh

-rw------- 1 root root 110M Mar 3 11:04 deveops2.tar

-rw------- 1 root root 110M Mar 3 11:04 deveops.tar.gz

[root@centos-82 ~]# file deveops2.tar deveops.tar.gz

deveops2.tar: POSIX tar archive

deveops.tar.gz: POSIX tar archive

[root@centos-82 ~]# gunzip deveops.tar.gz

gzip: deveops.tar.gz: not in gzip format ## image save doesn't gzip images.

[root@centos-82 ~]# tar tf deveops2.tar

0d59d15c626f2cb0a2d5cdfa08bc3e255e7d151819fde4b1981c7e5ded237a77/

0d59d15c626f2cb0a2d5cdfa08bc3e255e7d151819fde4b1981c7e5ded237a77/VERSION

0d59d15c626f2cb0a2d5cdfa08bc3e255e7d151819fde4b1981c7e5ded237a77/json

0d59d15c626f2cb0a2d5cdfa08bc3e255e7d151819fde4b1981c7e5ded237a77/layer.tar

......

manifest.json

[root@centos-82 ~]# docker image save nginx:latest busybox:latest devops/apache:1.0.1 -o output.tar

**4.2.2 image load 案例**

[root@centos-81 ~]# docker image load -i /tmp/deveops2.tar

adab5d09ba79: Loading layer [==================================================>] 1.416 MB/1.416 MB

1b5d74440182: Loading layer [==================================================>] 5.12 kB/5.12 kB

0a07e81f5da3: Loading layer [==================================================>] 58.47 MB/58.47 MB

92c15149e23b: Loading layer [==================================================>] 54.44 MB/54.44 MB

6b5e2ed60418: Loading layer [==================================================>] 3.584 kB/3.584 kB

Loaded image ID: sha256:d5fd027d10667a28b75c3bbe10171eae51411a775849bab396fea45354327690

Loaded image ID: sha256:d8233ab899d419c58cf3634c0df54ff5d8acc28f8173f09c21df4a07229e1205

Loaded image ID: sha256:f09fe80eb0e75e97b04b9dfb065ac3fda37a8fac0161f42fca1e6fe4d0977c80

[root@centos-81 ~]# docker image ls -a ##  使用 image save 导出时需要加 repostory和tag

REPOSITORY TAG IMAGE ID CREATED SIZE

<none> <none> d5fd027d1066 41 minutes ago 1.2 MB

<none> <none> d8233ab899d4 2 weeks ago 1.2 MB

<none> <none> f09fe80eb0e7 3 weeks ago 109 MB

[root@centos-81 ~]# docker image rm d8233ab899d4 f09fe80eb0e7 d5fd027d1066

[root@centos-81 ~]# docker image load -i /tmp/output.tar

adab5d09ba79: Loading layer [==================================================>] 1.416 MB/1.416 MB

Loaded image: busybox:latest

1b5d74440182: Loading layer [==================================================>] 5.12 kB/5.12 kB

Loaded image: devops/apache:1.0.1

0a07e81f5da3: Loading layer [==================================================>] 58.47 MB/58.47 MB

92c15149e23b: Loading layer [==================================================>] 54.44 MB/54.44 MB

6b5e2ed60418: Loading layer [==================================================>] 3.584 kB/3.584 kB

Loaded image: nginx:latest

[root@centos-81 ~]# docker image ls -a # good images

REPOSITORY TAG IMAGE ID CREATED SIZE

devops/apache 1.0.1 d5fd027d1066 46 minutes ago 1.2 MB

busybox latest d8233ab899d4 2 weeks ago 1.2 MB

nginx latest f09fe80eb0e7 3 weeks ago 109 MB

**5. container export & import**

**5.1. 指令**

docker [container] export [OPTIONS] CONTAINER

Options:

-o, --output string Write to a file, instead of STDOUT

#---

docker [image] import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]

Options:

-c, --change list Apply Dockerfile instruction to the created image (default [])

-m, --message string Set commit message for imported image

**5.2. 案例**

**5.2.1. export**

[root@centos-82 ~]# docker container run --name web01 -it busybox # Run a container and httpd service

/ # mkdir -p /opt/website

/ # echo "<p>Container export</p>" > /opt/website/index.html

/ # httpd -f -h /opt/website/

[root@centos-82 ~]# docker container export -o web.tar web01 # Export container to an image.

**5.2.2. import**

[root@centos-81 ~]# docker image import /tmp/web.tar ## No reporsitory and tags,bad image

[root@centos-81 ~]# docker image import /tmp/web.tar httpd:v0.1 ## Specify reporsitory and tag.Not running httpd when start container.

[root@centos-81 ~]# docker image import -c 'CMD ["/bin/httpd","-f","-h","/opt/website"]' /tmp/web.tar httpd:v0.2 ## Specify reporsitory,tag and CMD.

[root@centos-81 ~]# docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

httpd v0.2 3f22152f75b7 9 seconds ago 1.2 MB

httpd v0.1 562ec613ec3a About an hour ago 1.2 MB

<none> <none> 5010a1bb08b7 About an hour ago 1.2 MB

devops/apache 1.0.1 d5fd027d1066 2 hours ago 1.2 MB

busybox latest d8233ab899d4 2 weeks ago 1.2 MB

nginx latest f09fe80eb0e7 3 weeks ago 109 MB

**5.3. export & import VS save & load**

**5.3.1. image save VS container export**

* Command docker save is copy old\_image to file.tar.It keep old image\_name,image\_tag and keep all the layers
* Command docker export is copy container to file.tar.It keep state of container and  combine(合并) multiple(众多) layers into one layer

**5.3.2. image load VS image import**

* Command docker load is using for improt file.tar which is export by docker save
* Command docker import is using for import file.tar which is export by docker export

# 03-Dockerfile

## 1. 语法

### 1.1. 语法要求

* 执行docker build的目录为镜像创建的工作目录，所有被Dockerfile引用的文件都必须在此目录或者其子目录中，不可超出工作目录的边界
* Dockerfile可以引用文件也可以引用目录，支持通配符，对于引用的目录中个别不需要的文件可以将路径写入.dockerignore文件中，该文件同样支持通配符
* 注释以 # 开头，类似于shell 脚本
* 非注释的非空行是指令行，指令不区分大小写，但是约定都是大写，一个指令是镜像的一个层，因此需要合理编排，尽可能减少层级
* Dockerfile支持变量的引用，引用时与shell脚本一致: $var , ${var} , ${var:-default} , ${var:+word}

### 1.2. 指令

#### 1.2.1. # escape

* **Syntax**

# escape=\

# escape=`

* **Introduction**

The default escape is \,usually used in RUN directive.But in windows image,file path delimiter is \,so the ` is better choice in create window image.

#### 1.2.2. FROM

* **Syntax**

FROM image[:tag] [AS name]

FROM image[@digest] [AS name]

* **Introduction**

Specify base image,default tag is :latest.You can speicify tag or digest.If local host doesn't has the image,it will pull from repository.Digest is image hash.

#### 1.2.3. MAINTAINER(deprecated)

* **Syntax**

MAINTAINER strings

* **Introduction**

Add author information to image.The directive is deprecated,the new choice is LABEL command.

#### 1.2.4. LABEL

* **Syntax**

LABEL key=vaule key=vaule ……

* **Introduction**

Add label to image,such as author,version.In label,you can use a "\" to continue a single key-vaule to next line.

#### 1.2.5. COPY

* **Syntax**

COPY [--chown=user:group] src …… dest

COPY [--chown=user:group] ["src","src",……,"dest"]

* **Introduction**

Chown option only worked on Linux container.

COPY instruction copy files or directory based on docker-build workdir to container destination path.The src path support wildcards which will be done using Go's filepath match rules,such as "hom\* hom?.txt". The dest is an absolute path,or a path ralative to WORKDIR.

* If src is a directory,the whole contents of the directory are copied,include filesystem and metadata.
* If src is a directory,the directory is not copied,just its contents.
* If dest is ending with "/",it will be done an directory.Otherwise,it will be considered a regular file.
* If specify src by wildcards or sepcify multiple src,the dest must be direcotry.
* If dest doesn't exits,it will be created.
* If filename contain blank charactre,the JSON array is better choice

#### 1.2.6. ADD

* **Syntax**

ADD [--chown=user:group] src …… dest

ADD [--chown=user:group] ["src","src",……,"dest"]

* **Introduction**

The ADD instruction is like COPY instruction.But the instruction has some features:

* ADD can sepcify local files or remote file URLs from src.
* If src is URL,the destination will be chmod to 600.
* If src is tar archive format(.tar,.tar.gz,.tar.bz2,.tar.xz),it will be unpacked on destination(like tar -xf src).
* If scr is URL tar archive format,it will not be unpacked on destination.
* If URL requires authentication,you must be download by wget or curl on COM instruction.

#### 1.2.7. RUN

* **Syntax**

RUN command

RUN ["executable","param1","param2",…]

* **Introduction**

RUN command is running when you create image by "docker build".All commands must included in base image.

* "RUN command … " is shell form,the command is run in a shell,whinc by defualt is "/bin/sh -c" on Linux or "cmd /s /c" on Windows.

The default shell for shell form can be changed by "SHELL" command.

* "RUN ["executable","param1","param2",…]" is exec form,the command is run skep shell.
* In shell form you can use a "\" to continue a single RUN instruction onto next line.
* The exec form is parsed as a JSON array,so you must use double-quotes(") rather than single-quotes(').
* In windows image, RUN ["c:\windows\system32\tasklist.exe"] is syntax error.The correct syntax for this example is:RUN ["c:\\windows\\system32\\tasklist.exe"]]

#### 1.2.8. CMD

* **Syntax**

CMD command param1 param2

CMD ["executable","param1","param2"]

CMD ["param1","param2"]

* **Introduction**

The CMD directive used for define a default command when start a container.There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect.

* Shell format:

The first format is shell format.Run command by shell,default is "/bin/sh -c".

* Exec format:

The second format is exec format.It is preferred format.

The exec format skip shell to run command,so some enviroment or commands can't run success.

The exec format is JSON array,that need use double-quotes(").

* Entrypoint argument format

The third format is entrypoint argument format.

It define default arguments for ENTRYPOIN command.

#### 1.2.9. ENTRYPOINT

* **Syntax**

ENTRYPOINT command param1 param2

ENTRYPOINT ["executable","param1","param2"]

* **Introduction**

ENTRYPOINT is container default command when container start.The exec-form and shell-form has differences:

|  |  |  |  |
| --- | --- | --- | --- |
|  | No ENTRYPOINT | ENTRYPION e\_cmd e\_arg | ENTRYPOIN["e\_cmd","e\_arg"] |
| No CMD | Error | /bin/sh -c e\_cmd e\_arg | E\_cmd e\_arg |
| CMD c\_cmd c\_arg | /bin/sh -c c\_cmd c\_arg | /bin/sh -c e\_cmd e\_arg | E\_cmd e\_arg /bin/sh -c c\_cmd c\_arg |
| CMD ["c\_cmd","c\_arg"] | C\_cmd c\_arg | /bin/sh -c e\_cmd e\_arg | E\_cmd e\_arg c\_cmd c\_arg |
| CMD ["c\_arg"] | C\_arg | /bin/sh -c e\_cmd e\_arg | E\_cmd e\_arg c\_arg |

* When don't specify ENTRYPOIN,CMD will be container default Cmd.
* When specify exec-form ENTRYPOINT,CMD will become arguments for ENTRYPOIN directive.
* When specify shell-form ENTRYPOIN,CMD will be ignore.In most time,use "exec commnad".
* You can specify "--entrypoint string" overwrite ENTORYPIONT when run a container.
* The exec-from is better choice.

#### 1.2.10. EXPOSE

* **Syntax**

EXPOSE port[/protocol] ……

* **Introduction**

Set default expose port and protocol when you run container with "-P" optinon,default protocol is TCP.

Use the "-p" option will publish and map one or more ports.Use the "-P" option will publish all exposed ports and map to high-order ports.

#### 1.2.11. ENV

* **Syntax**

ENV key value

ENV key=vaule key=value …

* **Introduction**

Defines variable key to the value.The vaule be in the environment for all subsequent instruction in docker build,even you can change them when create and run a container by options "--env key=value".

When you need define multiple variables,you can use second format,and you also can use quotes include values.

#### 1.2.12. HEALTHCHECK

* **Syntax**

HEALTHCHECK [--interval=time --timeout=time --start-period=time --retries=number] CMD command

HEALTHCHECK NONE

* **Introduction**

Defines health check process to test container status.The first format is defined health check method.The second format is disable any healthcheck inherited from the base image.

Options:

* --interval: Sets time between tow check.Default 30s.
* --timeout:  Defines timeout time,default is 30s.
* --start-period: Defines seconds after container start to run health check.Default 0s.
* --retries: Retry times.Default 3.

Status:

* 0: Success, container is healthy.
* 1: Failed,container is not work correctly.

#### 1.2.13. VOLUME

* **Syntax**

VOLUME ["mounted\_point","mounted\_point",……]

VOLUME mounted\_point mounted\_point

* **Introduction**

Specify volume which will mounted by container-managed when container run.

**Docker will copy file to volume when the files created before volumes declared.After valume declared,any build steps change the data within the volume will be discarded.**

The volume can't specify host dirctory.

#### 1.2.14. USER

* **Syntax**

USER user[:group]

USER UID[:GID]

* **Introduction**

The USER command set user name or UID and group name or GID to use when running the image and for any RUN,CMD,and ENTRYPOIN directive.

When the user doesn't have a primary group when the image will be run with root group.

#### 1.2.15. WORKDIR

* **Syntax**

WORKDIR path

* **Introduction**

Specify work directory in container for any RUN,CMD,ENTRYPOIN,ADD,COPY instructions.If the path dosen't exist,it will be create.

#### 1.2.16. ONBULID

* **Syntax**

ONBUILD INSTRODUCTION

* **Introduction**

Defines introduction when the image referer by others.In other word,if other people create use base image which sets ONBUILD command,it will execute the INSTRODUCTION when referer.

#### 1.2.17. ARG

* **Syntax**

ARG key[=default\_value]

* **Introduction**

Define variable that user can pass value by "--build-arg key=value" when docker build.It use default value when user doesn't pass value and defines default in Dockerfile.

If you need specify image version,you can set "ARG VERSION" before FROM instruction,but the ARG instruction that before FROM will not be referer after FROM instruction,you must reset ARG instruction after FROM instruction.

#### 1.2.18. SHELL

* **Syntax**

SHELL ["executalbe","parameters"]

* **Introduction**

Defins shell which overwrite default shell ("/bin/sh -c" or "cmd /S /C").

#### 1.2.19. STOPSINGLE

* **Syntax**

STOPSIGNAL signal

* **Introduction**

Defines signal when execute "docker container stop",default is 15.

### 1.3. 多阶段构建

在学习go语言过程中，存在一个镜像制作的场景：

1. 下载 golang 官方镜像，使用Dockerfile将go代码进行编译，编译结果为一个二进制文件
2. 下载 业务容器基础镜像(比如centos)，将编译好的二进制文件通过Dockerfile拷贝到业务容器基础镜像中

为了实现上述的需求，以前仅有两种方式：分两个Dockerfile进行镜像的制作，使用shell脚本将两个制作流程合并。在 Docker 17.05 之后，Dockerfile 支持多阶段构建镜像，方式如下：

FROM golang:1.13 as builder # 编译代码的基础镜像

WORKDIR /workspace

COPY ./ .

RUN CGO\_ENABLED=0 GOOS=linux GOARCH=amd64 GO111MODULE=on GOPROXY=https://goproxy.cn go build -a -o test-project main.go

FROM centos:7

COPY --from=builder /workspace/test-project .

RUN chmod +x test-project

CMD ["/test-project"]

## 2. Examples

### 2.1. 常用的指令使用案例

#### 2.1.1. LABEL & COPY

[root@centos-82 test-01]# cp -r /etc/yum.repos.d/ ./

[root@centos-82 test-01]# cp /etc/passwd ./

[root@centos-82 test-01]# head -100 Dockerfile .dockerignore

==> Dockerfile <==

# Test FROM directive

FROM busybox:latest

LABEL author="heyingsheng <hys\_1992@outlook.com>" version="test-image-v1.0.1" baseimage="busybox:latest"

COPY yum.repos.d passwd /tmp/

==> .dockerignore <== ## 需要忽略的文件

yum.repos.d/CentOS-\*

[root@centos-82 ~]# docker container run --name c1 --rm test-image:v1.0.2 ls /tmp/

docker-ce.repo

passwd

[root@centos-82 ~]# docker image inspect -f {{.ContainerConfig.Labels}} test-image:v1.0.2

map[author:heyingsheng <hys\_1992@outlook.com> baseimage:busybox:latest version:test-image-v1.0.1]

#### 2.1.2. ADD

[root@centos-82 test-01]# cat Dockerfile

# Test FROM directive

FROM busybox:latest

LABEL author="heyingsheng <hys\_1992@outlook.com>" version="test-image-v1.0.1" baseimage="busybox:latest"

# URL 格式的文件不会解压，而直接 ADD 进去的本地 .tar[.xx] 格式文件会解压

ADD http://nginx.org/download/nginx-1.14.2.tar.gz /tmp/url\_tar/

ADD nginx-1.14.2.tar.gz /tmp/local\_tar/

[root@centos-82 test-01]# docker build -t test-image:v1.0.3 ./

[root@centos-82 ~]# docker container run --name c1 --rm test-image:v1.0.3 ls /tmp/url\_tar /tmp/local\_tar/

/tmp/local\_tar/:

nginx-1.14.2

/tmp/url\_tar:

nginx-1.14.2.tar.gz

#### 2.1.3. VOLUME

[root@centos-82 test-01]# cat Dockerfile

# Test FROM directive

FROM busybox:latest

LABEL author="heyingsheng <hys\_1992@outlook.com>" version="test-image-v1.0.1" baseimage="busybox:latest"

# COPY yum.repos.d passwd /tmp/

ADD nginx-1.14.2.tar.gz /tmp/local\_tar/

## 只能指定容器中的挂载点，而不能指定宿主机上的目录

VOLUME /tmp/

ADD passwd yum.repos.d /tmp/

RUN mv /tmp/local\_tar/nginx-1.14.2 /tmp/local\_tar/nginx\_src

[root@centos-82 test-01]# docker build -t test-image:v1.0.6 ./

[root@centos-82 ~]# docker container run --name c1 --rm test-image:v1.0.6 sh -c "ls /tmp/ /tmp/local\_tar ;sleep 60 "

/tmp/:

docker-ce.repo

local\_tar

passwd

/tmp/local\_tar:

nginx-1.14.2

#### 此处最后的RUN并没有修改声明volume之前创建的文件

[root@centos-82 test-01]# docker container inspect -f {{.Mounts}} c1

[{volume fb376e4f9dd94ec8b4e8a2778bd15b405dda653f3773ca4cb53fd120170eb6f4 /var/lib/docker/volumes/fb376e4f9dd94ec8b4e8a2778bd15b405dda653f3773ca4cb53fd120170eb6f4/\_data /tmp local true }]

#### 2.1.4. EXPOSE

[root@centos-82 test-01]# cat Dockerfile

# Test FROM directive

FROM busybox:latest

LABEL author="heyingsheng <hys\_1992@outlook.com>" version="test-image-v1.0.1" baseimage="busybox:latest"

COPY index.html /data/web/html/

EXPOSE 80/tcp

[root@centos-82 test-01]# docker build -t test-image:v1.0.8 ./

[root@centos-82 ~]# docker container run --name c1 --rm test-image:v1.0.8 /bin/httpd -f -h /data/web/html/

[root@centos-82 test-01]# docker container port c1 ## Default case doesn't expose 80 prot.

[root@centos-82 ~]# docker container run --name c1 -P --rm test-image:v1.0.8 /bin/httpd -f -h /data/web/html/

[root@centos-82 test-01]# docker container port c1 ## When you specify "-P",it will expose 80

80/tcp -> 0.0.0.0:32769

[root@centos-82 test-01]# curl 192.168.1.82:32769

<h1>Httpd Server</h1>

#### 2.1.5. CMD

##### 2.1.5.1. exec-form CMD

[root@centos-82 test-02]# vim Dockerfile

FROM busybox:latest AS baseimage

LABEL Author="heyingsheng <123@qq.com>"

ENV WEB\_ROOT="/data/html/www"

RUN mkdir -p $WEB\_ROOT && \

echo "Test index!" > $WEB\_ROOT/index.html

# CMD ["/bin/httpd","-f","-h","$WEB\_ROOT"] ## The exec format can't referer shell variable.

CMD ["/bin/httpd","-f","-h","/data/html/www"]

[root@centos-82 test-02]# docker build -t test-image:v2.0.2 ./

[root@centos-82 ~]# docker image inspect -f {{.Config.Cmd}} test-image:v2.0.2

[/bin/httpd -f -h /data/html/www]

[root@centos-82 ~]# docker run --name c2 --rm -d test-image:v2.0.2

[root@centos-82 ~]# docker container inspect -f {{.NetworkSettings.IPAddress}} c2

172.17.0.2

[root@centos-82 ~]# curl 172.17.0.2

Test index!

##### 2.1.5.2. Shell-form CMD

[root@centos-82 test-02]# cat Dockerfile

FROM busybox:latest AS baseimage

LABEL Author="heyingsheng <123@qq.com>"

ENV WEB\_ROOT="/data/html/www"

RUN mkdir -p $WEB\_ROOT && \

echo "Test index!" > $WEB\_ROOT/index.html

# CMD ["/bin/httpd","-f","-h","$WEB\_ROOT"]

# CMD ["/bin/httpd","-f","-h","/data/html/www"]

CMD /bin/httpd -f -h /data/html/www

[root@centos-82 test-02]# docker build -t test-image:v2.0.3 ./

[root@centos-82 ~]# docker image inspect -f {{.Config.Cmd}} test-image:v2.0.3

[/bin/sh -c /bin/httpd -f -h /data/html/www]

[root@centos-82 ~]# docker container run --name c1 --rm -d test-image:v2.0.3

[root@centos-82 ~]# docker container inspect -f {{.NetworkSettings.IPAddress}} c1

172.17.0.2

[root@centos-82 ~]# curl 172.17.0.2 ## Upper success.

Test index!

[root@centos-82 ~]# docker container inspect -f {{.Config.Cmd}} c1 ## The container first process is /bin/sh.

[/bin/sh -c /bin/httpd -f -h /data/html/www]

[root@centos-82 ~]# docker container exec c1 ps ## The PID 1 process is replaced!

PID USER TIME COMMAND

1 root 0:00 /bin/httpd -f -h /data/html/www

#### 2.1.6. ENTRYPOINT

##### 2.1.6.1. exec-form ENTRYPOINT

[root@centos-82 test-02]# cat Dockerfile

FROM busybox:latest

LABEL Author="heyingsheng <123@qq.com>"

RUN mkdir -p /data/web/html && \

echo "Test index!" > /data/web/html/index.html

CMD ["-h","/data/web/html"]

ENTRYPOINT ["/bin/httpd","-f"]

[root@centos-82 test-02]# docker build -t test-image:v2.0.6 ./

[root@centos-82 ~]# docker container run --name c1 --rm -d test-image:v2.0.7

[root@centos-82 ~]# docker container exec c1 ps uax ## The CMD arguments will append to ENTRYPOINT.

PID USER TIME COMMAND

1 root 0:00 /bin/httpd -f -h /data/web/html

6 root 0:00 ps uax

2.1.6.2. shell-form ENTRYPOINT

[root@centos-82 test-02]# cat Dockerfile

FROM busybox:latest

LABEL Author="heyingsheng <123@qq.com>"

RUN mkdir -p /data/web/html && \

echo "Test index!" > /data/web/html/index.html

CMD ["-h","/data/web/html"]

# ENTRYPOINT ["/bin/httpd","-f"]

ENTRYPOINT /bin/httpd -f

[root@centos-82 test-02]# docker build -t test-image:v2.0.7 ./

[root@centos-82 ~]# docker container run --name c1 --rm -d test-image:v2.0.7

[root@centos-82 ~]# docker container exec c1 ps uax ## The CMD directive is ignored.

PID USER TIME COMMAND

1 root 0:00 /bin/httpd -f

6 root 0:00 ps uax

2.1.6.3. configuration & ENTRYPOINT

[root@centos-82 test-02]# cat Dockerfile

FROM centos:centos7

LABEL Author="heyingsheng <123@qq.com>"

ADD http://nginx.org/download/nginx-1.14.2.tar.gz /usr/local/src/

RUN yum install -y gcc gcc-c++ pcre pcre-devel zlib zlib-devel openssl openssl-devel && \

useradd -M -s /sbin/nologin nginx && \

tar -xf /usr/local/src/nginx-1.14.2.tar.gz -C /usr/local/src/ && \

cd /usr/local/src/nginx-1.14.2 && \

./configure --user=nginx --group=nginx && \

make -j 4 && make install && \

mkdir -p /data/web/html/ && \

echo "Vhost Test!" > /data/web/html/index.html && \

yum clean all && \

cd / && \

rm -fr /var/cache/yum /usr/local/src/\*

EXPOSE 80/tcp

ADD nginx\_start.sh /bin/

ENV SERVER\_NAME="www.heyang.com" \

PORT=8080

CMD ["/usr/local/nginx/sbin/nginx"]

ENTRYPOINT ["/bin/nginx\_start.sh"]

[root@centos-82 test-02]# cat nginx\_start.sh

#!/bin/sh

cat <<EOF >/usr/local/nginx/conf/nginx.conf

user nginx;

worker\_processes 1;

events {

worker\_connections 1024;

}

daemon off;

http {

include mime.types;

default\_type application/octet-stream;

sendfile on;

keepalive\_timeout 65;

server {

listen ${IP:-0.0.0.0}:${PORT:-80};

server\_name ${SERVER\_NAME:localhost};

location / {

root /data/web/html/;

index index.html index.htm;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

}

EOF

exec $@

[root@centos-82 ~]# docker container run --name c1 --rm --env PORT=8081 -d test-image:v2.0.11

[root@centos-82 ~]# docker container exec c1 ps uax

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.0 20548 1448 ? Ss 23:31 0:00 nginx: master process /usr/local/nginx/sbin/nginx

[root@centos-82 ~]# curl 172.17.0.2:8081

Vhost Test!

# 04-Network

## 1. 常见的容器之间通信方式

### 1.1.  同一个宿主机内部

* 通过虚拟交换机来实现容器间通信

图片包含 箱线图

描述已自动生成

* 如果容器在不同的虚拟交换机上，则可以通过交换机直连或者路由转发的方式通信

图示

描述已自动生成

### 1.2. 跨宿主机的容器通信

* 通过桥接宿主机网卡的方式通信

图示

描述已自动生成

* NAT转发(docker默认的方式)

图示

描述已自动生成

* 叠加式网络(Overlay network)

图示

描述已自动生成

## 2. Docker网络模型

图示

描述已自动生成

1. 封闭式容器(none)

仅有一个lo口的容器，一般用于内部的数据处理等任务，而不需要与外界进行通信的容器。

1. 桥接式容器(Bridge,default)

每个容器在创建之初就分配了私有地址和lo口，其中私有地址接在docker的虚拟桥上，可以通过NAT方式与外界通信，这是默认的网络模型。

1. 联盟式容器(Joined)

多个容器共用同一个Net名称空间，而其它名称空间是私有的，这些容器内部可以通过lo口直接通信，而不需要走docker桥。

1. 开放式容器(Open)

开放式容器是联盟式容器的一种特例，此容器共享的是宿主机的网卡

## 3. 案例

### 3.1 Docker 容器网络信息查看

[root@centos-82 ~]# ip link show

## veth3c68900@if4 一半在docker0上，一半在容器中。master接口是 docker0

......

3: docker0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default

link/ether 02:42:86:d7:21:83 brd ff:ff:ff:ff:ff:ff

5: veth3c68900@if4: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue master docker0 state UP mode DEFAULT group default

link/ether 26:62:60:c8:23:f6 brd ff:ff:ff:ff:ff:ff link-netnsid 0

[root@centos-82 ~]# docker network ls

NETWORK ID NAME DRIVER SCOPE

19eb0f999458 bridge bridge local ## 默认方式，桥接

b12765b5a394 host host local ## 共享宿主机网络

67019894af82 none null local ## 封闭式容器，仅lo口

[root@centos-82 ~]# docker network inspect bridge

"Name": "bridge",

......

"Driver": "default",

......

"Subnet": "172.17.0.0/16", ## 子网地址池

"Gateway": "172.17.0.1" ## 网关

......

"Containers": { ## 包含的容器信息

"2efec0177ae1ed1d6f3705f6357f7c99157f2a74daa2082108c87a69a6b45e8c": {

"Name": "web01",

"EndpointID": "ca991021b1d70462551127cc34753ba1f745ab9c592ac3be56903693759c9132",

"MacAddress": "02:42:ac:11:00:02",

"IPv4Address": "172.17.0.2/16",

......

"com.docker.network.bridge.name": "docker0", ## 网络名称

"com.docker.network.driver.mtu": "1500" ## MTU

......

### 3.2. 模拟网络名称空间

[root@centos-82 ~]# ip netns add ns1   ## 创建网络名称空间

[root@centos-82 ~]# ip netns add ns2

[root@centos-82 ~]# ip netns list      ## 查看当前网络名称空间

ns2

ns1

[root@centos-82 ~]# ip netns exec ns1 ip a  ## 在指定的网络名称空间中执行命令

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

[root@centos-82 ~]# ip link add name veth1.1 type veth peer name veth1.2 ## 创建一对虚拟网卡

[root@centos-82 ~]# ip link show ## veth1.1和veth1.2是一对网卡

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: ens32: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP mode DEFAULT group default qlen 1000

link/ether 00:0c:29:88:a8:40 brd ff:ff:ff:ff:ff:ff

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default

link/ether 02:42:5d:21:bc:03 brd ff:ff:ff:ff:ff:ff

4: veth1.2@veth1.1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000

link/ether a6:b3:2f:16:12:46 brd ff:ff:ff:ff:ff:ff

5: veth1.1@veth1.2: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000

link/ether de:90:83:56:79:1a brd ff:ff:ff:ff:ff:ff

[root@centos-82 ~]# ip link set dev veth1.2 netns ns1 ## 将其中一个网卡一半接入网络名称空间

[root@centos-82 ~]# ip netns exec ns1 ip a            ## 接入的网卡处于未激活状态

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

4: veth1.2@if5: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000

link/ether a6:b3:2f:16:12:46 brd ff:ff:ff:ff:ff:ff link-netnsid 0

[root@centos-82 ~]# ip netns exec ns1 ip link set dev veth1.2 name eth0  ## 对网卡设备改名

[root@centos-82 ~]# ip netns exec ns1 ip a

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

4: eth0@if5: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000

link/ether a6:b3:2f:16:12:46 brd ff:ff:ff:ff:ff:ff link-netnsid 0

[root@centos-82 ~]# ip netns exec ns1 ifconfig -a

eth0: flags=4098<BROADCAST,MULTICAST> mtu 1500

ether a6:b3:2f:16:12:46 txqueuelen 1000 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=8<LOOPBACK> mtu 65536

loop txqueuelen 1000 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@centos-82 ~]# ifconfig veth1.1 10.154.2.2         ## 激活宿主机的网卡

[root@centos-82 ~]# ip netns exec ns1 ifconfig eth0 10.154.2.3  ## 激活名称空间中的网卡

[root@centos-82 ~]# ip netns exec ns1 ifconfig -a

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 10.154.2.3 netmask 255.0.0.0 broadcast 10.255.255.255

inet6 fe80::a4b3:2fff:fe16:1246 prefixlen 64 scopeid 0x20<link>

ether a6:b3:2f:16:12:46 txqueuelen 1000 (Ethernet)

RX packets 6 bytes 508 (508.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 6 bytes 508 (508.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=8<LOOPBACK> mtu 65536

loop txqueuelen 1000 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@centos-82 ~]# ping -c 2 -w 1 10.154.2.3 ## 宿主机与网络名称空间中的网卡通信

PING 10.154.2.3 (10.154.2.3) 56(84) bytes of data.

64 bytes from 10.154.2.3: icmp\_seq=1 ttl=64 time=0.046 ms

64 bytes from 10.154.2.3: icmp\_seq=2 ttl=64 time=0.045 ms

[root@centos-82 ~]# ip link set dev veth1.1 netns ns2  ## 将宿主机中的另一半虚拟网卡移动到另一个网络名称空间中

[root@centos-82 ~]# ip netns exec ns2 ifconfig -a

lo: flags=8<LOOPBACK> mtu 65536

loop txqueuelen 1000 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

veth1.1: flags=4098<BROADCAST,MULTICAST> mtu 1500

ether de:90:83:56:79:1a txqueuelen 1000 (Ethernet)

RX packets 12 bytes 928 (928.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 12 bytes 928 (928.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@centos-82 ~]# ip netns exec ns2 ip link set dev veth1.1 name eth0

[root@centos-82 ~]# ip netns exec ns2 ifconfig eth0 10.154.2.9  ## 激活网卡

[root@centos-82 ~]# ip netns exec ns2 ifconfig -a

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 10.154.2.9 netmask 255.0.0.0 broadcast 10.255.255.255

inet6 fe80::dc90:83ff:fe56:791a prefixlen 64 scopeid 0x20<link>

ether de:90:83:56:79:1a txqueuelen 1000 (Ethernet)

RX packets 12 bytes 928 (928.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 18 bytes 1436 (1.4 KiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=8<LOOPBACK> mtu 65536

loop txqueuelen 1000 (Local Loopback)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@centos-82 ~]# ip netns exec ns2 ping -c 1 -w 1 10.154.2.3 ## 两个名称空间实现通信

PING 10.154.2.3 (10.154.2.3) 56(84) bytes of data.

64 bytes from 10.154.2.3: icmp\_seq=1 ttl=64 time=0.045 ms

### 3.3. 容器的网络配置

#### 3.3.1. 容器创建时指定

[root@centos-82 ~]# docker run --name bx1 -it --rm busybox:latest  ## 默认的主机名是ID

/ # hostname

d7e05d8b2fe6

[root@centos-82 ~]# docker run --name bx1 --hostname container-test-bx1 -it --rm busybox:latest ## 手动指定主机名

/ # hostname

container-test-bx1

/ # cat /etc/hosts ## 默认hosts文件配置

127.0.0.1 localhost

::1 localhost ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

172.17.0.2 container-test-bx1

/ # cat /etc/resolv.conf ## 默认使用了宿主机的DNS服务器

# Generated by NetworkManager

nameserver 223.5.5.5

nameserver 114.114.114.114

[root@centos-82 ~]# docker run --name bx1 --hostname container-test-bx1 --dns 172.17.0.1 --dns 8.8.8.8 --add-host bbs.heyang.com:192.168.1.80 -it --rm busybox:latest

/ # cat /etc/hosts

127.0.0.1 localhost

::1 localhost ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

192.168.1.80 www.heyang.com

192.168.1.80 bbs.heyang.com

172.17.0.2 container-test-bx1

/ # cat /etc/resolv.conf

nameserver 172.17.0.1

nameserver 8.8.8.8

#### 3.3.2. 修改初始化配置

* Docker 网络配置参数

{

"bip": "192.168.1.5/24", # docker0桥的IP，定义这个IP后，其它非DNS的字段会自动计算

"fixed-cidr": "10.20.0.0/16", # IPv4 地址池

"fixed-cidr-v6": "2001:db8::/64", # IPv6 地址池

"mtu": 1500, # MTU 数据包分片大小

"default-gateway": "10.20.1.1", # 默认IPv4网关

"default-gateway-v6": "2001:db8:abcd::89", # 默认IPv6网关

"dns": ["10.20.1.1","20.20.1.3"] # 默认DNS地址

}

[root@centos-82 ~]# ip a show docker0

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default

inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0

......

[root@centos-82 ~]# vim /etc/docker/daemon.json

{

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

"bip":"10.154.50.1/24",

"dns":["114.114.114.114","192.168.1.99"]

}

[root@centos-82 ~]# systemctl restart docker

[root@centos-82 ~]# ip a show docker0

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default

inet 10.154.50.1/24 brd 10.154.50.255 scope global docker0

......

[root@centos-82 ~]# docker container run --name bx1 --rm -it busybox:latest

/ # ifconfig

eth0 Link encap:Ethernet HWaddr 02:42:0A:9A:32:02

inet addr:10.154.50.2 Bcast:10.154.50.255 Mask:255.255.255.0

......

/ # cat /etc/resolv.conf

nameserver 114.114.114.114

nameserver 192.168.1.99

#### 3.3.3. 添加新的网络设备

[root@centos-82 ~]# docker network ls

NETWORK ID NAME DRIVER SCOPE

f6fc3bdd3b71 bridge bridge local

b12765b5a394 host host local

67019894af82 none null local

[root@centos-82 ~]# docker network create -d bridge --subnet "10.175.0.0/16" --gateway "10.175.0.1" mybr0

3ad60ea3b43e8f1a432212f51958216cabcbab7b34679dab9af6e1139c96a0f6

[root@centos-82 ~]# docker network ls

NETWORK ID NAME DRIVER SCOPE

f6fc3bdd3b71 bridge bridge local

b12765b5a394 host host local

3ad60ea3b43e mybr0 bridge local

67019894af82 none null local

[root@centos-82 ~]# ip addr show

......

4: br-3ad60ea3b43e: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default

link/ether 02:42:cd:97:32:bd brd ff:ff:ff:ff:ff:ff

inet 10.175.0.1/16 brd 10.175.255.255 scope global br-3ad60ea3b43e

valid\_lft forever preferred\_lft forever

[root@centos-82 ~]# docker container run --name bx1 --network mybr0 -it --rm busybox:latest

/ # ip a

......

5: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER\_UP,M-DOWN> mtu 1500 qdisc noqueue

link/ether 02:42:0a:af:00:02 brd ff:ff:ff:ff:ff:ff

inet 10.175.0.2/16 brd 10.175.255.255 scope global eth0

valid\_lft forever preferred\_lft forever

### 3.4. Docker 网络模型

#### 3.4.1. None模型(仅lo)

[root@centos-82 ~]# docker container run --name bx1 --network none -it --rm busybox:latest  ## none模式

/ # ip a

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue 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

#### 3.4.2. Bridge模型(docker0)

##### 3.4.2.1. 指定Bridge模式

[root@centos-82 ~]# docker container run --name bx1 -it --rm busybox:latest  ## 默认bridge模式

/ # ip a

.......

6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER\_UP,M-DOWN> mtu 1500 qdisc noqueue

link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff

inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0

valid\_lft forever preferred\_lft forever

[root@centos-82 ~]# docker container run --name bx1 --network bridge -it --rm busybox:latest

/ # ip a

......

8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER\_UP,M-DOWN> mtu 1500 qdisc noqueue

link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff

inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0

valid\_lft forever preferred\_lft forever

##### 3.4.2.2. NAT 转发

1. 宿主机所有地址中的随机端口

[root@centos-82 ~]# docker container run --name web -p 80 --rm nginx:latest

[root@centos-82 ~]# docker container port web  ## 查看容器的端口映射

80/tcp -> 0.0.0.0:32768

[root@centos-82 ~]# netstat -lntp | grep docker ##监听在随机端口 32768

tcp6 0 0 :::32768 :::\* LISTEN 3441/docker-proxy

[root@centos-82 ~]# docker container inspect web | grep -w IPAddress ## 当前IP

"IPAddress": "172.17.0.2",

[root@centos-82 ~]# iptables -t nat -nvL | grep 172.17.0.2 ## 自动创建DNAT规则

0 0 MASQUERADE tcp -- \* \* 172.17.0.2 172.17.0.2 tcp dpt:80

1 60 DNAT tcp -- !docker0 \* 0.0.0.0/0 0.0.0.0/0 tcp dpt:32768 to:172.17.0.2:80

[root@centos-82 ~]# curl -s -I 172.17.0.2 | grep ^HTTP  ## 宿主机访问

HTTP/1.1 200 OK

[root@centos-50 ~]# curl -s -I 192.168.1.82:32768  | grep HTTP  ## 外部网络访问

HTTP/1.1 200 OK

1. 宿主机指定IP的随机端口

[root@centos-82 ~]# docker container run --name web -p 192.168.1.82::80 --rm nginx:latest

[root@centos-82 ~]# docker container port web

80/tcp -> 192.168.1.82:32768

1. 宿主机指定IP的指定端口

[root@centos-82 ~]# docker container run --name web -p 192.168.1.82:80:80 --rm nginx:latest

[root@centos-82 ~]# docker container port web

80/tcp -> 192.168.1.82:80

[root@centos-50 ~]# curl -s -I 192.168.1.82  | grep HTTP ## 外界可以直接访问

HTTP/1.1 200 OK

1. 宿主机所有地址中的指定端口

[root@centos-82 ~]# docker container run --name web -p 80:80 --rm nginx:latest

[root@centos-82 ~]# docker container port web

80/tcp -> 0.0.0.0:80

[root@centos-82 ~]# curl -s -I 127.0.0.1 | grep ^HTTP

HTTP/1.1 200 OK

[root@centos-50 ~]# curl -s -I 192.168.1.82  | grep HTTP

HTTP/1.1 200 OK

1. 宿主机多个端口映射到容器

[root@centos-82 ~]# docker container run --name web -p 80:80 -p 8080:80 --rm nginx:latest

[root@centos-82 ~]# docker container port web

80/tcp -> 0.0.0.0:8080

80/tcp -> 0.0.0.0:80

#### 3.4.3. Join 模型(联盟式)

[root@centos-82 ~]# docker run --name bx1 -it --rm busybox:latest

/ # ip a

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue 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

32: eth0@if33: <BROADCAST,MULTICAST,UP,LOWER\_UP,M-DOWN> mtu 1500 qdisc noqueue

link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff

inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0

valid\_lft forever preferred\_lft forever

/ # mkdir -p /data/web

/ # echo 'bx1' > /data/web/index.html

/ # httpd -h /data/web/

[root@centos-82 ~]# docker container run --name bx2 -it --rm  --network container:bx1 busybox:latest

/ # ip a ## 共享IP地址，但是其它名称空间隔离

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue 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

32: eth0@if33: <BROADCAST,MULTICAST,UP,LOWER\_UP,M-DOWN> mtu 1500 qdisc noqueue

link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff

inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0

valid\_lft forever preferred\_lft forever

/ # wget -O - -q 127.0.0.1 ## 可以通过lo访问bx1中的web服务

bx1

/ # ls /data/web ## Mount名称空间隔离

ls: /data/web: No such file or directory

#### 3.4.4. Open 模型(共享宿主机网络)

[root@centos-82 ~]# docker run --name bx1 -it --network host --rm busybox:latest

/ # ip a ## 显示的是宿主机的网络

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue 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: ens32: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast qlen 1000

link/ether 00:0c:29:88:a8:40 brd ff:ff:ff:ff:ff:ff

inet 192.168.1.82/24 brd 192.168.1.255 scope global ens32

valid\_lft forever preferred\_lft forever

inet6 fe80::76f9:ec2b:13f3:2b18/64 scope link

valid\_lft forever preferred\_lft forever

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue

link/ether 02:42:5d:21:bc:03 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::42:5dff:fe21:bc03/64 scope link

valid\_lft forever preferred\_lft forever

/ # mkdir /tmp/test/

/ # echo 'bx1' > /tmp/test/index.html

/ # httpd -h /tmp/test

[root@centos-82 ~]# netstat -lntp  ## 在本机上可以直接看到httpd进程

tcp6 0 0 :::80 :::\* LISTEN 5796/httpd

[root@centos-82 ~]# ps uax|grep http

root 5796 0.0 0.0 1272 40 ? Ss 23:01 0:00 httpd -h /tmp/test

[root@centos-82 ~]# ls /tmp/test  ## Mounts仍然是隔离的

ls: cannot access /tmp/test: No such file or directory

[root@centos-82 ~]# curl 127.0.0.1  ##  可以直接访问

bx1

**05-Volumes**

**1. Docker 卷结构**

Docker 镜像是分层结构，联合挂载，镜像中各层的文件都是只读的，容器中所有文件对外可见的状态是多层叠加后的可见状态。因此对于IO性能要求较高的容器，如果使用Overlayfs会严重影响容器的性能。

另外，容器在生命周期结束的时候会被删除，对于有数据持久化要求的容器而言，容器运行中有价值的数据并需要存储在外部的逻辑卷中。如果由容器直接使用共享存储系统，那么每次容器启动后都需要挂载，而且还需要具备与外界通信的能力，不仅繁琐，而且损失性能。

针对这种需求，docker推出了卷管理系统，在容器创建时直接使用宿主机的存储系统，当容器结束运行时，不会删除卷中的数据。

图示

描述已自动生成

如图中，宿主机将远程NFS服务器中的数据盘挂载到本地/data/目录下，创建httpd容器时，将本地的/data/盘映射到容器中的网站根目录:/data/html中。

当httpd容器挂掉后，无论在哪台服务器，只要满足宿主机挂载NFS存储，并且httpd容器映射网站根目录到/data/html即可。针对不需要持久化存储的应用，则可以不使用存储卷管理。

将宿主机上的目录挂载到容器中有两种方式：

* Bind mount volume

手动指定宿主机目录和容器中目录的映射关系，或使用容器编排工具(k8s或者docker-compose)指定

* Docker-managed volume

Docker自动管理的挂载关系，这种情况下宿主机目录不固定，后期维护成本高，仅用于存放一些临时数据

**2. 案例**

**2.1. Docker-managed volume**

[root@centos-82 ~]# docker container run --name web01 -v /data/ -it --rm busybox:latest ## Create container by docker-managed volumes

/ # df -h | grep data

/dev/sda3 15.5G 2.4G 13.1G 16% /data

[root@centos-82 ~]# docker container inspect -f {{.Mounts}} web01 ## show volumes info

[{volume e001f57178528126c1ae0dc4d4edbf1e0c797a903031f91f9fc8484efacd4183 /var/lib/docker/volumes/e001f57178528126c1ae0dc4d4edbf1e0c797a903031f91f9fc8484efacd4183/\_data /data local true }]

[root@centos-82 ~]# ls /var/lib/docker/volumes/e001f57178528126c1ae0dc4d4edbf1e0c797a903031f91f9fc8484efacd4183/\_data

/ # echo 'docker-mananged volume' > /data/test.txt ## write data to volume in container

[root@centos-82 ~]# cat /var/lib/docker/volumes/e001f57178528126c1ae0dc4d4edbf1e0c797a903031f91f9fc8484efacd4183/\_data/test.txt

docker-mananged volume

[root@centos-82 ~]# echo 'host docker-managed volume' >> /var/lib/docker/volumes/e001f57178528126c1ae0dc4d4edbf1e0c797a903031f91f9fc8484efacd4183/\_data/test.txt

/ # cat /data/test.txt

docker-mananged volume

host docker-managed volume

**2.2. Docker-managed volume**

[root@centos-82 ~]# docker run --name web01 -it --rm -v /data/website:/data/ busybox:latest ## Bind mount volume

/ # df -h | grep data

/dev/sda3 15.5G 2.4G 13.1G 16% /data

[root@centos-82 ~]# docker container inspect -f {{.Mounts}} web01

[{bind /data/website /data true rprivate}]

/ # echo 'bind mount volume' > /data/test.txt ## Test

[root@centos-82 ~]# cat /data/website/test.txt

bind mount volume

[root@centos-82 ~]# echo 'host bind mount volume' >> /data/website/test.txt

[root@centos-82 ~]# cat /data/website/test.txt

bind mount volume

host bind mount volume

/ # cat /data/test.txt

bind mount volume

host bind mount volume

/ # exit ## Exit

[root@centos-82 ~]# cat /data/website/test.txt ## When container exit,the data also exist.

bind mount volume

host bind mount volume

[root@centos-82 ~]# docker run --name web01 -it --rm -v /data/website:/data/ busybox:latest ## Restart container

/ # cat /data/test.txt

bind mount volume

host bind mount volume

**06-Registry**

**1. Docker Registry 分类**

Docker Registry 有两部分[组成](https://www.yuque.com/heyingsheng/trp3ic/cseut0" \l "pWjAj" \t "_blank)：

* Repostitory
* 由特定的一组镜像组成的仓库称为 Repostitory
* 一个 Docker Registry 可以包含多个 Repostitory
* Repostitory 可以分为两个大类
* 官方的仓库，格式 仓库名:标签，如 nginx:latest
* 其它类，如用户仓库、第三方仓库、私有仓库等等
* 一个镜像可以有多个标签
* Index

提供用户认证、镜像检索功能

**2. docker-registry 安装使用**

docker-registry 是 Docker 官方提供的镜像仓库管理软件，可以提供 docker pull 和 docker push 等操作，但是没有web 界面，镜像管理非常不方便，一般不作为私有仓库使用。

[root@centos-82 ~]# yum install -y docker-registry ## 实际安装软件包是 docker distribution

[root@centos-82 ~]# rpm -ql docker-distribution

/etc/docker-distribution/registry/config.yml ## Configuration

/usr/bin/registry

/usr/lib/systemd/system/docker-distribution.service ## Service

......

[root@centos-82 ~]# vim /etc/docker-distribution/registry/config.yml  ## 配置端口和镜像存放目录

version: 0.1

log:

fields:

service: registry

storage:

cache:

layerinfo: inmemory

filesystem:

rootdirectory: /data/docker-registry

http:

addr: :5000

[root@centos-82 ~]# mkdir /data/docker-registry

[root@centos-82 ~]# systemctl start docker-distribution.service

[root@centos-82 ~]# ss -lntp | grep 5000

LISTEN 0 128 :::5000 :::\* users:(("registry",pid=9789,fd=3)

[root@centos-81 ~]# docker tag httpd:v0.2 hub.docker.reg:5000/httpd:v0.2 ## /etc/hosts 配置域名映射

[root@centos-81 ~]# docker push hub.docker.reg:5000/httpd:v0.2 ## docker 默认使用https协议

The push refers to a repository [hub.docker.reg:5000/httpd]

Get https://hub.docker.reg:5000/v1/\_ping: http: server gave HTTP response to HTTPS client

[root@centos-81 ~]# vim /etc/docker/daemon.json ## 添加http协议的registry

{

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

"insecure-registries":["hub.docker.reg:5000"]

}

[root@centos-81 ~]# systemctl restart docker

[root@centos-81 ~]# docker push hub.docker.reg:5000/httpd:v0.2

[root@centos-81 ~]# curl -s <http://192.168.1.82:5000/v2/httpd/tags/list> | python -mjson.tool ## 根据仓库名获取tags

{

"name": "httpd",

"tags": [

"v0.2",

"v0.1"

]

}

[root@centos-82 ~]# tree /data/docker-registry/

/data/docker-registry/

└── docker

└── registry

└── v2

├── blobs

│   └── sha256

│   ├── 3f

│   │   └── 3f22152f75b71784bfb6946248858ace5fad0e1a0db4a208f20607d88810e60c

│   │   └── data

│   ├── 6d

│   │   └── 6df52055d83b6b866af9d4907574421aedfdf3f8c27c1ba45e7d1c9236000f80

│   │   └── data

│   └── 95

│   └── 95090e1ca932d43e6791e981e310d67d50af9ef7fe4ec1fda859064e6533a21c

│   └── data

└── repositories

└── httpd

├── \_layers

│   └── sha256

│   ├── 3f22152f75b71784bfb6946248858ace5fad0e1a0db4a208f20607d88810e60c

│   │   └── link

│   └── 6df52055d83b6b866af9d4907574421aedfdf3f8c27c1ba45e7d1c9236000f80

│   └── link

├── \_manifests

│   ├── revisions

│   │   └── sha256

│   │   └── 95090e1ca932d43e6791e981e310d67d50af9ef7fe4ec1fda859064e6533a21c

│   │   └── link

│   └── tags

│   └── v0.2

│   ├── current

│   │   └── link

│   └── index

│   └── sha256

│   └── 95090e1ca932d43e6791e981e310d67d50af9ef7fe4ec1fda859064e6533a21c

│   └── link

└── \_uploads

[root@centos-81 ~]# docker image ls | grep hub

[root@centos-81 ~]# docker pull hub.docker.reg:5000/httpd:v0.1 ## 从私有registry拉取镜像

Trying to pull repository hub.docker.reg:5000/httpd ...

v0.1: Pulling from hub.docker.reg:5000/httpd

Digest: sha256:ec0aa9e4aff0ab1a001f87277324df24281a00513637f0c8045c59c2667f3eb8

Status: Downloaded newer image for hub.docker.reg:5000/httpd:v0.1

[root@centos-81 ~]# docker image ls | grep hub

hub.docker.reg:5000/httpd v0.1 562ec613ec3a 4 weeks ago 1.2 MB

**3. Harbor 安装使用**

**3.1. Harbor 介绍**

Docker-distribution 虽然能实现私有镜像仓库，但是管理复杂，且没有web界面，不支持搜索等。VMware基于docker distribution二次开发了Harbor，实现了web界面管理仓库，功能性极大的增强。Harbor 的特性：

* 基于多用户，多项目的访问控制
* 镜像复制
* 可以在WEB界面管理镜像仓库，并且支持中文
* 日志审计

**3.2. Harbor 安装**

**3.2.1. 依赖**

|  |  |  |
| --- | --- | --- |
| Resource | Capacity | Description |
| CPU | minimal 2 CPU | 4 CPU is preferred |
| Mem | minimal 4GB | 8GB is preferred |
| Disk | minimal 40GB | 160GB is preferred |

|  |  |  |
| --- | --- | --- |
| Software | Version | Description |
| Python | version 2.7 or higher | Note that you may have to install Python on Linux distributions (Gentoo, Arch) that do not come with a Python interpreter installed by default |
| Docker engine | version 1.10 or higher | For installation instructions, please refer to: <https://docs.docker.com/engine/installation/> |
| Docker Compose | version 1.6.0 or higher | For installation instructions, please refer to: <https://docs.docker.com/compose/install/> |
| Openssl | latest is preferred | Generate certificate and keys for Harbor |

|  |  |  |
| --- | --- | --- |
| Port | Protocol | Description |
| 443 | HTTPS | Harbor portal and core API will accept requests on this port for https protocol |
| 4443 | HTTPS | Connections to the Docker Content Trust service for Harbor, only needed when Notary is enabled |
| 80 | HTTP | Harbor portal and core API will accept requests on this port for http protocol |

**3.2.2. 安装和配置**

[root@centos-82 ~]# wget https://storage.googleapis.com/harbor-releases/release-1.7.0/harbor-offline-installer-v1.7.4.tgz

[root@centos-82 ~]# tar -xf harbor-offline-installer-v1.7.4.tgz -C /usr/local/ && cd /usr/local/harbor

[root@centos-82 harbor]# readlink -f docker-compose.yml ## 镜像配置文件，一般只需要配置路径即可

/usr/local/harbor/docker-compose.yml

[root@centos-82 harbor]# grep -Ev "^$|#" harbor.cfg

\_version = 1.7.0

hostname = hub.docker.reg ## docker推拉镜像时需要指定的服务端地址

ui\_url\_protocol = http ## 使用的协议

max\_job\_workers = 6 ## 最大工作进程数

customize\_crt = on ## ssl密钥相关，仅在https模式下生效

ssl\_cert = /data/cert/server.crt

ssl\_cert\_key = /data/cert/server.key

secretkey\_path = /data

admiral\_url = NA

log\_rotate\_count = 50 ## 日志切割配置

log\_rotate\_size = 200M

http\_proxy =

https\_proxy =

no\_proxy = 127.0.0.1,localhost,core,registry

email\_identity = ## Email配置项

email\_server = smtp.mydomain.com

email\_server\_port = 25

email\_username = sample\_admin@mydomain.com

email\_password = abc

email\_from = admin <sample\_admin@mydomain.com>

email\_ssl = false

email\_insecure = false

harbor\_admin\_password = Harbor12345 ## admin初始密码，可登陆到WEB页面修改

auth\_mode = db\_auth

ldap\_url = ldaps://ldap.mydomain.com ## 若关联ldap可以配置该项目

ldap\_basedn = ou=people,dc=mydomain,dc=com

ldap\_uid = uid

ldap\_scope = 2

ldap\_timeout = 5

ldap\_verify\_cert = true

ldap\_group\_basedn = ou=group,dc=mydomain,dc=com

ldap\_group\_filter = objectclass=group

ldap\_group\_gid = cn

ldap\_group\_scope = 2

self\_registration = on

token\_expiration = 30

project\_creation\_restriction = everyone

db\_host = postgresql

db\_password = Database.harbor.123 ## 数据库密码

db\_port = 5432

db\_user = postgres

redis\_host = redis

redis\_port = 6379

redis\_password =

redis\_db\_index = 1,2,3

clair\_db\_host = postgresql

clair\_db\_password = root123

clair\_db\_port = 5432

clair\_db\_username = postgres

clair\_db = postgres

clair\_updaters\_interval = 12

uaa\_endpoint = uaa.mydomain.org

uaa\_clientid = id

uaa\_clientsecret = secret

uaa\_verify\_cert = true

uaa\_ca\_cert = /path/to/ca.pem

registry\_storage\_provider\_name = filesystem

registry\_storage\_provider\_config =

registry\_custom\_ca\_bundle =

[root@centos-82 harbor]# ./install.sh ## 执行安装脚本，依赖(docker-compose，epel源)

[root@centos-82 harbor]# docker image ls  ## 涉及到的镜像

REPOSITORY TAG IMAGE ID CREATED SIZE

goharbor/chartmuseum-photon v0.8.1-v1.7.4 7e2272c02339 3 weeks ago 113MB

goharbor/harbor-migrator v1.7.4 968c31d07d2f 3 weeks ago 678MB

goharbor/redis-photon v1.7.4 611d1ead0a28 3 weeks ago 99.7MB

goharbor/clair-photon v2.0.7-v1.7.4 01090529ab14 3 weeks ago 165MB

goharbor/notary-server-photon v0.6.1-v1.7.4 737518b1b943 3 weeks ago 135MB

goharbor/notary-signer-photon v0.6.1-v1.7.4 495dc3326120 3 weeks ago 132MB

goharbor/harbor-registryctl v1.7.4 723aed7bbf8d 3 weeks ago 102MB

goharbor/registry-photon v2.6.2-v1.7.4 f4743bd7b0d9 3 weeks ago 86.7MB

goharbor/nginx-photon v1.7.4 dda34e6afafe 3 weeks ago 35.9MB

goharbor/harbor-log v1.7.4 bf4916eef530 3 weeks ago 81.4MB

goharbor/harbor-jobservice v1.7.4 1b6a0445ae9c 3 weeks ago 84.1MB

goharbor/harbor-core v1.7.4 e603b8750d26 3 weeks ago 95.6MB

goharbor/harbor-portal v1.7.4 2ca1d845cafa 3 weeks ago 40.6MB

goharbor/harbor-adminserver v1.7.4 5706c65d65dc 3 weeks ago 72.3MB

goharbor/harbor-db v1.7.4 08d163f732f3 3 weeks ago 136MB

[root@centos-82 harbor]# docker container ls

CONTAINER ID IMAGE COMMAND ......

d5c7eeca50fb goharbor/nginx-photon:v1.7.4 "nginx -g 'daemon of…" ......

6ec59a513f9a goharbor/harbor-portal:v1.7.4 "nginx -g 'daemon of…" ......

201267927c06 goharbor/harbor-jobservice:v1.7.4 "/harbor/start.sh" ......

3711d9548a8b goharbor/harbor-core:v1.7.4 "/harbor/start.sh" ......

1c092514ca13 goharbor/harbor-registryctl:v1.7.4 "/harbor/start.sh" ......

264c26f4d27c goharbor/harbor-adminserver:v1.7.4 "/harbor/start.sh" ......

527ceb179532 goharbor/registry-photon:v2.6.2-v1.7.4 "/entrypoint.sh /etc…" ......

127009add4f7 goharbor/harbor-db:v1.7.4 "/entrypoint.sh post…" ......

ca8dcdd36bc1 goharbor/redis-photon:v1.7.4 "docker-entrypoint.s…" ......

a850c81e3abb goharbor/harbor-log:v1.7.4 "/bin/sh -c /usr/loc…" ......

[root@centos-82 harbor]# netstat -lntp

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 127.0.0.1:1514 0.0.0.0:\* LISTEN 11409/docker-proxy

tcp6 0 0 :::80 :::\* LISTEN 12326/docker-proxy

tcp6 0 0 :::443 :::\* LISTEN 12312/docker-proxy

tcp6 0 0 :::4443 :::\* LISTEN 12299/docker-proxy

[root@centos-82 ~]# cat /etc/docker/daemon.json

{

"registry-mirrors":["http://hub-mirror.c.163.com","https://registry.docker-cn.com"],

"insecure-registries":["hub.docker.reg"]

}

[root@centos-82 ~]# cat /etc/hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.1.82 hub.docker.reg

**3.2.3. 启停Harbor**

* 停止Harbor

[root@centos-82 ~]# cd /usr/local/harbor/ ## change woker directory to harbor root directory.

[root@centos-82 harbor]# docker-compose stop

Stopping nginx ... done

Stopping harbor-portal ... done

Stopping harbor-jobservice ... done

Stopping harbor-core ... done

Stopping registryctl ... done

Stopping harbor-adminserver ... done

Stopping registry ... done

Stopping harbor-db ... done

Stopping redis ... done

Stopping harbor-log ... done

* 启动Harbor

[root@centos-82 ~]# cd /usr/local/harbor/

[root@centos-82 harbor]# docker-compose start

Starting log ... done

Starting registry ... done

Starting adminserver ... done

Starting core ... done

Starting redis ... done

Starting jobservice ... done

Starting registryctl ... done

Starting postgresql ... done

Starting portal ... done

Starting proxy ... done

**3.3. Harbor 使用**

图片包含 网站

描述已自动生成

图片包含 图形用户界面

描述已自动生成

[root@centos-81 ~]# docker tag nginx:latest hub.docker.reg/devops/nginx:latest

[root@centos-81 ~]# docker login -u admin hub.docker.reg           ## Login

Password:

Login Succeeded

[root@centos-81 ~]# docker push hub.docker.reg/devops/nginx:latest ## Docker push

The push refers to a repository [hub.docker.reg/devops/nginx]

6b5e2ed60418: Pushed

92c15149e23b: Pushed

0a07e81f5da3: Pushed

latest: digest: sha256:5b49c8e2c890fbb0a35f6050ed3c5109c5bb47b9e774264f4f3aa85bb69e2033 size: 948

[root@centos-81 ~]# docker logout hub.docker.reg                   ## Logout

[root@centos-81 ~]# docker image pull hub.docker.reg/devops/nginx:latest  ## Docker pull

Trying to pull repository hub.docker.reg/devops/nginx ...

latest: Pulling from hub.docker.reg/devops/nginx

Digest: sha256:5b49c8e2c890fbb0a35f6050ed3c5109c5bb47b9e774264f4f3aa85bb69e2033

Status: Downloaded newer image for hub.docker.reg/devops/nginx:latest

[root@centos-81 ~]# docker image ls | grep hub.docker.reg

hub.docker.reg/devops/nginx latest f09fe80eb0e7 7 weeks ago 109 MB

# 07-Cgroups

## 1. CPU 资源限制

默认情况下，Docker Container 对宿主机的 CPU 资源的使用是没有限制的，这在大多数场景中是不合理的，因为容器异常时可能会占用抢占其它容器的 CPU 资源，因此需要对 CPU 资源进行限制。

|  |  |
| --- | --- |
| Options | Description |
| **-c|--cpu-share int** | 指定容器可使用的CPU核心时间片的权重，当多个容器争抢同一个核心时，按权重分配。默认值1024 |
| **--cpus=float** | 指定容器可以使用的CPU核心数(逻辑上)，可以是浮点数。 |
| **--cpuset-cpus** | 将容器和指定的CPU核心进行绑定 |
| --cpu-period | 设定单颗CPU核心的时间片，需要配合 --cpu-quota。范围1000μs~1000000μs。建议使用 --cpus 替代 |
| --cpu-quota | 指定容器能使用 --cpu-period 中的时间片数量。该参数不建议使用，尽可能使用 --cpus 替代 |

### 1.1. CPU share

[root@centos-82 ~]# docker container run --name t1 --cpuset-cpus 0,1 --rm -d polinux/stress stress -c 4 -t 120 # Default is 1024.

[root@centos-82 ~]# docker container run --name t2 --cpuset-cpus 0,1 --cpu-shares=2048 --rm -d polinux/stress stress -c 4 -t 120

[root@centos-82 ~]# docker stats

## cpu\_t1:cpu\_t2 = 1024:2048 = 1:2 ≈ 66:133

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

93d2f5fcf4da t2 133.52% 136KiB / 15.5GiB 0.00% 578B / 0B 0B / 0B 5

b15d2d2ae941 t1 66.40% 80KiB / 15.5GiB 0.00% 648B / 0B 0B / 0B 5

### 1.2. CPU period and CPU quota

[root@centos-82 ~]# docker container run --name t1 --rm -d --cpu-period 1000000 --cpu-quota 500000 polinux/stress:latest stress -c 4 -t 60 ## 50% CPU cycles

[root@centos-82 ~]# docker stats

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

806f2e493a1d t1 50.18% 84KiB / 15.5GiB 0.00% 578B / 0B 0B / 0B 5

[root@centos-82 ~]# docker container run --name t1 --rm -d --cpu-period 1000000 --cpu-quota 2000000 polinux/stress:latest stress -c 4 -t 60 ## 200% CPU cycles

[root@centos-82 ~]# docker stats

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

4a7f8bc4b3a8 t1 199.61% 204KiB / 15.5GiB 0.00% 578B / 0B 0B / 0B 5

### 1.3. CPUs

[root@centos-82 ~]# docker container run --name t1 --rm -d polinux/stress stress -c 4 -t 60  ## Unlimit cpus.

[root@centos-82 ~]# docker stats

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

779b4ba5c080 t1 400.26% 320KiB / 15.5GiB 0.00% 578B / 0B 0B / 0B 5

[root@centos-82 ~]# docker container run --name t1 --cpus=2 --rm -d polinux/stress stress -c 4 -t 120 ## limit cpus.

[root@centos-82 ~]# docker stats

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

baf520b31063 t1 200.22% 80KiB / 15.5GiB 0.00% 648B / 0B 0B / 0B 5

## 2. Mem 资源限制

需要注意的是，CPU 属于可压缩性的资源，即使 CPU 使用率达到 100% 也不会轻易宕机或者杀进程，只会导致其它容器无法有效的运行任务。但是内存属于不可压缩性资源，对于内存泄漏、垃圾回收不及时等容器，会占用大量的内存空间，而且系统无法释放这部分内存时，容易导致 OOM 甚至宕机。

|  |  |
| --- | --- |
| Options | Description |
| -m|--memory | 设置容器最大可用内存大小，不低于 4M |
| --memory-reservation | 设置可用内存大小的软限制 |
| --memory-swap | 设置 swap 可使用的情况，这个值并不是直接限制容器可用的 swap 大小 |
| --oom-kill-disable | 在使用了内存限制的容器上才能使用，用于避免容器被 OOM |
| --memory-swappiness | 设置 swappiness 的值，默认和操作系统中的一致 |
| --kernel-memory | 最大可用的内核内存的大小，不低于 4M |

--memory 和 --memory-swap 直接的关系

|  |  |  |
| --- | --- | --- |
| Options | Memory Size | Swap Size |
| --memory M --memory-swap S | M | S-M |
| --memory M --memory-swap M | M | 0 |
| --memory M --memory-swap 0 | M | Unset(2M) |
| --memory M --memory-swap -1 | M | Unlimit |
| --memory M | M | Unset(2M) |

[root@centos-82 ~]# docker container run --name t1 --rm --memory 1024m --memory-swap 1024m  polinux/stress:latest stress -m 5 -t 60 ## 禁用swap

[root@centos-82 ~]# dmesg  | grep -i 'out of memory'

[ 2480.558688] Memory cgroup out of memory: Kill process 4921 (stress) score 239 or sacrifice child

[root@centos-82 ~]# docker container run --name t1 --rm --memory 1024m --memory-swap 2048m  polinux/stress:latest stress -m 5 -t 60

[root@centos-82 ~]# docker stats ## Limit total memory 2048m and swap size is 1024m.

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

598431c16b9d t1 328.43% 1024MiB / 1GiB 99.98% 578B / 0B 1.16GB / 2.32GB 6

[root@centos-82 ~]# docker container exec t1 free -m ## The memeory info is host system.

total used free shared buffers cached

Mem: 15869 1670 14198 0 2 279

-/+ buffers/cache: 1388 14480

Swap: 4095 482 3613

## 3. I/O 资源限制

一般情况下，不会对容器的 I/O 读写进行限制，I/O 读写限制会造成严重的性能问题。

|  |  |
| --- | --- |
| Options | Description |
| --device-write-bps | Limit write rate (bytes per second) to a device.The size data of write to device in one second. |
| --device-write-iops | Limit write rate (IO per second) to a device.The counts of write to device in one second. |
| --device-read-bps | Limit read rate (bytes per second) to a device.The size data of read to device in one second. |
| --device-read-iops | Limit read rate (IO per second) to a device.The counts of read to device in one second. |

### 3.1. 限制 bps

[root@centos-76 ~]# docker run -dit --name io-2 -v /var/www/html/:/var/www/html --device /dev/sda:/dev/sda --device-write-bps /dev/sda:100kb centos:gtapp /bin/bash

[root@centos-76 ~]# docker exec e5199c2738 /bin/bash -c "time dd if=/dev/sda of=/var/www/html/io-2.file count=5 bs=1M oflag=direct,nonblock"

## limit 100 kb

5+0 records in

5+0 records out

5242880 bytes (5.2 MB) copied, 56.382 s, 93.0 kB/s # real speed is 93kb/s

real 0m56.385s

user 0m0.000s

sys 0m0.008s

### 3.2. 限制 ops

[root@centos-76 ~]# docker run -dit --name io-3 -v /var/www/html/:/var/www/html --device /dev/sda:/dev/sda --device-write-iops /dev/sda:10 centos:gtapp /bin/bash

[root@centos-76 ~]# docker exec 9946e38b6c171 /bin/bash -c "time dd if=/dev/sda of=/var/www/html/io-3.file count=1000 bs=4k oflag=direct,nonblock"

## limit 10 ops/s --> 10 \* 4k = 40 kb/s

1000+0 records in

1000+0 records out

4096000 bytes (4.1 MB) copied, 100.364 s, 40.8 kB/s

real 1m40.367s

user 0m0.004s

sys 0m0.420s