

第1章 Docker容器介绍

1.Docker是什么

- 1

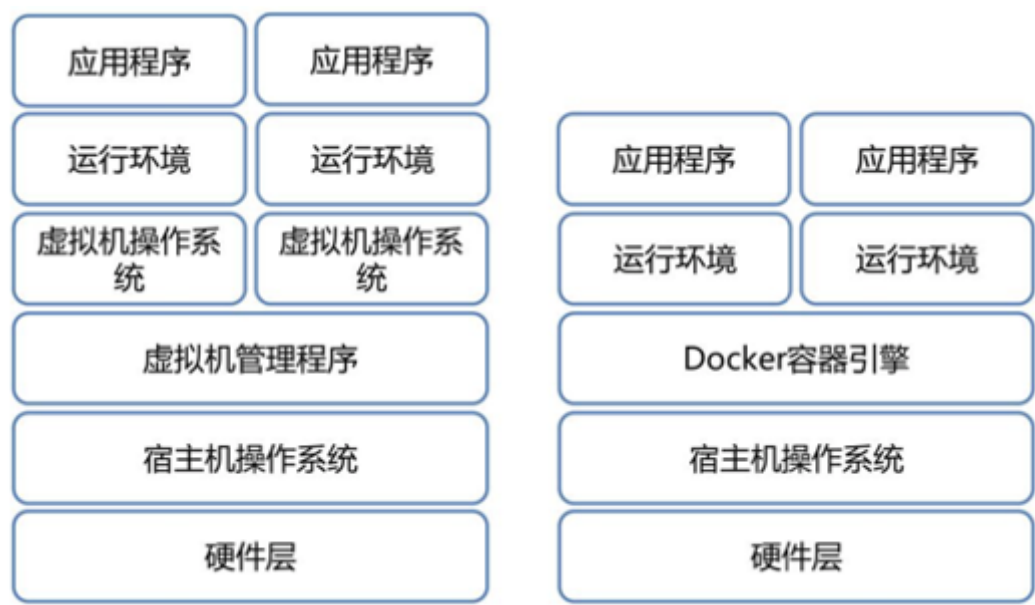
Docker是Docker.Inc 公司开源的一个基于LXC技术之上构建的Container容器引擎,源代码托管在GitHub 上,基于Go语言并遵从Apache2.0协议开源。
- 2

Docker是通过内核虚拟化技术（namespaces及cgroups等）来提供容器的资源隔离与安全保障等。
- 3

由于Docker通过操作系统层的虚拟化实现隔离，所以Docker容器在运行时，不需要类似虚拟机（VM）额外的操作系统开销，提高资源利用率。

2.容器与虚拟机对比

传统虚拟化和Docker分层对比:



VM虚拟化和Docker特性对比

特性	Docker	VM
启动速度	秒级	分钟级
硬盘使用	一般为MB	一般为GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个
隔离性	安全隔离	完全隔离

3.namespace和cgroup

namespace资源隔离

kernel提供了namespace的机制用来隔离相关资源。namespace设计之初就是为了实现轻量级的系统资源隔离。

可以让容器中的进程仿佛置身于一个独立的系统环境中。

namespace	系统调用参数	隔离内容
UTC	CLONE_NEWUTS	主机名和域名
IPC	CLONE_NEWIPC	信号量、消息队列和共享内存
PID	CLONE_NEWPID	进程编号
Network	CLONE_NEWNET	网络设备、网络栈、端口等
Mount	CLONE_NEWNS	文件系统
User	CLONE_NEWUSER	用户和用户组

cgroups资源限制

- 1 | cgroup的作用主要是用来控制资源的使用，比如限制CPU内存和磁盘的使用等

cgroups的四大作用：

- 1 | 资源限制： 比如设定任务内存使用的上限。

2 | 优先级分配： 比如给任务分配CPU的时间片数量和磁盘IO的带宽大小来控制任务运行的优先级。

3 | 资源统计： 比如统计CPU的使用时长、内存用量等。这个功能非常适用于计费。

4 | 任务控制： cgroups可以对任务执行挂起、恢复等操作。

4.docker的三个重要概念

Image(镜像):

- 1 | 那么镜像到底是什么呢？ Docker 镜像可以看作是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。

Container(容器):

- 1 | 容器(Container)的定义和镜像(Image)几乎一模一样，也是一堆层的统一视角，唯一区别在于容器的最上面那一层是可读可写的。

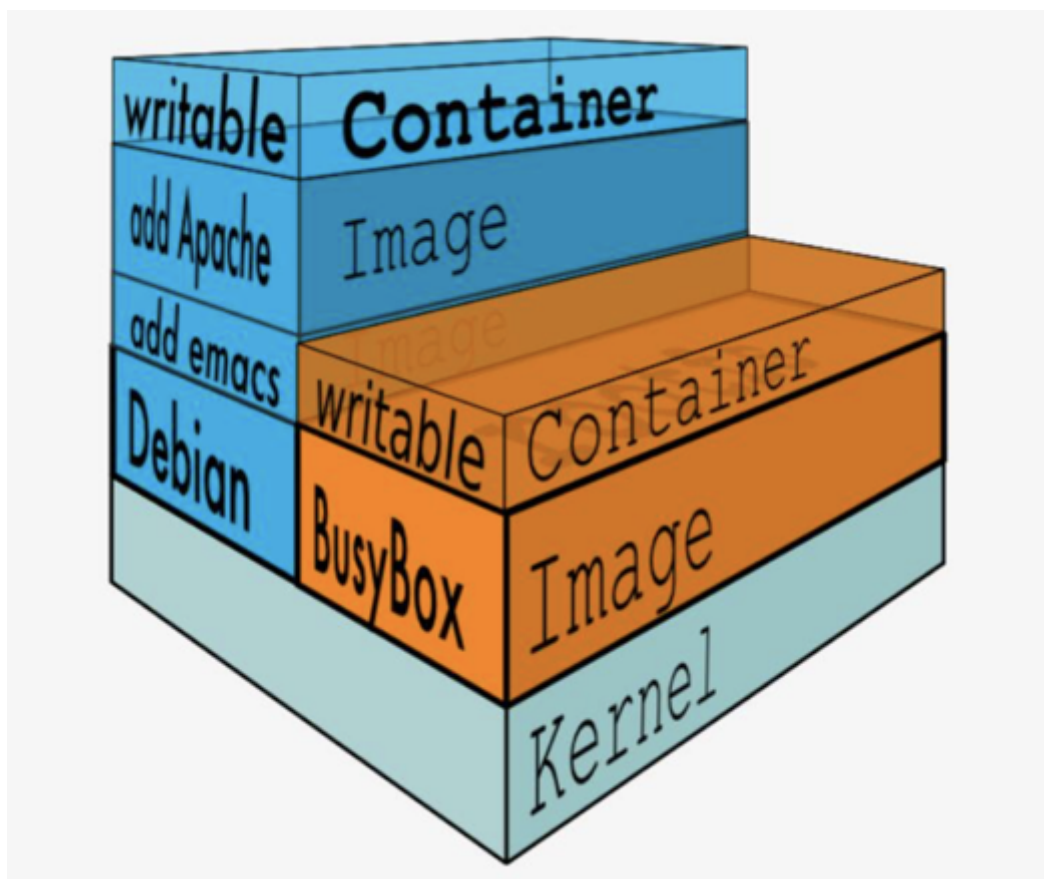
Repository(仓库):

- 1 | 镜像仓库是 Docker 用来集中存放镜像文件的地方，类似于我们之前常用的代码仓库。

2 | 通常，一个仓库会包含同一个软件不同版本的镜像，而标签就常用于对应该软件的各个版本。

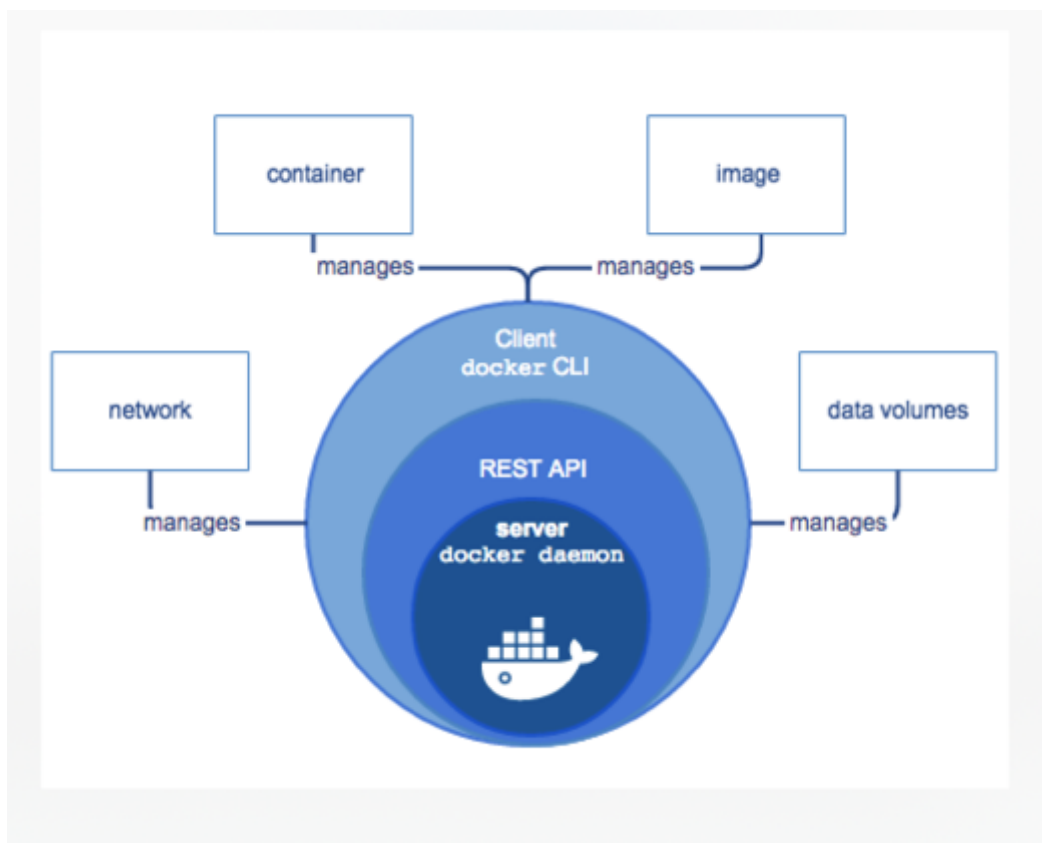
3 | 我们可以通过<仓库名>:<标签>的格式来指定具体是这个软件哪个版本的镜像。如果不给出标签，将以Latest 作为默认标签。

镜像和容器图解:



5.docker的组成部分

- 1 Docker是传统的CS架构分为docker client和docker server
- 2 Docker 客户端是 Docker 用户与 Docker 交互的主要方式。
- 3 当您使用 Docker 命令行运行命令时，Docker 客户端将这些命令发送给服务器端，服务端将执行这些命令。
- 4 Docker 命令使用 Docker API 。
- 5 Docker 客户端可以与多个服务端进行通信。



第2章 Docker安装部署

1.国内源安装docker-ce

这里我们使用清华源：

1 | <https://mirrors.tuna.tsinghua.edu.cn/help/docker-ce/>

操作步骤：

```
1 | yum remove docker docker-common docker-selinux docker-engine
2 | yum install -y yum-utils device-mapper-persistent-data lvm2
3 | wget -O /etc/yum.repos.d/docker-ce.repo
   https://download.docker.com/linux/centos/docker-ce.repo
4 | sed -i 's+download.docker.com+mirrors.tuna.tsinghua.edu.cn/docker-ce+'
   /etc/yum.repos.d/docker-ce.repo
5 | yum makecache fast
6 | yum install docker-ce
7 | systemctl start docker
```

2.国内远镜像加速配置

加速地址：

1 | <https://cr.console.aliyun.com/cn-hangzhou/instances/mirrors>

配置命令：

```
1 | mkdir -p /etc/docker
2 | tee /etc/docker/daemon.json <<- 'EOF'
3 | {
4 |   "registry-mirrors": ["https://ig2l319y.mirror.aliyuncs.com"]
5 | }
6 | EOF
7 | systemctl daemon-reload
8 | systemctl restart docker
```

3.运行第一个容器

运行一个Hello world

```
1 | docker run alpine /bin/echo "Hello world"
```

第3章 Docker镜像和容器管理

1.镜像相关命令

1.1 搜索镜像

选择镜像建议：

1. 优先选择官方的
2. 选择星星多的

搜索命令：

```
1 | docker search centos
```

使用curl命令获取镜像版本号：

```
1 | yum install jq
2 | curl -s https://registry.hub.docker.com/v1/repositories/centos/tags|jq
```

1.2 获取镜像

```
1 | docker pull centos
2 | docker pull busybox
3 | docker pull busybox:1.29
```

1.3 查看镜像

```
1 | docker images
```

1.4 删除镜像

```
1 | docker rmi centos
```

1.5 导出镜像

```
1 | docker save -o centos.tar centos:latest
```

1.6 导入镜像

```
1 | docker load < centos.tar
```

2.容器相关命令

2.1 启动容器

```
1 | docker run -d -p 80:80 nginx
2 | docker run --name mydocker -t -i centos /bin/bash
3 |
4 | #参数解释
5 | run 运行容器
6 | --name 指定容器的名称，但是不能和已经存在的重复
7 | -d 后台运行容器，并返回容器ID
8 | -p 80:80 端口映射 宿主机端口:容器端口
9 | -i 以交互模式运行容器，通常与 -t 同时使用；
10 | -t 为容器重新分配一个伪输入终端，通常与 -i 同时使用
```

2.2 停止容器

```
1 | docker stop
```

2.3 查看容器

```
1 | docker ps
2 | docker ps -a
3 | docker ps -q
4 | docker ps -aq
5 | docker stats 容器ID
6 | docker rm $(docker ps -q -f 'STATUS=exited')
```

2.4 进入容器

```
1 | docker exec 会分配一个新的终端tty
2 | docker exec -it 容器ID /bin/bash
3 | docker attach会使用同一个终端
4 | docker attach 容器ID
5
6 | #参数解释
7 | exec 在运行的容器中执行命令
8 | -i 保持终端打开
9 | -t 分配一个终端
```

2.5 删除容器

删除单个容器

```
1 | docker rm 容器ID
```

批量删除容器

```
1 | docker stop $(docker ps -q)
2 | docker rm $(docker ps -aq)
```

第4章 Docker网络管理

1.随机映射端口

```
1 | docker run -P
```

2.指定映射端口

-p 宿主机端口:容器端口

```
1 | -p 80:80 -p 443:443
2 | -p 宿主机IP:宿主机端口:容器端口
```

如果想多个容器使用8080端口，可以通过添加多个IP地址实现

```
1 | ifconfig eth0:1 10.0.0.13 up
2 | docker run -d -p 10.0.0.11:8080:80 nginx:latest
3 | docker run -d -p 10.0.0.13:8080:80 nginx:latest
```

进入容器里修改站点目录，然后访问测试

```
1 | docker exec -it bdb2a4e7e24d /bin/bash
2 | echo "web01" > /usr/share/nginx/html/index.html
3 | docker exec -it 31c1de138dda /bin/bash
4 | echo "web02" > /usr/share/nginx/html/index.html
```

访问测试：

```
1 [root@docker-11 ~]# curl 10.0.0.11:8080
2 web02
3 [root@docker-11 ~]# curl 10.0.0.13:8080
4 web01
```

第5章 Docker数据目录管理

1.映射容器目录

```
1 -v 宿主机目录:容器内目录
```

1.1 创建游戏代码目录

```
1 mkdir /data/xiaoniao -p
2 cd /data/
3 unzip xiaoniaofeifei.zip -d xiaoniao /
```

1.2 创建容器并映射数据目录

```
1 docker run -d -p 80:80 -v /data/xiaoniao:/usr/share/nginx/html nginx:latest
2 docker ps
```

1.3 访问游戏页面

```
1 10.0.0.11
```

2.实验-访问不同端口展示不同页面

需求:

- 1 访问8080端口, 展示xiaoniao首页
- 2 访问8090端口, 展示shenjingmao的首页

2.1 准备nginx配置文件

```
1 [root@docker-11 ~]# cat /data/game.conf
2 server {
3     listen      8080;
4     server_name localhost;
5     location / {
6         root    /opt/game/;
7         index   index.html index.htm;
8     }
9 }
10
11 server {
12     listen      8090;
13     server_name localhost;
14     location / {
15         root    /opt/xiaoniao/;
```



```
16     index index.html index.htm;
17 }
18 }
```

2.2 上传代码目录

```
1 [root@docker-11 /data]# ll
2 总用量 18896
3 drwxr-xr-x 5 root root      73 9月  7 23:03 game
4 -rw-r--r-- 1 root root      309 9月  7 22:57 game.conf
5 -rw-r--r-- 1 root root 19248295 8月 28 09:48 html5.zip
6 drwxr-xr-x 3 root root      92 9月  7 22:15 xiaoniao
7 -rw-r--r-- 1 root root   91014 9月  7 22:11 xiaoniaofeifei.zip
```

2.3 创建容器并挂载

需要挂载的内容：

```
1 1.nginx配置文件
2 2.游戏目录
```

创建容器命令：

```
1 docker run
2 -p 8080:8080 \
3 -p 8090:8090 \
4 -v /data/game:/opt/game \
5 -v /data/xiaoniao:/opt/xiaoniao \
6 -v /data/game.conf:/etc/nginx/conf.d/game.conf \
7 -d nginx:latest
```

2.4 访问测试

```
1 10.0.0.11:8080
2 10.0.0.11:8090
```

第6章 Docker镜像手动构建

1.手动制作游戏镜像

```
1 下面我们基于centos容器制作一个新镜像，并安装nginx服务
```

1.1 启动一个容器并安装nginx

```
1 [root@docker-11 ~]# docker run -it centos /bin/bash
2 [root@0ede2760ba65 /]# yum install wget install openssh-clients -y
3 [root@0ede2760ba65 /]# wget -O /etc/yum.repos.d/CentOS-Base.repo
  http://mirrors.aliyun.com/repo/Centos-7.repo
4 [root@0ede2760ba65 /]# wget -O /etc/yum.repos.d/epel.repo
  http://mirrors.aliyun.com/repo/epel-7.repo
```

```
5 [root@Oede2760ba65 /]# sed -i -e '/mirrors.cloud.aliyuncs.com/d' -e
'/mirrors.aliyuncs.com/d' /etc/yum.repos.d/CentOS-Base.repo
6 [root@Oede2760ba65 /]# cat /etc/yum.repos.d/nginx.repo
7 [nginx-stable]
8 name=nginx stable repo
9 baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
10 gpgcheck=1
11 enabled=1
12 gpgkey=https://nginx.org/keys/nginx_signing.key
13 [nginx-mainline]
14 name=nginx mainline repo
15 baseurl=http://nginx.org/packages/mainline/centos/$releasever/$basearch/
16 gpgcheck=1
17 enabled=0
18 gpgkey=https://nginx.org/keys/nginx_signing.key
19 [root@Oede2760ba65 /]# yum makecache fast
20 [root@Oede2760ba65 /]# yum install nginx -y
```

1.2 上传代码目录并配置nginx配置文件

```
1 [root@Oede2760ba65 /]# scp -r 10.0.0.11:/data/* /opt/
2 [root@Oede2760ba65 /]# ll /opt/
3 total 18896
4 drwxr-xr-x 5 root root      73 Sep  7 16:02 game
5 -rw-r--r-- 1 root root    303 Sep  7 16:02 game.conf
6 -rw-r--r-- 1 root root 19248295 Sep  7 16:02 html5.zip
7 drwxr-xr-x 3 root root     92 Sep  7 16:02 xiaoniao
8 -rw-r--r-- 1 root root  91014 Sep  7 16:02 xiaoniaofeifei.zip
9 [root@Oede2760ba65 /]# cp /opt/game.conf /etc/nginx/conf.d/
```

1.3 将容器提交为新的镜像

```
1 [root@docker-11 ~]# docker ps -aq
2 0ede2760ba65
3 [root@docker-11 ~]# docker commit 0ede2760ba65 game:v1
4 sha256:a61d28fbfe27ebe36d4b73825b55e5f94097083273ab56dccce0453ce2bd6d38
```

1.4 测试镜像功能是否可用

```
1 [root@docker-11 ~]# docker run -d -p 8080:8080 -p 8090:8090 game:v1 nginx -g
'daemon off;'
2 f58f209d4761c4bdd9bb164c0050a94a3273b1ee0e57eafe29e48b1517c72950
```

1.5 将新镜像导出

```
1 docker save -o game_v1.tar game:v1
```

2.手动制作云盘镜像

2.1 创建容器

```
1 docker run -d -p 80:80 --name clould game:v1 nginx -g 'daemon off;'
```

2.2 进入容器安装php并求改运行用户

```
1 [root@d0c987bcefa2 /]# yum install php-fpm -y
2 [root@d0c987bcefa2 /]# php-fpm -v
3 PHP 5.4.16 (fpm-fcgi) (built: Oct 30 2018 19:32:20)
4 Copyright (c) 1997-2013 The PHP Group
5 Zend Engine v2.4.0, Copyright (c) 1998-2013 Zend Technologies
6 [root@d0c987bcefa2 /]# sed -i '/^user/c user = nginx' /etc/php-fpm.d/www.conf
7 [root@d0c987bcefa2 /]# sed -i '/^group/c group = nginx' /etc/php-fpm.d/www.conf
8 [root@d0c987bcefa2 /]# sed -i '/daemonize/s#no#yes#g' /etc/php-fpm.conf
9 [root@d0c987bcefa2 /]# php-fpm -c /etc/php.ini -y /etc/php-fpm.conf
10 [root@d0c987bcefa2 /]# php-fpm -c /etc/php.ini -y /etc/php-fpm.conf

11 [root@d0c987bcefa2 /]# ps -ef|grep php
12 root          77      0  0 21:43 ?           00:00:00 php-fpm: master process
    (/etc/php-fpm.conf)
13 nginx         78      77  0 21:43 ?           00:00:00 php-fpm: pool www
14 nginx         79      77  0 21:43 ?           00:00:00 php-fpm: pool www
15 nginx         80      77  0 21:43 ?           00:00:00 php-fpm: pool www
16 nginx         81      77  0 21:43 ?           00:00:00 php-fpm: pool www
17 nginx         82      77  0 21:43 ?           00:00:00 php-fpm: pool www
```

2.3 配置nginx

```
1 [root@d0c987bcefa2 /]# cat /etc/nginx/conf.d/cloud.conf
2 server {
3     listen 80;
4     server_name localhost;
5     root /code;
6     index index.php index.html;
7
8     location ~ /\.php$ {
9         root /code;
10        fastcgi_pass 127.0.0.1:9000;
11        fastcgi_index index.php;
12        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
13        include fastcgi_params;
14    }
15 }
16 [root@d0c987bcefa2 /]# nginx -t
17 nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
18 nginx: configuration file /etc/nginx/nginx.conf test is successful
19 [root@d0c987bcefa2 /]# nginx -s reload
```

2.4 下载代码目录

```
1 [root@d0c987bcefa2 /]# mkdir /code
2 [root@d0c987bcefa2 /]# cd /code/
3 [root@d0c987bcefa2 code]# scp -r 10.0.0.11:/data/kod/* /code/
4 [root@d0c987bcefa2 code]# ls
5 ChangeLog.md README.md app config data index.php plugins static
6 [root@d0c987bcefa2 code]# chown -R nginx:nginx /code/
```

2.5 测试

```
1 [root@d0c987bcefa2 code]# curl -I 127.0.0.1
2 HTTP/1.1 302 Moved Temporarily
3 Server: nginx/1.16.1
4 Date: Sat, 07 Sep 2019 21:53:17 GMT
5 Content-Type: text/html; charset=utf-8
6 Connection: keep-alive
7 X-Powered-By: PHP/5.4.16
8 Set-Cookie: KOD_SESSION_ID_9d6d9=1jq63o0tmcscon6eb3gdpqscf4; path=/
9 Set-Cookie: KOD_SESSION_ID_9d6d9=1jq63o0tmcscon6eb3gdpqscf4; path=/
10 Set-Cookie: KOD_SESSION_ID_9d6d9=1jq63o0tmcscon6eb3gdpqscf4; path=/
11 Set-Cookie: KOD_SESSION_SSO=bboh1p0h1uc50tfibrg67dnra7; path=/
12 Expires: Thu, 19 Nov 1981 08:52:00 GMT
13 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
14 Pragma: no-cache
15 Set-Cookie: KOD_SESSION_ID_9d6d9=deleted; expires=Thu, 01-Jan-1970 00:00:01
    GMT; path=/
16 Set-Cookie: kod_name=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT
17 Set-Cookie: kodToken=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT
18 Set-Cookie: X-CSRF-TOKEN=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT
19 Location: ./index.php?user/login
```

2.6 提交新的镜像

```
1 [root@docker-11 ~]# docker commit d0c987bcefa2 kod:v1
2 sha256:169df6e8db11bd044e3e05237f2947783f9cc7a65b643dc9206ecf05fdc4a3ea
```

2.7 编写启动脚本并提交新镜像

```
1 [root@docker-11 ~]# docker exec -it c14835183fb5 /bin/bash
2 [root@c14835183fb5 /]# cat init.sh
3 #!/bin/bash
4 php-fpm -c /etc/php.ini -y /etc/php-fpm.conf
5 nginx -g 'daemon off;'
6 [root@c14835183fb5 /]# chmod +x init.sh
7 [root@docker-11 ~]# docker commit c14835183fb5 kod:v2
8 sha256:c05ebdf400aa7f7a27aa857df0d9c75c42943db89abca66f79101771db8e9585
```

2.8 启动测试

```
1 [root@docker-11 ~]# docker stop $(docker ps -qa)
2 [root@docker-11 ~]# docker run -d -p 80:80 kod:v2 /bin/bash /init.sh
3 dccf4aea5471713872e4fefaca45f7fac3bffec8f5f602570863ed14231dea1a
4 [root@docker-11 ~]# docker ps
5 CONTAINER ID        IMAGE               COMMAND             CREATED
   STATUS             PORTS
   NAMES
6 dccf4aea5471        kod:v2             "/bin/bash /init.sh" 36 seconds
   ago               Up 35 seconds      0.0.0.0:80->80/tcp
   magical_napier
```

2.9 添加GD库

```
1 此时打开页面提示缺少GD库，进入容器内安装php支持插件
2 [root@dccf4aea5471 /]yum install php-mbstring php-gd -y
3 然后重启容器
4 [root@docker-11 ~]# docker restart dccf4aea5471
```

2.10 访问测试没问题后提交新镜像

```
1 [root@docker-11 ~]# docker commit dccf4aea5471 kod:v2
2 sha256:23051ce545a2eb6bb50bb2307bd9cfbaf6139e52f205a4126fb1d8d974c417f4
```

第7章 Dockerfile自动构建Docker镜像

1.Dockerfile介绍

- 1 通过前面的练习我们已经掌握了手动制作镜像的方法，但是这种方法命令繁多，而且不能自动化，操作起来比较复杂。
- 2 实际上在企业中我们会使用Dockerfile来自动化构建镜像。
- 3 Dockerfile是一种可以被docker解释并执行的脚本，拥有自己固定的指令。
- 4 有了这个利器之后我们运维就可以解放双手了，只要编写好脚本就可以按照我们期望的结果构建相应的镜像。
- 5 如果需要更新，只需要修改Dockerfile里很少的代码就可以重复的构建。

2.Dockerfile操作命令说明

- 1 Docker通过对于在Dockerfile中的一系列指令的顺序解析实现自动的image的构建
- 2 通过使用build命令，根据Dockerfile的描述来构建镜像
- 3 通过源代码路径的方式
- 4 通过标准输入流的方式
- 5 Dockerfile指令：
- 6 只支持Docker自己定义的一套指令，不支持自定义
- 7 大小写不敏感，但是建议全部使用大写
- 8 根据Dockerfile的内容顺序执行
- 9 FROM：
- 10 FROM {base镜像}
- 11 必须放在Dockerfile的第一行，表示从哪个base image开始构建
- 12 MAINTAINER：
- 13 可选的，用来标识image作者的地方

14 RUN:

15 每一个RUN指令都会是在一个新的container里面运行, 并提交为一个image作为下一个RUN的 base

16 一个Dockerfile中可以包含多个RUN, 按定义顺序执行

17 RUN支持两种运行方式:

18 RUN <cmd> 这个会当作/bin/sh -c "cmd" 运行

19 RUN ["executable", "arg1", ...], Docker把他当作json的顺序来解析, 因此必须使用双引号, 而且executable需要是完整路径

20 RUN 都是启动一个容器、执行命令、然后提交存储层文件变更。第一层 RUN command1 的执行仅仅是当前进程, 一个内存上的变化而已, 其结果不会造成任何文件。而到第二层的时候, 启动的是一个全新的容器, 跟第一层的容器更完全没关系, 自然不可能继承前一层构建过程中的内存变化。而如果需要将两条命令或者多条命令联合起来执行需要加上&&。如: cd /usr/local/src && wget xxxxxxxx

21 CMD:

22 CMD的作用是作为执行container时候的默认行为 (容器默认的启动命令)

23 当运行container的时候声明了command, 则不再用image中的CMD默认所定义的命令

24 一个Dockerfile中只能有一个有效的CMD, 当定义多个CMD的时候, 只有最后一个才会起作用

25 CMD定义的三种方式:

26 CMD <cmd> 这个会当作/bin/sh -c "cmd"来执行

27 CMD ["executable","arg1",....]

28 CMD ["arg1","arg2"], 这个时候CMD作为ENTRYPOINT的参数

29 EXPOSE 声明端口

30 格式为 EXPOSE <端口1> [<端口2>...].

31 EXPOSE 指令是声明运行时容器提供服务端口, 这只是一个声明, 在运行时并不会因为这个声明应用就会开启这个端口的服务。在 Dockerfile 中写入这样的声明有两个好处, 一个是帮助镜像使用者理解这个镜像服务的守护端口, 以方便配置映射; 另一个用处则是在运行时使用随机端口映射时, 也就是 docker run -P 时, 会自动随机映射 EXPOSE 的端口。

32 ENTRYPOINT:

33 entrypoint的作用是, 把整个container变成了一个可执行的文件, 这样不能够通过替换CMD的方法来改变创建container的方式。但是可以通过参数传递的方法影响到container内部

34 每个Dockerfile只能包含一个entrypoint, 多个entrypoint只有最后一个有效

35 当定义了entrypoint以后, CMD只能作为参数进行传递

36 entrypoint定义方式:

37 entrypoint ["executable","arg1","arg2"], 这种定义方式下, CMD可以通过json的方式来定义entrypoint的参数, 可以通过在运行container的时候通过指定command的方式传递参数

38 entrypoint <cmd>, 当作/bin/bash -c "cmd"运行命令

39 ADD & COPY:

40 当在源代码构建的方式下, 可以通过ADD和COPY的方式, 把host上的文件或者目录复制到image中

41 ADD和COPY的源必须在context路径下

42 当src为网络URL的情况下, ADD指令可以把它下载到dest的指定位置, 这个在任何build的方式下都可以work

43 ADD相对COPY还有一个多的功能, 能够进行自动解压压缩包

44 ENV:

45 ENV key value

46 用来设置环境变量, 后续的RUN可以使用它所创建的环境变量

47 当创建基于该镜像的container的时候, 会自动拥有设置的环境变量

48 WORKDIR:

49 用来指定当前工作目录 (或者称为当前目录)

50 当使用相对目录的情况下, 采用上一个WORKDIR指定的目录作为基准

51 USER:

52 指定UID或者username, 来决定运行RUN指令的用户

53 ONBUILD:

54 ONBUILD作为一个trigger的标记, 可以用来trigger任何Dockerfile中的指令

55 可以定义多个ONBUILD指令

56 当下一个镜像B使用镜像A作为base的时候, 在FROM A指令前, 会先按照顺序执行在构建A时候定义的ONBUILD指令

57 ONBUILD <DOCKERFILE 指令> <content>

```
58 VOLUME:
59     用来创建一个在image之外的mount point, 用来在多个container之间实现数据共享
60     运行使用json array的方式定义多个volume
61     VOLUME ["/var/data1", "/var/data2"]
62     或者plain text的情况下定义多个VOLUME指令
```

4.Dockerfile小试身手

构建思路:

1. 先不要着急写Dockerfile, 首先手动进入容器, 然后正常执行安装步骤。并确保运行正常
2. 收集好安装步骤的命令以及正确的配置文件
3. 将收集的配置文件按照一定规范保存在相应的目录下
4. 根据收集到的安装步骤编写Dockerfile
5. 构建镜像并启动测试

小项目: 使用Dockerfile构建Centos7+Nginx镜像

```
1  #1.创建目录
2  [root@docker-11 ~]# mkdir dockerfile
3  [root@docker-11 ~]# cd dockerfile/
4  [root@docker-11 ~/dockerfile]# mkdir nginx_base
5  [root@docker-11 ~/dockerfile]# cd nginx_base/
6
7  #2.准备文件
8  [root@docker-11 ~/dockerfile/nginx_base]# cat > local.repo << 'EOF'
9  [local]
10 name=local
11 enable=1
12 gpgcheck=0
13 baseurl=http://10.0.0.100
14 EOF
15 [root@docker-11 ~/dockerfile/nginx_base]# ll
16 total 8
17 -rw-r--r-- 1 root root 292 Jul 22 15:01 Dockerfile
18 -rw-r--r-- 1 root root 65 Jul 22 15:47 local.repo
19
20 #3.编写Dockerfile
21 [root@docker-11 ~/dockerfile/nginx_base]# cat > Dockerfile << 'EOF'
22 FROM centos:7
23 RUN rm -rf /etc/yum.repos.d/*
24 ADD local.repo /etc/yum.repos.d/local.repo
25 RUN yum makecache fast \
26     yum install nginx -y \
27     yum clean all
28 EXPOSE 80
29 CMD ["nginx","-g","daemon off;"]
30 EOF
31
32 #4.构建镜像
33 [root@docker-11 ~/dockerfile/nginx_base]# cat build.sh
34 #!/bin/bash
35 docker build -t centos7_nginx:1.20 .
36 [root@docker-11 ~/dockerfile/nginx_base]# bash build.sh
37
38 #5.启动测试
```

```
39 [root@docker-11 ~/dockerfile/nginx_base]# docker run -d -p 10.0.0.11:80:80
centos7_nginx:1.20
40 [root@docker-11 ~/dockerfile/nginx_base]# curl -I 10.0.0.11
```

5.Dockerfile分段构建的最佳实践

避免安装不必要的软件包

- 1 只安装容器必须的软件，不是必须有的不要安装，比如一些工具类的命令，`wget net-tools`等

安装完成后清除缓存

- 1 安装完成后我们可以把不用的压缩包以及软件缓存等文件删除掉，以减少镜像的体积

一个容器内不要跑太多的应用

- 1 Docker倡导一个容器应该只关注一件事，应该保证一个容器只有一个进程。
- 2 但是这也不是硬性要求，比如nginx+php+mysql的应用，那么我们也可以将nginx+php放在一个容器里，mysql单独放一个容器。
- 3 但是我们还是希望一个容器专注于一件事，尽量保持干净和模块化。

最小化镜像层数

- 1 Dockerfile里每条RUN, COPY, ADD指令都会创建指令层，所以我们可以将多条RUN命令尽可能的合并成一个RUN指令，这样就减少了构建镜像的层数。

增加可读性

- 1 我们刚才说了可以讲多个RUN指令合并成一个RUN指令，但是这样做可能会导致RUN指令很长很长，那么我们可以像shell命令那样使用\来换行，增加可读性，例如：
- 2 RUN yum install git \
- 3 wget \
- 4 net-tools \
- 5 tree

使用supervisor控制多个进程

```
1 #1.安装软件
2 yum -y install supervisor
3
4 #2.编写进程配置文件
5 [root@b7757c17bf36 ~]# cat /etc/supervisord.d/nginx_php.ini
6 [program:nginx]
7 command=nginx -g 'daemon off;'
8 autostart=true
9 autorestart=true
10 startsecs = 5
11 redirect_stderr = true
12 stdout_logfile_maxbytes = 20MB
13 stdout_logfile_backups = 20
14 stdout_logfile = /var/log/supervisor/nginx.log
```



```

15
16 [program:php-fpm]
17 command=php-fpm -c /etc/php.ini -y /etc/php-fpm.conf
18 autostart=true
19 autorestart=true
20 startsecs = 5
21 redirect_stderr = true
22 stdout_logfile_maxbytes = 20MB
23 stdout_logfile_backups = 20
24 stdout_logfile = /var/log/supervisor/php-fpm.log
25
26 #3.修改supervisord配置文件放在前台启动
27 sed -i "s#nodaemon=false#nodaemon=true#g" /etc/supervisord.conf
28
29 #4.启动supervisord程序
30 supervisord -c /etc/supervisord.conf
31
32 #5.使用命令
33 supervisorctl update
34 supervisorctl status
35 supervisorctl start nginx
36 supervisorctl restart nginx
37 supervisorctl stop nginx

```

6.使用Dockerfile创建云盘镜像

```

1  基于Dockerfile构建云盘镜像
2
3  1.先创建目录
4  [root@docker-11 ~]# cd dockerfile/
5  [root@docker-11 ~/dockerfile]# ls
6  nginx_base
7  [root@docker-11 ~/dockerfile]# mkdir kod
8  [root@docker-11 ~/dockerfile]# cd kod/
9  [root@docker-11 ~/dockerfile/kod]#
10
11 2.收集配置文件
12 mkdir conf
13 cd conf
14 cat > local.repo << EOF
15 [local]
16 name=local
17 enable=1
18 gpgcheck=0
19 baseurl=http://10.0.0.100
20 EOF
21 docker cp b7757c17bf36:/etc/php-fpm.d/www.conf .
22 docker cp b7757c17bf36:/etc/nginx/conf.d/cloud.conf .
23 docker cp b7757c17bf36:/etc/supervisord.conf .
24 docker cp b7757c17bf36:/etc/supervisord.d/nginx_php.ini .
25
26 3.准备代码目录
27 mkdir code/
28 cd code
29 docker cp b7757c17bf36:/code/ .
30 tar zcvf code.tar.gz code
31

```

```

32 4.编写Dockerfile
33 FROM centos:7
34 RUN rm -rf /etc/yum.repos.d/*
35 ADD conf/local.repo /etc/yum.repos.d/local.repo
36 RUN yum install nginx php-fpm php-mbstring php-gd supervisor -y
37 ADD conf/www.conf /etc/php-fpm.d/www.conf
38 ADD conf/cloud.conf /etc/nginx/conf.d/cloud.conf
39 ADD conf/supervisord.conf /etc/supervisord.conf
40 ADD conf/nginx_php.ini /etc/supervisord.d/nginx_php.ini
41 ADD code/code.tar.gz /
42 RUN chown -R nginx:nginx /code/
43 EXPOSE 80
44 CMD ["supervisord","-c","/etc/supervisord.conf"]
45
46 5.构建镜像
47 docker build -t kod:v2 .
48
49 6.启动容器测试
50 docker run -d --name kod_v1 -p 8080:80 kod:v1

```

7.使用Dockerfile创建KVM图形化管理工具镜像

1

8.构建Tomcat镜像

8.1 构建基础CentOS7镜像

```

1 #1.基础镜像需要的操作
2 安装网络工具包
3 配置yum源
4 更改时区
5
6 #2.创建目录
7 mkdir centos7
8 cd centos7/
9
10 #3.准备配置文件
11 [root@docker-11 centos7]# ll
12 总用量 16
13 -rw-r--r-- 1 root root 1759 7月 22 21:02 CentOS-Base.repo
14 -rw-r--r-- 1 root root 664 7月 22 21:02 epel.repo
15
16 #4.编写dockerfile
17 FROM centos:7
18 RUN rm -f /etc/yum.repos.d/*
19 ADD CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo
20 ADD epel.repo /etc/yum.repos.d/epel.repo
21 ADD supervisord.conf /etc/supervisord.conf
22 RUN yum install net-tools bash-completion supervisor -y \
23     && yum clean all \
24     && rm -f /etc/localtime \
25     && ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime \
26     && groupadd -g 1000 www \
27     && useradd -u 1000 -g 1000 www -M -s /sbin/nologin

```

```
28
29 #5.构建命令
30 docker build -t centos7_base:v1 .
```

8.2 构建基础JDK镜像

```
1 #1.创建目录
2 mkdir dockerfile/jdk/
3
4 #2.准备配置文件
5 [root@docker-11 ~]# cd dockerfile/jdk/
6 [root@docker-11 jdk]# docker cp 28e7ff621f8c:/etc/profile .
7 [root@docker-11 jdk]# vim profile
8 [root@docker-11 jdk]# tail -4 dockerfile/jdk/profile
9 export JAVA_HOME=/opt/jdk
10 export TOMCAT_HOME=/opt/tomcat
11 export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$TOMCAT_HOME/bin:$PATH
12 export
13 CLASSPATH=.$CLASSPATH:$JAVA_HOME/lib:$JAVA_HOME/jre/lib:$JAVA_HOME/lib/tools.
14 jar
15
16 #3.将jdk上传到指定目录
17 [root@docker-11 ~]# tree dockerfile/
18 dockerfile/
19 └─ jdk
20     ├── jdk-8u60-linux-x64.tar.gz
21     └─ profile
22
23 #4.编写Dockerfile
24 [root@docker-11 jdk]# cat Dockerfile
25 FROM centos7_base:v1
26 ADD jdk-8u60-linux-x64.tar.gz /opt/
27 ADD profile /etc/profile
28 RUN ln -s /opt/jdk1.8.0_60 /opt/jdk
29 ENV JAVA_HOME /opt/jdk
30 ENV JRE_HOME $JAVA_HOME/jre
31 ENV CLASSPATH $JAVA_HOME/lib/:$JRE_HOME/lib/
32 ENV PATH $PATH:$JAVA_HOME/bin
33
34 #5.编写构建命令脚本
35 [root@docker-11 jdk]# cat > /root/dockerfile/jdk/build.sh << 'EOF'
36 #!/bin/bash
37 docker build -t centos7_jdk:8u60 .
38 EOF
39 [root@docker-11 jdk]# bash build.sh
40
41 #6.查看构建后的镜像
42 [root@docker-11 jdk]# docker images|grep jdk
43 centos7_jdk 8u60 58a880ff9253 19 seconds ago 569MB
44
45 #7.使用jdk镜像启动容器测试
46 [root@docker-11 jdk]# docker run -it --rm centos7_jdk:8u60 java -version
47 java version "1.8.0_60"
48 Java(TM) SE Runtime Environment (build 1.8.0_60-b27)
49 Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
```

8.3 构建Tomcat镜像

```
1 #1.创建目录
2 [root@docker-11 ~]# mkdir dockerfile/tomcat
3
4 #2.上传压缩包
5 [root@docker-11 ~]# cd dockerfile/tomcat
6 [root@docker-11 tomcat]# wget
https://mirrors.tuna.tsinghua.edu.cn/apache/tomcat/tomcat-
8/v8.5.69/bin/apache-tomcat-8.5.69.tar.gz
7
8 #3.编写Dockerfile
9 [root@docker-11 tomcat]# cat > Dockerfile << 'EOF'
10 FROM centos7_jdk:8u60
11 ADD apache-tomcat-8.5.69.tar.gz /opt/
12 RUN ln -s /opt/apache-tomcat-8.5.69 /opt/tomcat
13 EOF
14
15 #4.编写构建命令脚本
16 [root@docker-11 jdk]# cat > /root/dockerfile/jdk/build.sh << 'EOF'
17 #!/bin/bash
18 docker build -t tomcat_base:8.5.69 .
19 EOF
20 [root@docker-11 tomcat]# tree
21 .
22 ├── apache-tomcat-8.5.69.tar.gz
23 ├── build.sh
24 └── Dockerfile
25
26 #5.构建镜像
27 [root@docker-11 tomcat]# bash build.sh
28
29 #6.测试访问
30 [root@docker-11 tomcat]# docker run -d -it tomcat_base:8.5.69 /bin/bash
31 [root@docker-11 tomcat]# docker exec -it b2011b0b3eb3 /bin/bash
32 [root@b2011b0b3eb3 /]# /opt/tomcat/bin/catalina.sh start
```

8.4 构建业务镜像

```
1 #1.创建目录
2 [root@docker-11 ~]# mkdir dockerfile/webapp1
3
4 #2.编写业务文件
5 [root@docker-11 ~]# cd dockerfile/webapp1
6 [root@docker-11 webapp1]# mkdir app
7 [root@docker-11 webapp1]# echo "v1" > app/index.jsp
8 [root@docker-11 webapp1]# tar zcf app.tar.gz app/
9
10 #3.修改tomcat配置文件
11 [root@docker-11 webapp1]# docker cp b2011b0b3eb3:/opt/tomcat/conf/server.xml
.
12 [root@docker-11 webapp1]# vim server.xml
13 <Host name="localhost" appBase="/opt/tomcat/webapps"
14
15 #4.编写supervisor配置文件
16 [root@docker-11 webapp1]# cat tomcat.ini
```

```
17 [program:tomcat]
18 command=/opt/tomcat/bin/catalina.sh run
19 autostart=true
20 autorestart=true
21 startsecs = 5
22 redirect_stderr = true
23 stdout_logfile_maxbytes = 20MB
24 stdout_logfile_backups = 20
25 stdout_logfile = /var/log/supervisor/tomcat.log
26
27 #5.编写Dockerfile文件
28 [root@docker-11 webapp1]# cat Dockerfile
29 FROM tomcat_base:8.5.69
30 ADD tomcat.ini /etc/supervisord.d/tomcat.ini
31 ADD server.xml /opt/tomcat/conf/server.xml
32 ADD app.tar.gz /opt/tomcat/webapps/
33 EXPOSE 8080
34 CMD ["supervisord","-c","/etc/supervisord.conf"]
35
36 #6.编写构建脚本
37 [root@docker-11 webapp1]# cat > build.sh << 'EOF'
38 #!/bin/bash
39 docker build -t tomcat_app:v1 .
40 EOF
41
42 #7.运行容器测试
43 docker run -d -it -p 8080:8080 tomcat_app:v1
```

第8章 企业级私有仓库Docker-harbor

1.部署步骤

- 1 第一步：安装docker和docker-compose
- 2 第二步：下载harbor-offline-installer-v1.9.0-rc1.tgz
- 3 第三步：上传到/opt,并解压
- 4 第四步：修改harbor.yml配置文件 hostname = 10.0.0.11 harbor_admin_password = 123456
- 5 第五步：执行install.sh

2.安装docker-compose

1.安装docker-compose

```
1 yum install -y docker-compose
```

2.检查

```
1 docker-compose version
```

3.上传解压docker-harbor

```
1 [root@docker-11 ~]# cd /opt/
2 [root@docker-11 /opt]# ls
3 harbor-offline-installer-v1.9.0-rc1.tgz
4 [root@docker-11 /opt]# tar xzf harbor-offline-installer-v1.9.0-rc1.tgz
5 [root@docker-11 /opt]# ls
6 harbor harbor-offline-installer-v1.9.0-rc1.tgz
7 [root@docker-11 /opt]# cd harbor/
```

4.修改配置文件

修改2个地方：

```
1 [root@docker-11 /opt/harbor]# egrep "10.0.0.11|123456" harbor.yml
2 hostname: 10.0.0.11
3 harbor_admin_password: 123456
```

5.安装

```
1 [root@docker-11 /opt/harbor]# ./install.sh
```

6.修改docker信任仓库

```
1 [root@docker-11 /opt/harbor]# cat /etc/docker/daemon.json
2 {
3     "registry-mirrors": ["http://hub-mirror.c.163.com"],
4     "insecure-registries": ["http://10.0.0.11"]
5 }
```

7.重启docker

```
1 systemctl restart docker
```

8.给镜像打标签并提交到harbor

docker登陆harbor

```
1 [root@docker-11 /opt/harbor]# docker login 10.0.0.11
2 Username: zhangya
3 Password:
4 WARNING! Your password will be stored unencrypted in
   /root/.docker/config.json.
5 Configure a credential helper to remove this warning. See
6 https://docs.docker.com/engine/reference/commandline/login/#credentials-store
7
8 Login Succeeded
```

运行一个容器：

```
1 [root@docker-11 ~]# docker run -d -p 8080:80 centos_kod:v1
2 78be80f7c2029b68e8943e38fa99131ec6709f798e63c94afb5a7fdfa4a8047c
```

查看容器ID:

```
1 [root@docker-11 ~]# docker ps|grep kod
2 78be80f7c202          centos_kod:v1
   "/bin/bash /init.sh"    15 seconds ago      Up 13 seconds
   0.0.0.0:8080->80/tcp      tender_dirac
```

将容器提交为新镜像并且更改为harbor仓库的地址

```
1 [root@docker-11 ~]# docker commit 78be80f7c202 10.0.0.11/linux/centos_kod:v1
2 sha256:6bf1e1eef1969bcd4c82472aed945d4dda74a923c0d7dae91e38539676f8c240
```

查看镜像

```
1 [root@docker-11 ~/dockerfile/kod]# docker images
2 REPOSITORY              TAG          IMAGE ID
3 10.0.0.11/linux/centos_kod v1           6bf1e1eef196
   13 minutes ago         465MB
```

将新镜像推送到harbor上

```
1 [root@docker-11 /opt/harbor]# docker push 10.0.0.11/linux/centos_kod:v1
```

9.在docker-harbor上查看

```
1 10.0.0.11
2 账号:admin
3 密码:123456
```

10.其他主机上下载镜像

配置docker信任仓库

```
1 [root@docker-12 ~]# cat /etc/docker/daemon.json
2 {
3     "registry-mirrors": ["http://hub-mirror.c.163.com"],
4     "insecure-registries": ["http://10.0.0.11"],
5 }
```

从Harbor仓库拉取镜像

```
1 [root@docker-12 ~]# docker pull 10.0.0.11/linux/centos_kod:v1
```

第9章 Docker网络模式

1.Docker网络的四种模式

- | | | |
|---|-----------|---|
| 1 | Host | 容器将不会虚拟出自己的网卡，配置自己的IP等，而是使用宿主机的IP和端口。 |
| 2 | Bridge | 此模式会为每一个容器分配、设置IP等，并将容器连接到一个docker0虚拟网桥，通过docker0网桥以及Iptables nat表配置与宿主机通信。 |
| 3 | None | 此模式关闭了容器的网络功能。 |
| 4 | Container | 创建的容器不会创建自己的网卡，配置自己的IP，而是和一个指定的容器共享IP、端口范围。 |

查看网络模式命令:

```
1 [root@node-51 ~]# docker network ls
2 NETWORK ID          NAME           DRIVER          SCOPE
3 3791e4fc9c18        bridge        bridge         local
4 b494337929ef        host         host          local
5 a153ac0003e3        none         null          local
```

查看网卡命令:

```
1 [root@node-51 ~]# ip a
2 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
   default qlen 1000
3     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4     inet 127.0.0.1/8 scope host lo
5         valid_lft forever preferred_lft forever
6     inet6 ::1/128 scope host
7         valid_lft forever preferred_lft forever
8 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
   group default qlen 1000
9     link/ether 00:0c:29:11:6b:18 brd ff:ff:ff:ff:ff:ff
10    inet 10.0.0.51/24 brd 10.0.0.255 scope global eth0
11        valid_lft forever preferred_lft forever
12    inet6 fe80::20c:29ff:fe11:6b18/64 scope link
13        valid_lft forever preferred_lft forever
14 3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
   DOWN group default
15    link/ether 02:42:bb:96:63:c7 brd ff:ff:ff:ff:ff:ff
16    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
17        valid_lft forever preferred_lft forever
```

查看桥接网卡命令

```
1 yum install bridge-utils -y
2 brctl show
```

2. Bridge模式

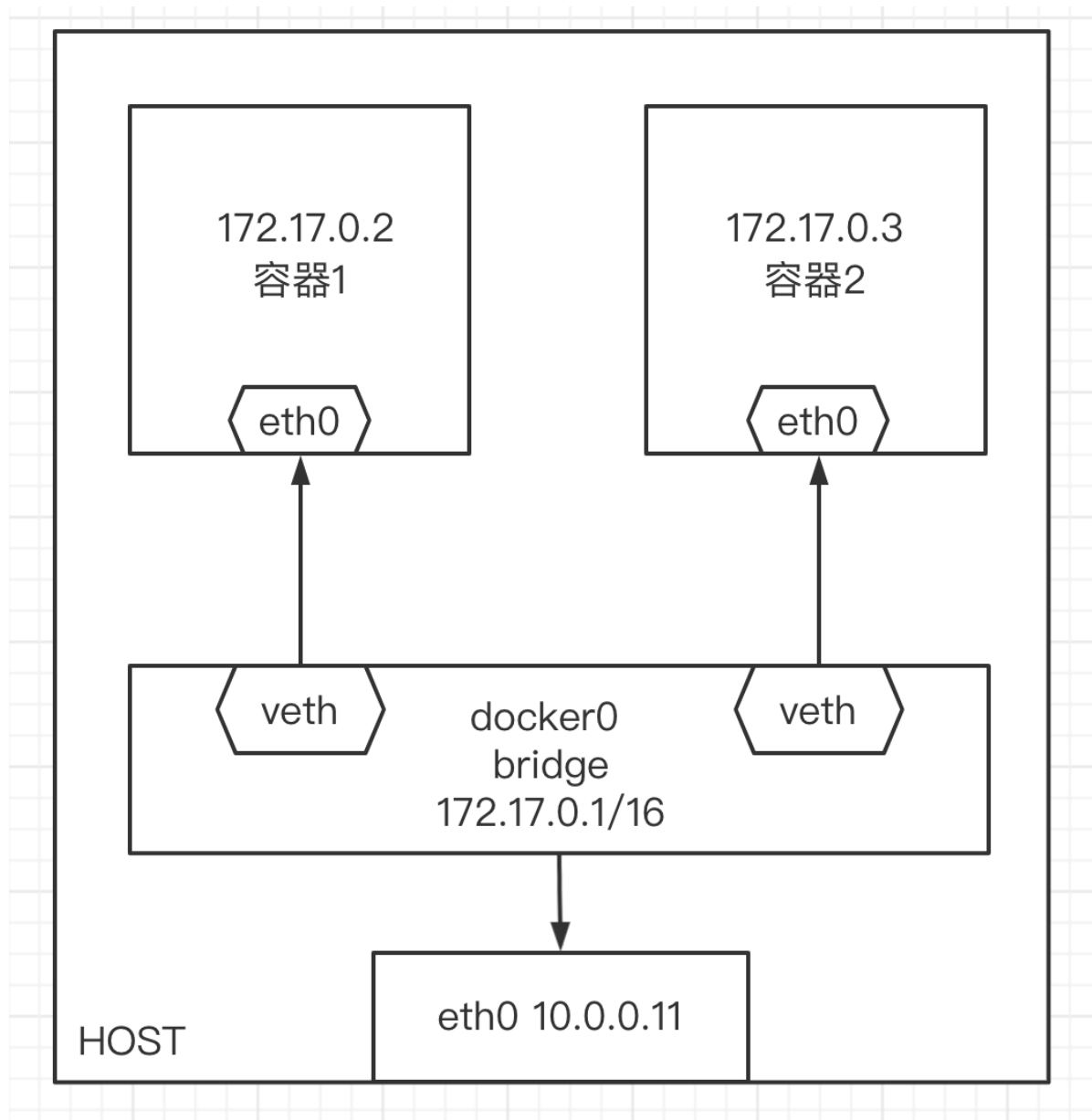
2.1 Bridge模式介绍

1. 当Docker Daemon第一次启动时会创建一个虚拟的网桥，默认名称是Docker0
2. 创建完后会给这个网桥分配一个子网，默认是172.17.0.1/16
3. 由Docker创建的每一个容器，都会创建一个veth设备对，其中一端关联到网桥上，另一端放在容器里映射为eth0，然后从网桥的地址段内给容器内的eth0分配一个IP地址，这样容器之间就可以互通了。

网络模式特点：

1. 同一宿主机的容器之间可以互相通信，不同宿主机之间不能互相通信
2. 桥接模式的容器可以自动获取172.17.0.0/16网段的IP地址
3. 其他机器不能直接访问容器，可以通过映射端口的形式访问
4. 每个容器映射到宿主机的端口不能重复
5. 容器可以借助宿主机的网络访问其他机器

2.2 Bridge模式示意图



2.3 查看Bridge的详细信息

查看桥接模式的详细信息：

```
1 [root@docker-11 ~]# docker network inspect bridge
```

容器内查看：

```
1 [root@docker-11 ~]# docker run -it busybox /bin/sh
2 / # cd
3 ~ # ip a
```

```

4 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
5     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
6     inet 127.0.0.1/8 scope host lo
7         valid_lft forever preferred_lft forever
8 14: eth0@if15: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue
9     link/ether 02:42:ac:11:00:05 brd ff:ff:ff:ff:ff:ff
10    inet 172.17.0.5/16 brd 172.17.255.255 scope global eth0
11        valid_lft forever preferred_lft forever
12 ~ #
13 ~ # route -n
14 Kernel IP routing table
15 Destination      Gateway            Genmask           Flags Metric Ref    Use Iface
16 0.0.0.0           172.17.0.1        0.0.0.0           UG      0      0      0 eth0
17 172.17.0.0       0.0.0.0           255.255.0.0       U       0      0      0 eth0
18
19 ~ # ping 10.0.0.12 -c 1
20 PING 10.0.0.12 (10.0.0.12): 56 data bytes
21 64 bytes from 10.0.0.12: seq=0 ttl=63 time=0.471 ms
22
23 --- 10.0.0.12 ping statistics ---
24 1 packets transmitted, 1 packets received, 0% packet loss
25 round-trip min/avg/max = 0.471/0.471/0.471 ms
26
27 ~ # traceroute 10.0.0.12
28 traceroute to 10.0.0.12 (10.0.0.12), 30 hops max, 46 byte packets
29  1  172.17.0.1 (172.17.0.1)  0.010 ms  0.005 ms  0.005 ms
30  2  10.0.0.12 (10.0.0.12)  0.257 ms  0.246 ms  0.192 ms

```

2.4 修改桥接模式默认的网络配置

方法1: 修改systemd文件添加bip参数

```

1 [root@docker-11 ~]# vim /lib/systemd/system/docker.service
2 ExecStart=/usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock --bip=192.168.1.1/24
3 [root@docker-11 ~]# systemctl daemon-reload
4 [root@docker-11 ~]# systemctl restart docker.service
5 [root@docker-11 ~]# ip a
6 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
7     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
8     inet 127.0.0.1/8 scope host lo
9         valid_lft forever preferred_lft forever
10    inet6 ::1/128 scope host
11        valid_lft forever preferred_lft forever
12 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
13     link/ether 00:0c:29:9a:74:d9 brd ff:ff:ff:ff:ff:ff
14     inet 10.0.0.11/24 brd 10.0.0.255 scope global eth0
15         valid_lft forever preferred_lft forever
16     inet6 fe80::20c:29ff:fe9a:74d9/64 scope link
17         valid_lft forever preferred_lft forever
18 3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
19     link/ether 02:42:ed:fc:67:1f brd ff:ff:ff:ff:ff:ff
20     inet 192.168.1.1/24 brd 192.168.1.255 scope global docker0

```

```

21         valid_lft forever preferred_lft forever
22         inet6 fe80::42:edff:fe6c:671f/64 scope link
23         valid_lft forever preferred_lft forever
24
25 [root@docker-11 ~]# docker run -it busybox /bin/sh
26 / # ip a
27 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
28     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
29     inet 127.0.0.1/8 scope host lo
30         valid_lft forever preferred_lft forever
31 18: eth0@if19: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
    noqueue
32     link/ether 02:42:c0:a8:01:02 brd ff:ff:ff:ff:ff:ff
33     inet 192.168.1.2/24 brd 192.168.1.255 scope global eth0
34         valid_lft forever preferred_lft forever

```

方法2: 修改daemon.json文件

```

1 [root@docker-11 ~]# cat /etc/docker/daemon.json
2 {
3     "bip": "192.168.2.1/24",
4     "registry-mirrors": ["https://ig2l3l9y.mirror.aliyuncs.com"]
5 }

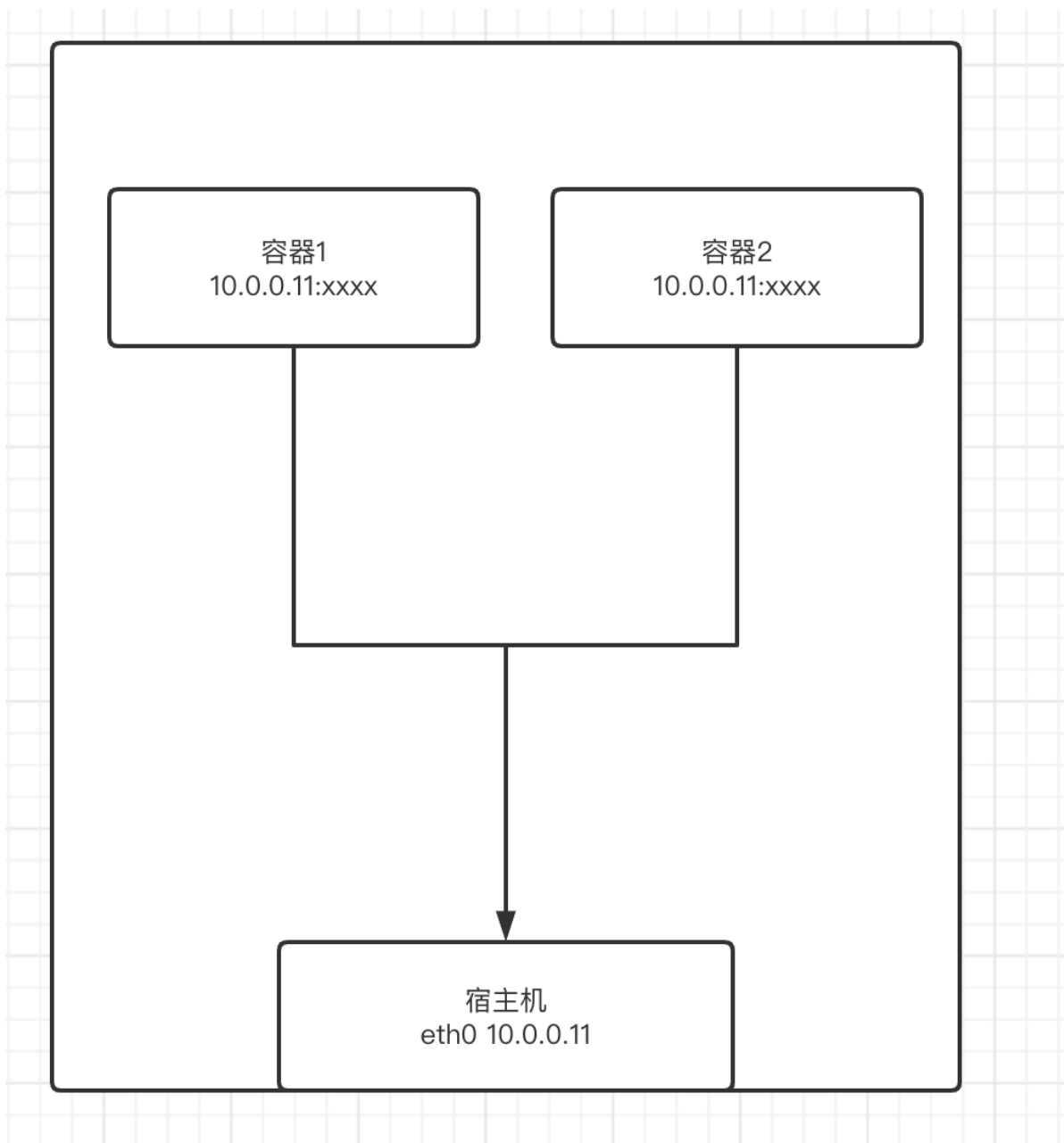
```

3.Host模式

3.1 Host模式说明

- 1 1.Host模式启动的容器不会虚拟出自己的网卡和IP，而是使用宿主机的IP和端口。
- 2 2.但是其他的资源比如文件系统和进程列表还是和宿主机隔离的。
- 3 3.启动容器需要使用指定的参数 `--network host`
- 4 4.Host模式不支持端口映射
- 5 5.因为直接使用宿主机的网络资源，所以性能较好

3.2 Host模式示意图



3.3 Host模式演示

```
1 #查看当前宿主机的端口和容器运行情况
2 [root@docker-11 ~]# netstat -lntup|grep 80
3 [root@docker-11 ~]# docker ps
4 CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
5
6 #运行一个nginx容器
7 [root@docker-11 ~]# docker run -d --network host nginx
8 f6f44b316317f1727d648801836653866fe25f2ad8c24bf6fe9e7e2e8ee1b6ea
9 [root@docker-11 ~]# docker ps
10 CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
11 f6f44b316317  nginx    "/docker-entrypoint...."  5 seconds ago  Up 4
    seconds          charming_blackwell
12
13 #再次查看宿主机端口情况
14 [root@docker-11 ~]# netstat -lntup|grep 80
15 tcp           0        0 0.0.0.0:80      0.0.0.0:*
    20842/nginx: master
```

```

16  tcp6      0      0 :::80          :::*           LISTEN
    20842/nginx: master
17
18  #进入容器后配置源信息
19  [root@docker-11 ~]# docker exec -it f6f44b316317 /bin/bash
20  root@docker-11:/# cat >/etc/apt/sources.list << 'EOF'
21  > deb https://mirrors.tuna.tsinghua.edu.cn/debian/ buster main contrib non-
    free
22  > deb https://mirrors.tuna.tsinghua.edu.cn/debian/ buster-updates main
    contrib non-free
23  > deb https://mirrors.tuna.tsinghua.edu.cn/debian/ buster-backports main
    contrib non-free
24  > deb https://mirrors.tuna.tsinghua.edu.cn/debian-security buster/updates
    main contrib non-free
25  > EOF
26  root@docker-11:/# apt update
27
28  #在容器内安装网络命令
29  root@docker-11:/# apt install iproute2 net-tools -y
30
31  #查看网络信息
32  root@docker-11:/# ifconfig
33  docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
34          inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
35          inet6 fe80::42:edff:fe6c:671f  prefixlen 64  scopeid 0x20<link>
36          ether 02:42:ed:fc:67:1f  txqueuelen 0  (Ethernet)
37          RX packets 22  bytes 1106 (1.0 KiB)
38          RX errors 0  dropped 0  overruns 0  frame 0
39          TX packets 26  bytes 2154 (2.1 KiB)
40          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
41
42  eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
43          inet 10.0.0.11  netmask 255.255.255.0  broadcast 10.0.0.255
44          inet6 fe80::20c:29ff:fe9a:74d9  prefixlen 64  scopeid 0x20<link>
45          ether 00:0c:29:9a:74:d9  txqueuelen 1000  (Ethernet)
46          RX packets 208690  bytes 293033849 (279.4 MiB)
47          RX errors 0  dropped 0  overruns 0  frame 0
48          TX packets 67922  bytes 4640476 (4.4 MiB)
49          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
50
51  lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
52          inet 127.0.0.1  netmask 255.0.0.0
53          inet6 ::1  prefixlen 128  scopeid 0x10<host>
54          loop txqueuelen 1000  (Local Loopback)
55          RX packets 12  bytes 1563 (1.5 KiB)
56          RX errors 0  dropped 0  overruns 0  frame 0
57          TX packets 12  bytes 1563 (1.5 KiB)
58          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

```

3.4 Host模式注意

不能端口映射

```

1  [root@docker-11 ~]# docker run -p 8080:80 --network host -d nginx
2  WARNING: Published ports are discarded when using host network mode
3  e4fc457b171fa488db1c2bb0293d2eaaef8948b5ecbac339feb92f943aa565bf1

```

host模式下端口不能重复.如果再次启动相同端口的虚拟机就会失败

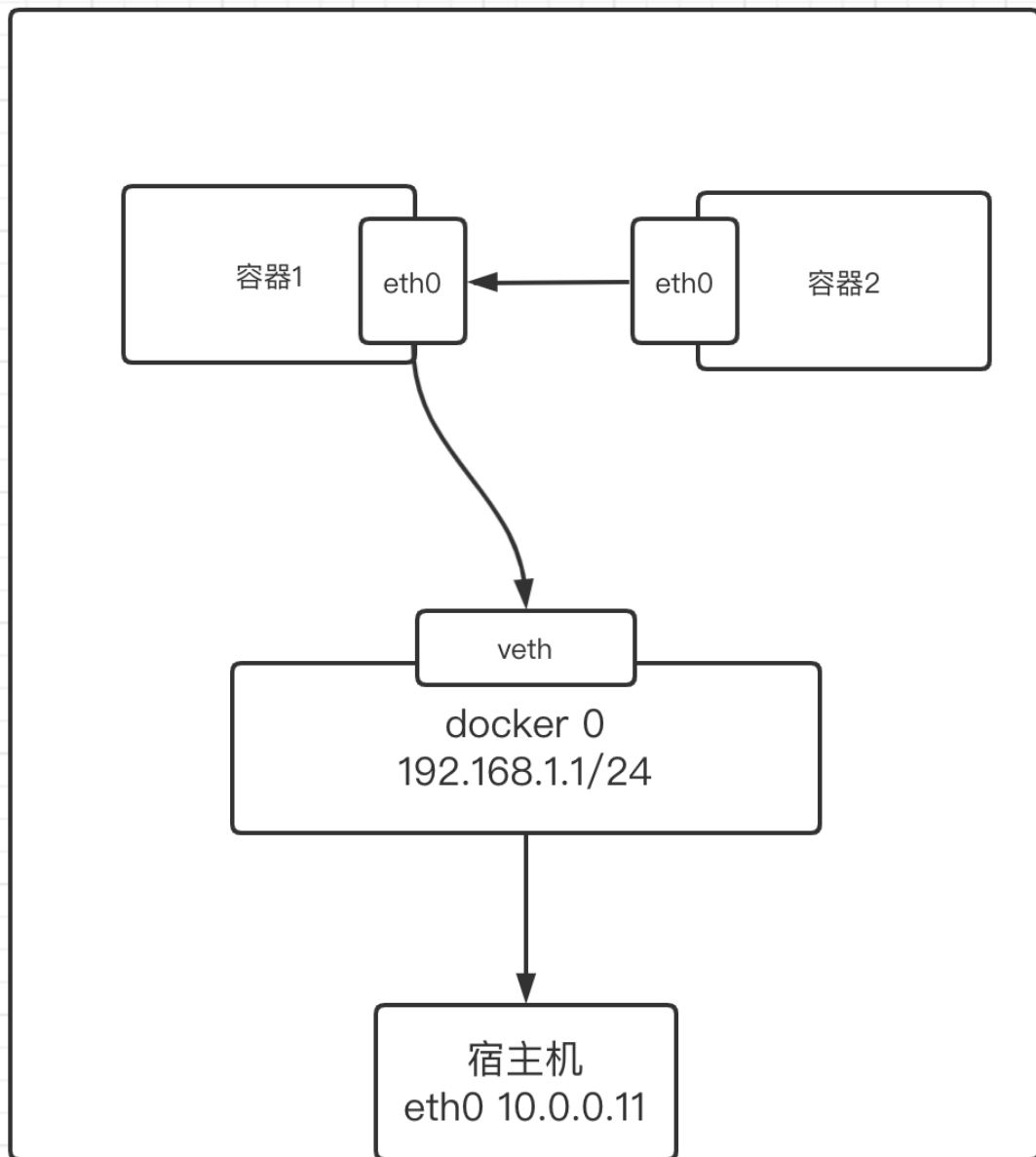
```
1 #启动多个host模式的容器
2 [root@docker-11 ~]# docker run -d --network host nginx
3 638c60c90e5d17f75de1805f81e8b064c4e144616e186b57a8170fc789f1a71b
4 [root@docker-11 ~]# docker run -d --network host nginx
5 9866a4ac70f4a08bd91e3a0f820ee796f1353d3fbf93f5f8ff48a52fb84850e7
6
7 #查看容器发现并没有启动成功
8 [root@docker-11 ~]# docker ps
9 CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
10  f6f44b316317   nginx     "/docker-entrypoint...."  11 minutes ago Up 11
    minutes      charming_blackwell
11
12 #通过查看失败的容器日志发现是端口冲突了
13 [root@docker-11 ~]# docker logs -f 638c60c90e5d
14 /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to
    perform configuration
15 /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
16 /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-
    default.sh
17 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of
    /etc/nginx/conf.d/default.conf
18 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPV6 in
    /etc/nginx/conf.d/default.conf
19 /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-
    templates.sh
20 /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-
    processes.sh
21 /docker-entrypoint.sh: Configuration complete; ready for start up
22 2021/07/21 13:21:17 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address
    already in use)
23 nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
```

4.Container模式

4.1 Container模式说明

1. Container模式创建的容器不会创建自己的网卡和IP, 而是和一个已经存在的容器共享同一个网络空间
2. Container模式的容器和宿主机网络空间互相隔离。

4.2 Container模式示意图



4.3 Container模式演示

```
1 #运行第一个容器没有nginx服务
2 docker run -it --name web1 -p 80:80 nginx
3 docker exec -it web1 /bin/bash
4 curl 127.0.0.1
5
6 #运行第二个容器拥有nginx服务
7 docker run -d --name web2 --network container:web1 nginx
8 curl 127.0.0.1
```

5.None模式

5.1 None模式介绍

- 1 如果使用None模式，则容器不会创建任何网络配置，没有网卡也没有IP地址，因此机会不会使用这种模式

5.2 使用None模式

```
1 [root@docker-11 ~]# docker run --network none -it busybox:latest /bin/sh
2 / #
3 / # ip a
4 / # ping 10.0.0.11
5 PING 10.0.0.11 (10.0.0.11): 56 data bytes
6 ping: sendto: Network is unreachable
7 / # route -n
8 Kernel IP routing table
9 Destination      Gateway            Genmask           Flags Metric Ref    Use Iface
```

6.自定义网络模式

6.1 自定义网络模式特点

- 1 自定义网络可以独立设置容器的使用的网段，而且在同一网络里的虚拟机不需要link就可以直接使用容器名进行互相访问

6.2 自定义网络模式语法

```
1 #创建自定义网络
2 docker network create -d <mode> --subnet <CIDR> --gateway <网关> <自定义网络名称>
3
4 #引用自定义网络
5 docker run --network <自定义网络名称> <镜像名称>
6
7 #删除自定义网络
8 docker network rm <自定义网络名称或网络ID>
```

6.3 自定义网络模式实验

```
1 #创建自定义网络
2 docker network create -d bridge --subnet 192.168.100.0/24 --gateway 192.168.1.1 sz-net
3
4 #查看信息
5 docker inspect my-net
6
7 #查看网卡
8 ip a
9
10 #查看网桥
11 brctl show
12
13 #利用自定义的网络创建容器
14 ##运行第一个容器
15 docker run --name busybox_1 -it --network my-net busybox /bin/sh
16 ip a
17 route -n
18
19 ##运行第二个容器
```



```
20 docker run --name busybox_2 -it --network my-net busybox /bin/sh
21 ip a
22 route -n
23 ping busybox_1
```

第10章 Docker容器单机编排工具

1.docker-compose介绍

- 1 Compose 是用于定义和运行多容器 Docker 应用程序的工具。
- 2 通过Compose，您可以使用YML文件来配置应用程序需要的所有服务。
- 3 写好yaml文件之后，只需要运行一条命令，就会按照资源清单里的配置运行相应的容器服务。

Compose 使用的三个步骤：

- 1 1.使用 Dockerfile 定义应用程序的环境。
- 2 2.使用 docker-compose.yml 定义构成应用程序的服务，这样它们可以在隔离环境中一起运行。
- 3 3.最后，执行 docker-compose up 命令来启动并运行整个应用程序。

官方版本说明:

- 1 <https://docs.docker.com/compose/compose-file/compose-versioning/>

2.安装docker-compose

方法1:直接yum安装-版本比较老

- 1 yum install docker-compose

方法2:使用官方脚本安装

- 1 curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-\$(uname -s)-\$(uname -m)" -o /usr/local/bin/docker-compose
- 2 chmod +x /usr/local/bin/docker-compose
- 3 docker-compose --version

3.docker-compose命令格式

- | | | |
|----|-----------|--|
| 1 | build | #构建镜像 |
| 2 | bundle | #从当前docker compose 文件生成一个以<当前目录>为名称的json格式的Docker Bundle 备 份文件 |
| 3 | config -q | #查看当前配置，没有错误不输出任何信息 |
| 4 | create | #创建服务，较少使用 |
| 5 | down | #停止和删除所有容器、网络、镜像和卷 |
| 6 | events | #从容器接收实时事件，可以指定json 日志格式，较少使用 |
| 7 | exec | #进入指定容器进行操作 |
| 8 | help | #显示帮助细信息 |
| 9 | images | #显示镜像信息 |
| 10 | kill | #强制终止运行中的容器 |
| 11 | logs | #查看容器的日志 |
| 12 | pause | #暂停服务 |
| 13 | port | #查看端口 |

14	ps	#列出容器
15	pull	#重新拉取镜像，镜像发生变化后，需要重新拉取镜像，较少使用
16	push	#上传镜像
17	restart	#重启服务
18	rm	#删除已经停止的服务
19	run	#一次性运行容器
20	scale	#设置指定服务运行的容器个数
21	start	#启动服务
22	stop	#停止服务
23	top	#显示容器运行状态
24	unpause	#取消暂定
25	up	#创建并启动容器

4.docker-compose语法介绍

官方英文参考文档：

1 | <https://github.com/compose-spec/compose-spec/blob/master/spec.md>

菜鸟教程翻译文档：

1 | <https://www.runoob.com/docker/docker-compose.html>

模板案例：

```

1  version: '版本号'
2  services:
3    服务名称1:
4      image: 容器镜像
5      container_name: 容器名称
6      environment:
7        - 环境变量1=值1
8        - 环境变量2=值2
9      volumes:
10       - 存储驱动1:容器内的数据目录路径
11       - 宿主机目录路径:容器内的数据目录路径
12     ports:
13       - 宿主机端口:映射到容器内的端口
14     networks:
15       - 自定义网络的名称
16     links:
17       - namenode
18
19   服务名称2:
20     image: 容器镜像
21     container_name: 容器名称
22     environment:
23       - 环境变量1=值1
24       - 环境变量2=值2
25     volumes:
26       - 存储驱动2:对应容器内的数据目录路径
27     ports:
28       - 宿主机端口:映射到容器内的端口
29     networks:
30       - 自定义网络的名称

```

```
31     links:
32         - namenode
33
34     networks:
35         default:
36             external: true
37             name: 自定义网络名称
```

5.使用docker-compose部署zabbix

官方文档:

```
1 | https://www.zabbix.com/documentation/5.0/zh/manual/installation/containers
```

使用自定义网络:

```
1 | docker network create -d bridge --subnet 172.16.1.0/24 --gateway 172.16.1.1
   | zabbix-net
```

修改后的docker-compose文件:

```
1  version: '3.9'
2  services:
3      mysql:
4          image: mysql:5.7
5          container_name: mysql
6          user: 2000:2000
7          environment:
8              - "MYSQL_ROOT_PASSWORD=123"
9              - "MYSQL_DATABASE=zabbix"
10             - "MYSQL_USER=zabbix"
11             - "MYSQL_PASSWORD=zabbix"
12          command:
13              --character-set-server=utf8
14              --collation-server=utf8_bin
15          volumes:
16              - /data/mysql:/var/lib/mysql
17
18      zabbix-server-mysql:
19          image: zabbix/zabbix-server-mysql
20          container_name: zabbix-server-mysql
21          environment:
22              - "DB_SERVER_HOST=mysql"
23              - "MYSQL_USER=zabbix"
24              - "MYSQL_PASSWORD=zabbix"
25          ports:
26              - "10051:10051"
27          depends_on:
28              - mysql
29
30      zabbix-web-nginx-mysql:
31          image: zabbix/zabbix-web-nginx-mysql
32          container_name: zabbix-web-nginx-mysql
33          environment:
34              - "DB_SERVER_HOST=mysql"
```

```

35     - "MYSQL_USER=zabbix"
36     - "MYSQL_PASSWORD=zabbix"
37     - "ZBX_SERVER_HOST=zabbix-server-mysql"
38     - "PHP_TZ=Asia/Shanghai"
39     ports:
40     - "80:8080"
41
42     networks:
43     default:
44     external: true
45     name: zabbix-net

```

6.使用docker-compose部署wordpress

```

1  version: '3'
2  services:
3    mysql:
4      image: mysql:5.7
5      container_name: mysql
6      user: 2000:2000
7      environment:
8        - "MYSQL_ROOT_PASSWORD=123"
9        - "MYSQL_DATABASE=wordpress"
10       - "MYSQL_USER=wordpress"
11       - "MYSQL_PASSWORD=wordpress"
12     volumes:
13     - "/data/wordpress:/var/lib/mysql"
14     ports:
15     - "3306:3306"
16     command:
17       --character-set-server=utf8
18       --collation-server=utf8_bin
19
20     nginx_php:
21       image: nginx_php:v1
22       container_name: nginx_php
23       ports:
24       - "80:80"
25
26     networks:
27     default:
28     external: true
29     name: wordpress

```

7.docker-compose部署jenkins

```

1  进入jenkins容器里运行dockeri

```

8.docker-compose运行EBK

```

1  version: '2.2'
2  services:
3    es01:
4      image: docker.elastic.co/elasticsearch/elasticsearch:7.13.4

```

```

5     container_name: es01
6     environment:
7         - node.name=es01
8         - cluster.name=es-docker-cluster
9         - discovery.seed_hosts=es02
10        - cluster.initial_master_nodes=es01,es02
11        - bootstrap.memory_lock=true
12        - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
13    ulimits:
14        memlock:
15            soft: -1
16            hard: -1
17    volumes:
18        - /data/es_01:/usr/share/elasticsearch/data
19    ports:
20        - 9200:9200
21    networks:
22        - elastic
23    es02:
24        image: docker.elastic.co/elasticsearch/elasticsearch:7.13.4
25        container_name: es02
26        environment:
27            - node.name=es02
28            - cluster.name=es-docker-cluster
29            - discovery.seed_hosts=es01
30            - cluster.initial_master_nodes=es01,es02
31            - bootstrap.memory_lock=true
32            - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
33        ulimits:
34            memlock:
35                soft: -1
36                hard: -1
37        volumes:
38            - /data/es_02:/usr/share/elasticsearch/data
39        networks:
40            - elastic
41
42    kibana:
43        image: docker.elastic.co/kibana/kibana:7.13.4
44        container_name: kibana
45        environment:
46            SERVER_NAME: 10.0.0.12
47            ELASTICSEARCH_HOSTS: http://es01
48        ports:
49            - 5601:5601
50        networks:
51            - elastic
52
53    networks:
54        elastic:
55            driver: bridge

```

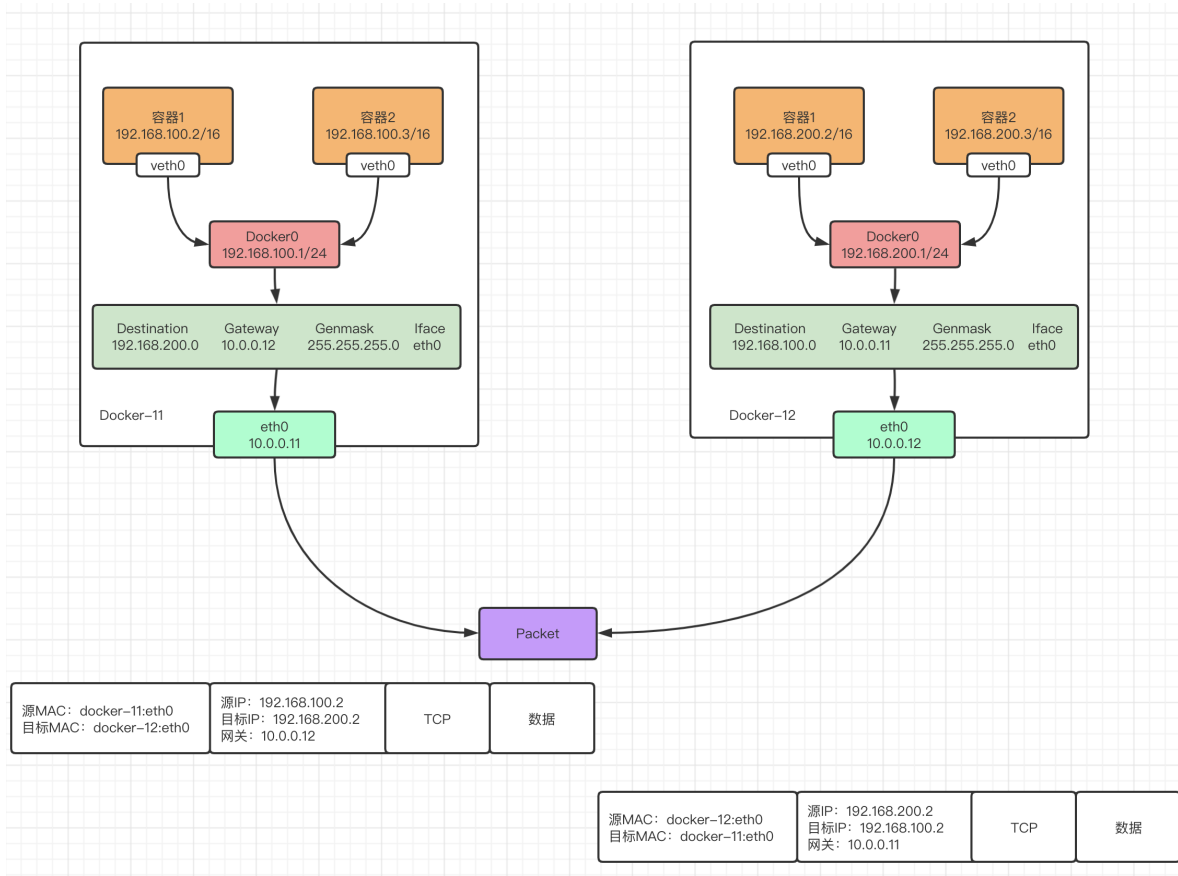
第11章 Docker容器跨主机通信

1.Docker跨主机网络类型

- 1 静态路由
- 2 flannel
- 3 overlay
- 4 macvlan
- 5 calico

2.静态路由模式

2.1 静态路由模式说明



配置说明：

- 1 1. 两台宿主机的容器IP处于不同的网段
- 2 2. 两台主机都配置了静态路由，发给对方的网段的数据包通过eth0网卡，网关指向对方的eth0地址
- 3 3. 防火墙开发内网转发规则

2.2 两台主机创建不同的docker0网段

docker-11配置

```
1 cat > /etc/docker/daemon.json << 'EOF'
2 {
3     "bip": "192.168.100.1/24",
4     "registry-mirrors": ["https://ig2l319y.mirror.aliyuncs.com"]
5 }
6 EOF
7 systemctl daemon-reload
8 systemctl restart docker
9 ip netns exec 192.168.100
```

docker-12配置

```
1 cat > /etc/docker/daemon.json << 'EOF'
2 {
3     "bip": "192.168.200.1/24",
4     "registry-mirrors": ["https://ig2l319y.mirror.aliyuncs.com"]
5 }
6 EOF
7 systemctl daemon-reload
8 systemctl restart docker
9 ip a | grep 192.168.200
```

2.3 添加静态路由和iptables规则

docker-11配置

```
1 route add -net 192.168.200.0/24 gw 10.0.0.12
2 iptables -A FORWARD -s 10.0.0.0/24 -j ACCEPT
```

docker-12配置

```
1 route add -net 192.168.100.0/24 gw 10.0.0.11
2 iptables -A FORWARD -s 10.0.0.0/24 -j ACCEPT
```

2.4 跨主机容器通信测试

docker-11启动容器

```
1 [root@docker-11 ~]# docker run -it busybox /bin/sh
2 / # ip a
3 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
4     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5     inet 127.0.0.1/8 scope host lo
6         valid_lft forever preferred_lft forever
7 13: eth0@if14: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue
8     link/ether 02:42:c0:a8:64:02 brd ff:ff:ff:ff:ff:ff
9     inet 192.168.100.2/24 brd 192.168.100.255 scope global eth0
10         valid_lft forever preferred_lft forever
```

docker-12启动容器

```
1 [root@docker-12 ~]# docker run -it busybox /bin/sh
2 / # ip a
3 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
4     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5     inet 127.0.0.1/8 scope host lo
6         valid_lft forever preferred_lft forever
7 10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
noqueue
8     link/ether 02:42:c0:a8:c8:02 brd ff:ff:ff:ff:ff:ff
9     inet 192.168.200.2/24 brd 192.168.200.255 scope global eth0
10         valid_lft forever preferred_lft forever
```

docker-11启动容器访问docker-12容器测试

```
1 / # ping -c 1 192.168.200.2
2 PING 192.168.200.2 (192.168.200.2): 56 data bytes
3 64 bytes from 192.168.200.2: seq=0 ttl=62 time=0.531 ms
```

docker-12启动容器访问docker-11容器测试

```
1 / # ping -c 1 192.168.100.2
2 PING 192.168.100.2 (192.168.100.2): 56 data bytes
3 64 bytes from 192.168.100.2: seq=0 ttl=62 time=0.631 ms
```

2.5 抓包查看

docker-11抓包

```
1 [root@docker-11 ~]# yum install tcpdump -y
2 [root@docker-11 ~]# tcpdump -i eth0 -nn icmp
3 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
4 listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
5 19:59:41.665444 IP 10.0.0.11 > 192.168.200.2: ICMP echo request, id 7, seq 0,
   length 64
6 19:59:41.665791 IP 192.168.200.2 > 10.0.0.11: ICMP echo reply, id 7, seq 0,
   length 64
```

docker-12抓包

```
1 [root@docker-12 ~]# yum install tcpdump -y
2 [root@docker-12 ~]# tcpdump -i eth0 -nn icmp
3 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
4 listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
5 20:01:20.616844 IP 10.0.0.12 > 192.168.100.2: ICMP echo request, id 7, seq 0,
   length 64
6 20:01:20.617351 IP 192.168.100.2 > 10.0.0.12: ICMP echo reply, id 7, seq 0,
   length 64
```

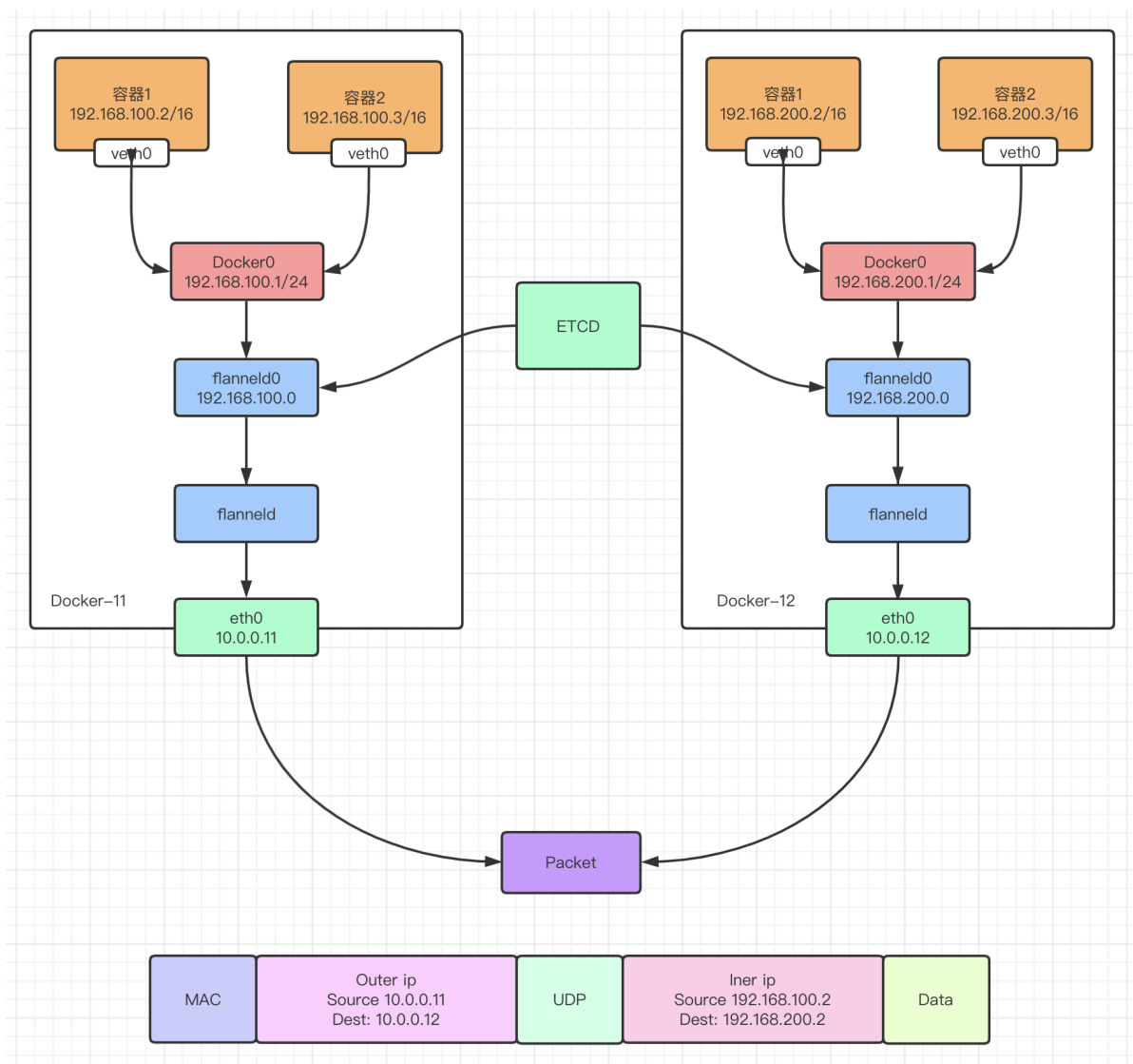
3.跨主机通信-flannel实现

3.1 flannel介绍

```
1 Flannel是一种基于overlay网络的跨主机容器网络解决方案，即将TCP数据包封装在另一种网络包里面进行路由转发和通信，Flannel是CoreOS开发，专门用于docker多机互联的一个工具，让集群中的不同节点主机创建的容器都具有全集群唯一的虚拟ip地址
```

3.2 flannel通信原理

流程图解



文字说明

1. 数据从源容器中发出后，经由所在主机的docker0虚拟网卡转发到flannel0虚拟网卡。
2. 源主机的flannelD服务将原本的数据内容UDP封装后根据自己的路由表投递给目的节点的flannelD服务，数据到达目标主机后被解包，然后直接进入目的节点的flannel0虚拟网卡，然后被转发到目的主机的docker0虚拟网卡，最后就像本机容器通信一样由docker0路由到达目标容器。
3. 使每个结点上的容器分配的地址不冲突。Flannel通过Etcd分配了每个节点可用的IP地址段后，再修改Docker的启动参数。“--bip=X.X.X.X/X”这个参数，它限制了所在节点容器获得的IP范围。

3.3 实验环境

- 1 | 10.0.0.11 | etcd, flannel, docker
- 2 | 10.0.0.12 | flannel, docker

3.4 docker-11安装配置etcd

单节点安装etcd

- 1 | yum install etcd -y

编辑配置文件

```
1 cat > /etc/etcd/etcd.conf << 'EOF'
2 # [member]
3 ETCD_NAME=default
4 ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
5 ETCD_LISTEN_CLIENT_URLS="http://10.0.0.11:2379,http://127.0.0.1:2379"
6
7 # #[cluster]
8 ETCD_INITIAL_CLUSTER_STATE="new"
9 ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
10 ETCD_ADVERTISE_CLIENT_URLS="http://10.0.0.11:2379"
11 EOF
```

启动etcd

```
1 systemctl start etcd
2 systemctl enable etcd
```

测试etcd功能

```
1 etcdctl -C http://10.0.0.11:2379 cluster-health
2 etcdctl -C http://10.0.0.11:2379 set /testdir/testkey "Hello world"
3 etcdctl -C http://10.0.0.11:2379 get /testdir/testkey
```

防火墙

```
1 iptables -A INPUT -p tcp -m tcp --dport 2379 -m state --state NEW,ESTABLISHED
  -j ACCEPT
2 iptables -A INPUT -p tcp -m tcp --dport 2380 -m state --state NEW,ESTABLISHED
  -j ACCEPT
```

3.5 安装配置Flannel-两台机器都操作

安装Flannel

```
1 yum install flannel -y
```

配置Flannel

```
1 cp /etc/sysconfig/flanneld /opt/flanneld.bak
2 cat > /etc/sysconfig/flanneld << 'EOF'
3 # Flanneld configuration options
4
5 # etcd url location. Point this to the server where etcd runs
6 FLANNEL_ETCD_ENDPOINTS="http://10.0.0.11:2379"
7
8 # etcd config key. This is the configuration key that flannel queries
9 # For address range assignment
10 FLANNEL_ETCD_PREFIX="/atomic.io/network"
11
12 # Any additional options that you want to pass
13 #FLANNEL_OPTIONS=""
14 EOF
```

配置etcd数据库

```
1 | etcdctl mk /atomic.io/network/config '{ "Network": "192.168.0.0/16" }'
```

启动flanneld

```
1 | systemctl start flanneld.service
2 | systemctl enable flanneld.service
```

检查端口

```
1 | netstat -lntup|grep flannel
```

3.6 配置Docker关联Flannel网络

修改docker配置文件:

```
1 | vim /usr/lib/systemd/system/docker.service
2 | .....
3 | EnvironmentFile=/run/flannel/docker
4 | ExecStart=/usr/bin/dockerd -H fd:// $DOCKER_NETWORK_OPTIONS
5 | .....
6 | systemctl daemon-reload
7 | systemctl restart docker
```

3.7 创建防火墙规则

```
1 | iptables -P FORWARD ACCEPT
```

3.8 创建容器测试

docker-11创建容器:

```
1 | docker run -it busybox /bin/sh
```

查看IP地址:

```
1 | / # ip a
2 | 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
3 |     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4 |     inet 127.0.0.1/8 scope host lo
5 |         valid_lft forever preferred_lft forever
6 | 11: eth0@if12: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1472 qdisc noqueue
7 |     link/ether 02:42:c0:a8:38:02 brd ff:ff:ff:ff:ff:ff
8 |     inet 192.168.56.2/24 brd 192.168.56.255 scope global eth0
9 |         valid_lft forever preferred_lft forever
```

docker-12创建容器:

```
1 | docker run -it busybox /bin/sh
```

查看IP地址:

```
1 / # ip a
2 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
3     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4     inet 127.0.0.1/8 scope host lo
5         valid_lft forever preferred_lft forever
6 8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1472 qdisc noqueue
7     link/ether 02:42:c0:a8:24:02 brd ff:ff:ff:ff:ff:ff
8     inet 192.168.36.2/24 brd 192.168.36.255 scope global eth0
9         valid_lft forever preferred_lft forever
```

测试容器间可否通讯:

```
1 ping 192.168.58.2
2 ping 192.168.58.3
```

4. macvlan模式

4.1 创建网络

```
1 docker network create -d macvlan --subnet 10.0.0.0/24 --gateway 10.0.0.2 -o
   parent=eth0 macvlan_1
```

4.2 启动容器

docker-11启动容器

```
1 docker run -it --network macvlan_1 --ip 10.0.0.100 alpine
```

docker-12启动容器

```
1 docker run -it --network macvlan_1 --ip 10.0.0.200 alpine
```

启动后互相ping发现可以正常通讯

```
1 ping 10.0.0.100
2 ping 10.0.0.200
```

5. 跨主机通信-Consul实现

5.1 Consul介绍

- 1 Consul是一个服务网格（微服务间的 TCP/IP，负责服务之间的网络调用、限流、熔断和监控）解决方案，它是一个一个分布式的，高度可用的系统，而且开发使用都很简便。
- 2 它提供了一个功能齐全的控制平面，主要特点是：服务发现、健康检查、键值存储、安全服务通信、多数据中心。

5.2 二进制安装步骤

```
1 | wget https://releases.hashicorp.com/consul/1.4.4/consul_1.4.4_linux_amd64.zip
2 | unzip consul_1.4.4_linux_amd64.zip
3 | mv consul /usr/bin/
4 | chmod +x /usr/bin/consul
5 | nohup consul agent -server -bootstrap -ui -data-dir /var/lib/consul -
  | client=10.0.0.11 -bind=10.0.0.11 &>/var/log/consul.log &
6 | tail -f /var/log/consul.log
```

5.3 修改docker-11启动文件

```
1 | [root@docker-11 ~]# vim /lib/systemd/system/docker.service
2 | #ExecStart=/usr/bin/dockerd -H fd:// --
  | containerd=/run/containerd/containerd.sock
3 | ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:2375 -H
  | unix:///var/run/docker.sock --cluster-store consul://10.0.0.11:8500 --cluster-
  | advertise 10.0.0.11:2375
```

5.4 重启docker-11

```
1 | systemctl daemon-reload
2 | systemctl restart docker.service
```

5.5 同样方法修改docker-12的配置

```
1 | [root@docker-12 ~]# vim /lib/systemd/system/docker.service
2 | #ExecStart=/usr/bin/dockerd -H fd:// --
  | containerd=/run/containerd/containerd.sock
3 | ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:2375 -H
  | unix:///var/run/docker.sock --cluster-store consul://10.0.0.11:8500 --cluster-
  | advertise 10.0.0.12:2375
```

5.6 重启docker2

```
1 | systemctl daemon-reload
2 | systemctl restart docker.service
```

5.7 在docker主机上创建overlay网络

在docker1上创建网络，然后会自动同步到docker2上

```
1 | docker network create -d overlay overlay_net
```

5.8 分别在两个节点上创建容器

docker1上运行命令

```
1 | docker run -it --net=overlay_net --name busybox01 busybox:latest
```

docker2上运行命令

```
1 docker run -it --net=overlay_net --name busybox02 busybox:latest
```

5.9 测试联通性

```
1 docker run -it --net=overlay_net --name busybox01 busybox:latest
2 #ping 10.0.0.3
```

第12章 Docker资源限制

1.Docker资源限制说明

```
1
```

2.容器的内存限制

官方文档:

```
1 https://docs.docker.com/config/containers/resource_constraints/
```

Docker限制内存相关参数:

```
1 -m 允许容器使用的最大内存, 单位有k,m,g
2 --oom-kill-disable
```

下载压测工具镜像:

```
1 docker pull lorel/docker-stress-ng
```

压测工具参数说明:

```
1 #查看帮助说明
2 docker run --name mem_test -it --rm lorel/docker-stress-ng
3
4 #常用参数
5 -m N, --vm N          启动N个workers, 默认一个256M内
```

创建一个没有内存限制的容器:

```
1 #启动一个前台窗口任务
2 docker run --name mem_test -it lorel/docker-stress-ng --vm 2
3
4 #另开一个窗口查看
5 CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O
6 49493229356b   c1        99.46%    514.2MiB / 1.934GiB  25.96%    1.1kB / 0B
   0B / 0B      5
```

创建一个限制了内存大小的容器

```

1 #启动一个前台窗口任务
2 docker run --name mem_test --rm -m 300m -it lorel/docker-stress-ng --vm 2
3
4 #新开窗口查看
5 CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %      NET I/O
   BLOCK I/O   PIDS
6 7d1a3b482a3a   mem_test  98.75%    294.9MiB / 300MiB   98.30%     656B / 0B
   0B / 0B      5

```

3.容器的CPU限制

官方文档：

```
1 | https://docs.docker.com/config/containers/resource\_constraints/
```

Docker限制CPU相关参数：

```
1 | --cpus=<value>
```

查看宿主机CPU核数：

```

1 [root@docker-11 ~]# lscpu
2 Architecture:      x86_64
3 CPU op-mode(s):    32-bit, 64-bit
4 Byte Order:        Little Endian
5 CPU(s):             4

```

压测工具命令：

```

1 #不限制容器的CPU使用，压测工具开启4个CPU
2 docker run --name cpu_test -it --rm lorel/docker-stress-ng --cpu 4
3
4 #新开窗口查看CPU占用情况
5 CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %      NET I/O
   BLOCK I/O   PIDS
6 8701e7f14f6f   cpu_test  402.31%    9.973MiB / 1.934GiB   0.50%     1.1kB /
   0B   1.58MB / 0B   5
7
8 #限制容器只能使用1.5个CPU
9 docker run --cpus 1.5 --name cpu_test -it --rm lorel/docker-stress-ng --cpu 4
10
11 #查看容器运行状态
12 CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %      NET I/O
   BLOCK I/O   PIDS
13 ae710912bb3e   cpu_test  149.31%    14.9MiB / 1.934GiB   0.75%     656B / 0B
   0B / 0B      5

```

第13章 Docker监控

1.docker自带的监控命令

- 1 | `docker container ps` :查看正在运行的容器
- 2 | `docker container top` :知道某个容器运行了哪些进程
- 3 | `docker container stats` :显示每个容器各种资源使用情况

2.cAdvisor+ prometheus+ grafana组件介绍

2.1 cAdvisor介绍

- 1 | 1.cAdvisor是google开发的容器监控工具，cAdvisor会显示当前host的资源使用情况，包括CPU，内存，网络，文件系统。
- 2 | 2.不过cAdvisor提供的操作界面略显简陋，而且需要在不同页面之间跳转，并且只能监控一个host，这难免让人质疑他的实用性，但cAdvisor有一个亮点是可以将监控到的数据导出给第三方工具，有这些工具进一步加工处理。
- 3 | 3.所以我们可以把cAdvisor定位为一个监控数据收集器，收集和导出数据是他的强项，而非展示数据。
- 4 | cAdvisor支持很多第三方工具，其中就包含prometheus

2.2 prometheus

- 1 | Prometheus是一个非常优秀的监控工具。提供了监控数据搜集，存储，处理，可视化和告警一系列完整的解决方案。
- 2 | 包含组件：
- 3 | Node Exporter :负责收集host硬件和操作系统数据，以容器的形式运行在所有host上
- 4 | cAdvisor :负责收集容器数据，以容器的形式运行在所有host上

2.3 grafana

- 1 | grafana是一款支持多种数据源的图形展示工具

3.使用docker-compose部署

3.1 地址规划

- 1 | 10.0.0.11 cAdvisor+ Node Exporter +prometheus+ grafana
- 2 | 10.0.0.12 cAdvisor+ Node Exporter

3.2 编写prometheus配置文件

```
1 | cat > prometheus.yml << 'EOF'
2 | scrape_configs:
3 |   - job_name: cadvisor
4 |     scrape_interval: 5s
5 |     static_configs:
6 |       - targets:
7 |         - 10.0.0.11:8080
8 |         - 10.0.0.12:8080
9 |
10 |   - job_name: prometheus
11 |     scrape_interval: 5s
12 |     static_configs:
13 |       - targets:
```



```
14     - 10.0.0.11:9090
15
16 - job_name: node_exporter
17   scrape_interval: 5s
18   static_configs:
19     - targets:
20       - 10.0.0.11:9100
21       - 10.0.0.12:9100
22 EOF
```

3.2 编写docker-compose文件

docker-11配置

```
1 cat >docker-compose.yml<<EOF
2 version: '3.2'
3 services:
4   prometheus:
5     image: prom/prometheus:latest
6     container_name: prometheus
7     ports:
8       - 9090:9090
9     command:
10      - --config.file=/etc/prometheus/prometheus.yml
11     volumes:
12      - ./prometheus.yml:/etc/prometheus/prometheus.yml:ro
13     depends_on:
14      - cadvisor
15
16   node-exporter:
17     image: prom/node-exporter:latest
18     container_name: node_exporter
19     ports:
20      - 9100:9100
21
22   cadvisor:
23     image: google/cadvisor:latest
24     container_name: cadvisor
25     ports:
26      - 8080:8080
27     volumes:
28      - /:/rootfs:ro
29      - /var/run:/var/run:rw
30      - /sys:/sys:ro
31      - /var/lib/docker:/var/lib/docker:ro
32
33   grafana:
34     image: grafana/grafana:latest
35     container_name: grafana
36     ports:
37      - 3000:3000
38 EOF
```

docker-12配置:

```
1 cat >docker-compose.yml<<EOF
```

```

2 version: '3.2'
3 services:
4   node-exporter:
5     image: prom/node-exporter:latest
6     container_name: node_exporter
7     ports:
8       - 9100:9100
9
10  cadvisor:
11    image: google/cadvisor:latest
12    container_name: cadvisor
13    ports:
14      - 8080:8080
15    volumes:
16      - /:/rootfs:ro
17      - /var/run:/var/run:rw
18      - /sys:/sys:ro
19      - /var/lib/docker:/var/lib/docker:ro
20 EOF

```

运行命令：

```
1 docker-compose -f docker-compose.yml up -d
```

4.web页面操作

访问地址:

```

1 10.0.0.11:3000
2 admin admin

```

添加数据源：

```

1 DataSources
2 Name:Prometheus
3 URL:http://10.0.0.11:9090

```

下载监控面板文件:

```
1 https://grafana.com/api/dashboards/10619/revisions/1/download
```

第13章 Jenkins自动化部署Docker

1.部署流程

部署流程：

```

1 1.下载代码
2 2.编译镜像
3 3.推送镜像
4 4.停止正在运行的容器
5 5.启动新容器
6 6.清理jenkins主机上的镜像

```

回滚流程:

1. 选择需要回滚的版本
2. 停止正在运行的容器
3. 启动新容器

2.pipeline脚本

```
1 pipeline{
2     agent any
3
4     parameters {
5         gitParameter name: 'git_version',
6                       branchFilter: 'origin/(.*)',
7                       type: 'PT_TAG',
8                       defaultValue: 'v1.0',
9                       description: '发布新版本'
10        choice(name: 'base_image', choices:
11        ['nginx:1.16','nginx:1.17'],description: '请选择基础镜像版本')
12        choice(name: 'deploy_env', choices:
13        ['deploy','rollback'],description: 'deploy: 发布版本\nrollback: 回滚版本')
14
15    }
16
17    stages{
18        stage("下载代码"){
19            steps{
20                checkout([$class: 'GitSCM',
21                        branches: [[name: '*/master']],
22                        doGenerateSubmoduleConfigurations: false,
23                        extensions: [[$class:
24                        'RelativeTargetDirectory',
25                        relativeTargetDir: 'game']],
26                        submoduleCfg: [],
27                        userRemoteConfigs: [[credentialsId:
28                        'b8c1f793-47ed-4903-995d-2273673d8f87',
29                        url: 'git@10.0.0.200:dev/docker-
30                        pipeline.git']]])
31            }
32        }
33
34        stage("编译镜像"){
35            when {
36                environment name: 'deploy_env', value: 'deploy'
37            }
38            steps{
39                writeFile file: "Dockerfile", text: ""FROM
40                10.0.0.205/base_image/${params.base_image}\nADD game
41                /usr/share/nginx/html/""
42                sh "docker build -t
43                10.0.0.205/image/game:${params.git_version} . && docker push
44                10.0.0.205/image/game:${params.git_version}"
45            }
46        }
47    }
48 }
```

```
39     stage("推送镜像"){
40         when {
41             environment name: 'deploy_env', value: 'deploy'
42         }
43         steps{
44             sh "docker build -t
10.0.0.205/image/game:${params.git_version} . && docker push
10.0.0.205/image/game:${params.git_version}"
45         }
46     }
47
48     stage("部署容器"){
49         when {
50             environment name: 'deploy_env', value: 'deploy'
51         }
52         steps{
53             sh 'ssh 10.0.0.204 "docker stop game && docker rm game &&
docker run --name game -p 80:80 -d 10.0.0.205/image/game:${git_version} &&
docker ps"'
54         }
55     }
56
57     stage("清理构建镜像"){
58         when {
59             environment name: 'deploy_env', value: 'deploy'
60         }
61         steps{
62             sh "docker rmi 10.0.0.205/image/game:${params.git_version}"
63         }
64     }
65
66     stage("回滚镜像"){
67         when {
68             environment name: 'deploy_env', value: 'rollback'
69         }
70
71         steps{
72             sh 'ssh 10.0.0.204 "docker stop game && docker rm game &&
docker run --name game -p 80:80 -d 10.0.0.205/image/game:${git_version} &&
docker ps"'
73         }
74     }
75 }
76 }
```