## **1. 布署软件的问题**

* 如果想让软件运行起来要保证操作系统的设置，各种库和组件的安装都是正确的
* 热带鱼&冷水鱼 冷水鱼适应的水温在5-30度，而热带鱼只能适应22-30度水温，低于22度半小时就冻死了

## **2. 虚拟机**

* 虚拟机（virtual machine）就是带环境安装的一种解决方案。它可以在一种操作系统里面运行另一种操作系统
  + 资源占用多
  + 冗余步骤多
  + 启动速度慢

## **3. Linux 容器**

* 由于虚拟机存在这些缺点，Linux 发展出了另一种虚拟化技术：Linux 容器（Linux Containers，缩写为 LXC）。
* Linux 容器不是模拟一个完整的操作系统，而是对进程进行隔离。或者说，在正常进程的外面套了一个保护层。对于容器里面的进程来说，它接触到的各种资源都是虚拟的，从而实现与底层系统的隔离。
  + 启动快
  + 资源占用少
  + 体积小

## **4. Docker 是什么**

* Docker 属于 Linux 容器的一种封装，提供简单易用的容器使用接口。它是目前最流行的 Linux 容器解决方案。
* Docker 将应用程序与该程序的依赖，打包在一个文件里面。运行这个文件，就会生成一个虚拟容器。程序在这个虚拟容器里运行，就好像在真实的物理机上运行一样

## **5. docker和KVM**

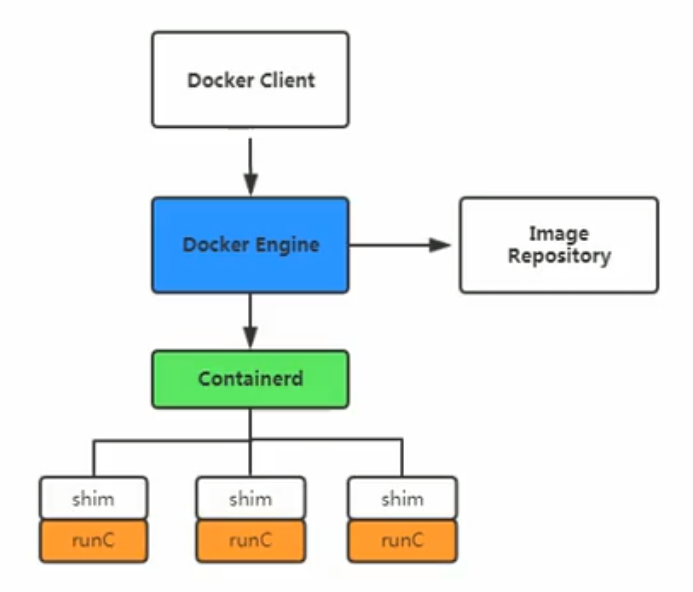
* 启动时间
  + Docker秒级启动
  + KVM分钟级启动
* 轻量级 容器镜像通常以M为单位，虚拟机以G为单位，容器资源占用小，要比虚拟要部署更快速
  + 容器共享宿主机内核，系统级虚拟化，占用资源少，容器性能基本接近物理机
  + 虚拟机需要虚拟化一些设备，具有完整的OS,虚拟机开销大，因而降低性能，没有容器性能好
* 安全性
  + 由于共享宿主机内核，只是进程隔离，因此隔离性和稳定性不如虚拟机，容器具有一定权限访问宿主机内核，存在一下安全隐患
* 使用要求
  + KVM基于硬件的完全虚拟化，需要硬件CPU虚拟化技术支持
  + 容器共享宿主机内核，可运行在主机的Linux的发行版，不用考虑CPU是否支持虚拟化技术

## **6. docker应用场景**

* 节省项目环境部署时间
  + 单项目打包
  + 整套项目打包
  + 新开源技术
* 环境一致性
* 持续集成
* 微服务
* 弹性伸缩

## **7. Docker 体系结构**

* containerd 是一个守护进程，使用runc管理容器，向Docker Engine提供接口
* shim 只负责管理一个容器
* runC是一个轻量级工具，只用来运行容器



## **8. Docker内部组件**

* namespaces 命名空间，Linux内核提供的一种对进程资源隔离的机制，例如进程、网络、挂载等资源
* cgroups 控制组,linux内核提供的一种限制进程资源的机制，例如cpu 内存等资源
* unonFS 联合文件系统，支持将不同位置的目录挂载到同一虚拟文件系统，形成一种分层的模型

## **9. docker安装**

* docker分为企业版(EE)和社区版(CE)
* [docker-ce](https://docs.docker.com/install/linux/docker-ce/centos/)
* [hub.docker](https://hub.docker.com/)

### **9.1 安装**

yum install -y yum-utils device-mapper-persistent-data lvm2

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

yum install docker-ce docker-ce-cli containerd.io

### **9.2 启动**

systemctl start docker

### **9.3 查看docker版本**

$ docker version

$ docker info

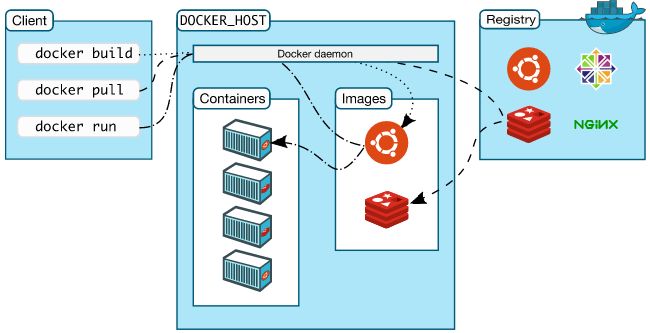
### **9.4 卸载**

docker info

yum remove docker

rm -rf /**var**/lib/docker

## **10. Docker架构**



## **11.阿里云加速**

* [镜像仓库](https://dev.aliyun.com/search.html)
* [镜像加速器](https://cr.console.aliyun.com/cn-hangzhou/instances/mirrors)

sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'

{

"registry-mirrors": ["https://fwvjnv59.mirror.aliyuncs.com"]

}

EOF

# 重载所有修改过的配置文件

sudo systemctl daemon-reload

sudo systemctl restart docker

## **12. image镜像**

* Docker 把应用程序及其依赖，打包在 image 文件里面。只有通过这个文件，才能生成 Docker 容器
* image 文件可以看作是容器的模板
* Docker 根据 image 文件生成容器的实例
* 同一个 image 文件，可以生成多个同时运行的容器实例
* 镜像不是一个单一的文件，而是有多层
* 容器其实就是在镜像的最上面加了一层读写层，在运行容器里做的任何文件改动，都会写到这个读写层里。如果容器删除了，最上面的读写层也就删除了，改动也就丢失了
* 我们可以通过docker history <ID/NAME> 查看镜像中各层内容及大小，每层对应着Dockerfile中的一条指令

| **命令** | **含义** | **语法** | **案例** |
| --- | --- | --- | --- |
| ls | 查看全部镜像 | docker image ls |  |
| search | 查找镜像 | docker search [imageName] |  |
| history | 查看镜像历史 | docker history [imageName] |  |
| inspect | 显示一个或多个镜像详细信息 | docker inspect [imageName] |  |
| pull | 拉取镜像 | docker pull [imageName] |  |
| push | 推送一个镜像到镜像仓库 | docker push [imageName] |  |
| rmi | 删除镜像 | docker rmi [imageName] docker image rmi 2 |  |
| prune | 移除未使用的镜像，没有标记或补任何容器引用 | docker image prune | docker image prune |
| tag | 标记本地镜像，将其归入某一仓库 | docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG] | docker tag centos:7 zhangrenyang/centos:v1 |
| export | 将容器文件系统作为一个tar归档文件导出到STDOUT | docker export [OPTIONS] CONTAINER | docker export -o hello-world.tar b2712f1067a3 |
| import | 导入容器快照文件系统tar归档文件并创建镜像 | docker import [OPTIONS] file/URL/- [REPOSITORY[:TAG]] | docker import hello-world.tar |
| save | 将指定镜像保存成tar文件 | docker save [OPTIONS] IMAGE [IMAGE...] | docker save -o hello-world.tar hello-world:latest |
| load | 加载tar文件并创建镜像 |  | docker load -i hello-world.tar |
| build | 根据Dockerfile构建镜像 | docker build [OPTIONS] PATH / URL / - | docker build -t zf/ubuntu:v1 . |

* 用户既可以使用 docker load 来导入镜像存储文件到本地镜像库，也可以使用 docker import 来导入一个容器快照到本地镜像库
* 这两者的区别在于容器(import)快照文件将丢弃所有的历史记录和元数据信息（即仅保存容器当时的快照状态），而镜像(load)存储文件将保存完整记录，体积也要大
* 此外，从容器(import)快照文件导入时可以重新指定标签等元数据信息

### **12.1 查看镜像**

docker image ls

| **字段** | **含义** |
| --- | --- |
| REPOSITORY | 仓库地址 |
| TAG | 标签 |
| IMAGE\_ID | 镜像ID |
| CREATED | 创建时间 |
| SIZE | 镜像大小 |

### **12.2 查找镜像**

docker search ubuntu

| **字段** | **含义** |
| --- | --- |
| NAME | 名称 |
| DESCRIPTION | 描述 |
| STARTS | 星星的数量 |
| OFFICIAL | 是否官方源 |

### **12.3 拉取镜像**

docker pull docker.io/hello-world

* docker image pull是抓取 image 文件的命令
* docker.io/hello-world是 image 文件在仓库里面的位置，其中docker.io是 image的作者，hello-world是 image 文件的名字
* Docker 官方提供的 image 文件，都放在docker.io组里面，所以它的是默认组，可以省略 docker image pull hello-world

### **12.4 删除镜像**

docker rmi hello-world

## **13. 容器**

* docker run 命令会从 image 文件，生成一个正在运行的容器实例。
* docker container run命令具有自动抓取 image 文件的功能。如果发现本地没有指定的 image 文件，就会从仓库自动抓取
* 输出提示以后，hello world就会停止运行，容器自动终止。
* 有些容器不会自动终止
* image 文件生成的容器实例，本身也是一个文件，称为容器文件
* 容器生成，就会同时存在两个文件： image 文件和容器文件
* 关闭容器并不会删除容器文件，只是容器停止运行

### **13.1 命令**

| **命令** | **含义** | **案例** |
| --- | --- | --- |
| run | 从镜像运行一个容器 | docker run ubuntu /bin/echo 'hello-world' |
| ls | 列出容器 | docker container ls |
| inspect | 显示一个或多个容器详细信息 | docker inspect |
| attach | 要attach上去的容器必须正在运行，可以同时连接上同一个container来共享屏幕 | docker attach [OPTIONS] CONTAINER | docker attach 6d1a25f95132 |
| stats | 显示容器资源使用统计 | docker container stats |  |
| top | 显示一个容器运行的进程 | docker container top |  |
| update | 更新一个或多个容器配置 |  | docker update -m 500m --memory-swap -1 6d1a25f95132 |
| port | 列出指定的容器的端口映射 | docker run -d -p 8080:80 nginx docker container port containerID |  |
| ps | 查看当前运行的容器 | docker ps -a -l |  |
| kill [containerId] | 终止容器(发送SIGKILL ) | docker kill [containerId] |  |
| rm [containerId] | 删除容器 | docker rm [containerId] |  |
| start [containerId] | 启动已经生成、已经停止运行的容器文件 | docker start [containerId] |  |
| stop [containerId] | 终止容器运行 (发送 SIGTERM ) | docker stop [containerId] docker container stop $(docker container ps -aq) |  |
| logs [containerId] | 查看 docker 容器的输出 | docker logs [containerId] |  |
| exec [containerId] | 进入一个正在运行的 docker 容器执行命令 | docker container exec -it f6a53629488b /bin/bash |  |
| cp [containerId] | 从正在运行的 Docker 容器里面，将文件拷贝到本机 | docker container cp f6a53629488b:/root/root.txt . |  |
| commit [containerId] | 根据一个现有容器创建一个新的镜像 | docker commit -a "zhufeng" -m "mynginx" a404c6c174a2 mynginx:v1 |  |

* docker容器的主线程（dockfile中CMD执行的命令）结束，容器会退出
  + 以使用交互式启动 docker run -i [CONTAINER\_NAME or CONTAINER\_ID]
  + tty选项 docker run -dit [CONTAINER\_NAME or CONTAINER\_ID]
  + 守护态（Daemonized）形式运行 docker run -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"

### **13.2 启动容器**

docker run ubuntu /bin/echo "Hello world"

* docker: Docker 的二进制执行文件。
* run:与前面的 docker 组合来运行一个容器。
* ubuntu指定要运行的镜像，Docker首先从本地主机上查找镜像是否存在，如果不存在，Docker 就会从镜像仓库 Docker Hub 下载公共镜像。
* /bin/echo "Hello world": 在启动的容器里执行的命令

Docker以ubuntu镜像创建一个新容器，然后在容器里执行 bin/echo "Hello world"，然后输出结果

* Docker attach必须是登陆到一个已经运行的容器里。需要注意的是如果从这个容器中exit退出的话，就会导致容器停止

| **参数** | **含义** |
| --- | --- |
| -i --interactive | 交互式 |
| -t --tty | 分配一个伪终端 |
| -d --detach | 运行容器到后台 |
| -a --attach list | 附加到运行的容器 |
| -e --env list | 设置环境变量 | docker run -d -p 1010:80 -e username="zhufeng" nginx \ docker container exec -it 3695dc5b9c2d /bin/bash |
| -p --publish list | 发布容器端口到主机 |  |
| -P | --publish-all |  |

### **13.3 查看容器**

docker ps

docker -a

docker -l

* -a 显示所有的容器，包括已停止的
* -l 显示最新的那个容器

| **字段** | **含义** |
| --- | --- |
| CONTAINER ID | 容器ID |
| IMAGE | 使用的镜像 |
| COMMAND | 使用的命令 |
| CREATED | 创建时间 |
| STATUS | 状态 |
| PORTS | 端口号 |
| NAMES | 自动分配的名称 |

### **13.4 运行交互式的容器**

docker run -i -t ubuntu /bin/bash

* -t=--interactive 在新容器内指定一个伪终端或终端。
* -i=--tty 允许你对容器内的标准输入 (STDIN) 进行交互。

我们可以通过运行exit命令或者使用CTRL+D来退出容器。

### **13.5 后台运行容器**

docker run --detach centos ping www.baidu.com

docker ps

docker logs --follow ad04d9acde94

docker stop ad04d9acde94

### **13.6 kill**

docker kill 5a5c3a760f61

kill是不管容器同不同意，直接执行kill -9，强行终止；stop的话，首先给容器发送一个TERM信号，让容器做一些退出前必须的保护性、安全性操作，然后让容器自动停止运行，如果在一段时间内，容器还是没有停止，再进行kill -9，强行终止

### **13.7 删除容器**

* docker rm 删除容器
* docker rmi  删除镜像
* docker rm $(docker ps -a -q)

docker rm 5a5c3a760f61

### **13.8 启动容器**

docker start [containerId]

### **13.9 停止容器**

docker stop [containerId]

### **13.10 进入一个容器**

docker attach [containerID]

### **13.11 进入一个正在运行中的容器**

docker container -exec -it [containerID] /bin/bash

### **13.12 拷贝文件**

docker container cp [containerID] /readme.md .

### **13.13 自动删除**

docker run --rm ubuntu /bin/bash

## **14. commit制作个性化镜像**

* docker commit :从容器创建一个新的镜像。
* docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
  + -a :提交的镜像作者
  + -c :使用Dockerfile指令来创建镜像
  + -m :提交时的说明文字
  + -p :在commit时，将容器暂停
* 停止容器后不会自动删除这个容器，除非在启动容器的时候指定了 --rm 标志
* 使用 docker ps -a 命令查看 Docker 主机上包含停止的容器在内的所有容器
* 停止状态的容器的可写层仍然占用磁盘空间。要清理可以使用 docker container prune 命令

docker container commit -m"我的nginx" -a"zhangrenyang" 3695dc5b9c2d zhangrenyang/mynginx:v1

docker image ls

docker container run zhangrenyang/mynginx /bin/bash

docker container rm b2839066c362

docker container prune

docker image rmi c79ef5b3f5fc

## **15. 制作Dockerfile**

* Docker 的镜像是用一层一层的文件组成的
* docker inspect命令可以查看镜像或者容器
* Layers就是镜像的层文件，只读不能修改。基于镜像创建的容器会共享这些文件层

docker inspect centos

### **15.1 编写Dockerfile**

* -t --tag list 镜像名称
* -f --file string 指定Dockerfile文件的位置

| **指令** | **含义** | **示例** |
| --- | --- | --- |
| FROM | 构建的新镜像是基于哪个镜像 | FROM centos:6 |
| MAINTAINER | 镜像维护者姓名或邮箱地址 | MAINTAINER zhufengjiagou |
| RUN | 构建镜像时运行的shell命令 | RUN yum install httpd |
| CMD | CMD 设置容器启动后默认执行的命令及其参数，但 CMD 能够被 docker run 后面跟的命令行参数替换 | CMD /usr/sbin/sshd -D |
| EXPOSE | 声明容器运行的服务器端口 | EXPOSE 80 443 |
| ENV | 设置容器内的环境变量 | ENV MYSQL\_ROOT\_PASSWORD 123456 |
| ADD | 拷贝文件或目录到镜像中，如果是URL或者压缩包会自动下载和解压 | ADD ,ADD <https://xxx.com/html.tar.gz> /var/[www.html](http://www.html/), ADD html.tar.gz /var/www/html |
| COPY | 拷贝文件或目录到镜像 | COPY ./start.sh /start.sh |
| ENTRYPOINT | 配置容器启动时运行的命令 | ENTRYPOINT /bin/bash -c '/start.sh' |
| VOLUME | 指定容器挂载点到宿主自动生成的目录或其它容器 | VOLUME ["/var/lib/mysql"] |
| USER | 为 RUN CMD和ENTRYPOINT执行命令指定运行用户 | USER zhufengjiagou |
| WORKDIR | 为RUN CMD ENTRYPOINT COPY ADD 设置工作目录 | WORKDIR /data |
| HEALTHCHECK | 健康检查 | HEALTHCHECK --interval=5m --timeout=3s --retries=3 CMS curl -f htp://localhost |
| ARG | 在构建镜像时指定一些参数 | ARG user |

* cmd给出的是一个容器的默认的可执行体。也就是容器启动以后，默认的执行的命令。重点就是这个"默认"。意味着，如果docker run没有指定任何的执行命令或者dockerfile里面也没有entrypoint，那么，就会使用cmd指定的默认的执行命令执行。同时也从侧面说明了entrypoint的含义，它才是真正的容器启动以后要执行命令

### **15.2 .dockerignore**

表示要排除，不要打包到image中的文件路径

.git

node\_modules

### **15.3 Dockerfile**

#### **15.3.1 安装node**

* [nvm](https://github.com/creationix/nvm/blob/master/README.md)

wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash

source /root/.bashrc

nvm install stable

node -v

npm i cnpm -g

npm i nrm -g

#### **15.3.2 安装express项目生成器**

npm install express-generator -g

express app

#### **15.3.3 Dockerfile**

FROM node

COPY ./app /app

WORKDIR /app

RUN npm install

EXPOSE 3000

* FROM 表示该镜像继承的镜像 :表示标签
* COPY 是将当前目录下的app目录下面的文件都拷贝到image里的/app目录中
* WORKDIR 指定工作路径，类似于执行 cd 命令
* RUN npm install 在/app目录下安装依赖，安装后的依赖也会打包到image目录中
* EXPOSE 暴露3000端口，允许外部连接这个端口

### **15.4 创建image**

docker build -t express-demo .

* -t用来指定image镜像的名称，后面还可以加冒号指定标签，如果不指定默认就是latest
* . 表示Dockerfile文件的所有路径,.就表示当前路径

### **15.5 使用新的镜像运行容器**

docker container run -p 3333:3000 -it express-demo /bin/bash

npm start

* -p 参数是将容器的3000端口映射为本机的3333端口
* -it 参数是将容器的shell容器映射为当前的shell,在本机容器中执行的命令都会发送到容器当中执行
* express-demo image的名称
* /bin/bash 容器启动后执行的第一个命令,这里是启动了bash容器以便执行脚本
* --rm 在容器终止运行后自动删除容器文件

### **15.6 CMD**

Dockerfile

+ CMD npm start

重新制作镜像

docker build -t express-demo .

* RUN命令在 image 文件的构建阶段执行，执行结果都会打包进入 image 文件；CMD命令则是在容器启动后执行
* 一个 Dockerfile 可以包含多个RUN命令，但是只能有一个CMD命令
* 指定了CMD命令以后，docker container run命令就不能附加命令了（比如前面的/bin/bash），否则它会覆盖CMD命令

### **15.7 发布image**

* [注册账户](https://hub.docker.com/)
* docker tag SOURCE\_IMAGE[:TAG] TARGET\_IMAGE[:TAG]

docker login

docker image tag [imageName] [username]/[repository]:[tag]

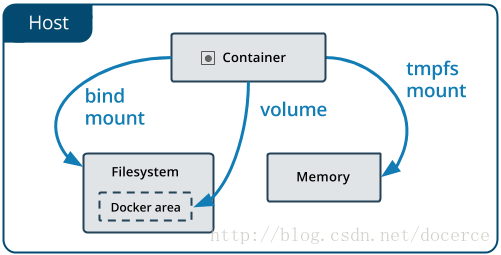
docker image build -t [username]/[repository]:[tag] .

docker tag express-demo zhangrenyang/express-demo:v1

docker push zhangrenyang/express-demo:v1

## **16. 数据盘**

* 删除容器的时候，容器层里创建的文件也会被删除掉，如果有些数据你想永久保存，比如Web服务器的日志，数据库管理系统中的数据，可以为容器创建一个数据盘



### **16.1 volume**

* volumes Docker管理宿主机文件系统的一部分(/var/lib/docker/volumes)
* 如果没有指定卷，则会自动创建
* 建议使用--mount ,更通用

### **16.1.1 创建数据卷**

docker volume --help

docker volume create nginx-vol

docker volume ls

docker volume inspect nginx-vol

#把nginx-vol数据卷挂载到/usr/share/nginx/html,挂载后容器内的文件会同步到数据卷中

docker run -d --name=nginx1 --mount src=nginx-vol,dst=/usr/share/nginx/html nginx

docker run -d --name=nginx2 -v nginx-vol:/usr/share/nginx/html -p 3000:80 nginx

### **16.1.2 删除数据卷**

docker container stop nginx1 停止容器

docker container rm nginx1 删除容器

docker volume rm nginx-vol 删除数据库

#### **16.1.3 管理数据盘**

docker volume ls #列出所有的数据盘

docker volume ls -f dangling=true #列出已经孤立的数据盘

docker volume rm xxxx #删除数据盘

docker volume ls #列出数据盘

### **16.2 Bind mounts**

* 此方式与Linux系统的mount方式很相似，即是会覆盖容器内已存在的目录或文件，但并不会改变容器内原有的文件，当umount后容器内原有的文件就会还原
* 创建容器的时候我们可以通过-v或--volumn给它指定一下数据盘
* bind mounts 可以存储在宿主机系统的任意位置
* 如果源文件/目录不存在，不会自动创建，会抛出一个错误
* 如果挂载目标在容器中非空目录，则该目录现有内容将被隐藏

#### **16.2.1 默认数据盘**

* -v 参数两种挂载数据方式都可以用

docker run -v /mnt:/mnt -it --name logs centos bash

cd /mnt

echo 1 > 1.txt

exit

docker inspect logs"Mounts": [

{

"Source":"/mnt/sda1/var/lib/docker/volumes/dea6a8b3aefafa907d883895bbf931a502a51959f83d63b7ece8d7814cf5d489/\_data",

"Destination": "/mnt",

}

]

* Source的值就是我们给容器指定的数据盘在主机上的位置
* Destination的值是这个数据盘在容器上的位置

#### **16.2.2 指定数据盘**

mkdir ~/data

docker run -v ~/data:/mnt -it --name logs2 centos bash

cd /mnt

echo 3 > 3.txt

exit

cat ~/data/3.txt

* ~/data:/mnt 把当前用户目录中的data目录映射到/mnt上

#### **16.2.3 指定数据盘容器**

* docker create [OPTIONS] IMAGE [COMMAND] [ARG...] 创建一个新的容器但不启动

docker create -v /mnt:/mnt --name logger centos

docker run --volumes-from logger --name logger3 -i -t centos bash

cd /mnt

touch logger3

docker run --volumes-from logger --name logger4 -i -t centos bash

cd /mnt

touch logger4

## **17. 网络**

* 安装Docker时，它会自动创建三个网络，bridge（创建容器默认连接到此网络）、 none 、host
  + None：该模式关闭了容器的网络功能,对外界完全隔离
  + 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。
  + bridge 桥接网络，此模式会为每一个容器分配IP
* 可以使用该--network标志来指定容器应连接到哪些网络

### **17.1 bridge(桥接)**

* bridge网络代表所有Docker安装中存在的网络
* 除非你使用该docker run --network=<NETWORK>选项指定，否则Docker守护程序默认将容器连接到此网络
* bridge模式使用 --net=bridge 指定，默认设置

docker network ls #列出当前的网络

docker inspect bridge #查看当前的桥连网络

docker run -d --name nginx1 nginx

docker run -d --name nginx2 --link nginx1 nginx

docker exec -it nginx2 bash

apt update

apt install -y inetutils-ping #ping

apt install -y dnsutils #nslookup

apt install -y net-tools #ifconfig

apt install -y iproute2 #ip

apt install -y curl #curl

cat /etc/hosts

ping nginx1

### **17.2 none**

* none模式使用--net=none指定

# --net 指定无网络

docker run -d --name nginx\_none --net none nginx

docker inspect none

docker exec -it nginx\_none bash

ip addr

### **17.3 host**

* host模式使用 --net=host 指定

docker run -d --name nginx\_host --net host nginx

docker inspect host

docker exec -it nginx\_host bash

ip addr

### **17.4 端口映射**

# 查看镜像里暴露出的端口号

docker image inspect nginx

"ExposedPorts": {"80/tcp": {}}

# 让宿主机的8080端口映射到docker容器的80端口

docker run -d --name port\_nginx -p 8080:80 nginx

# 查看主机绑定的端口

docker container port port\_nginx

### **17.5 指向主机的随机端口**

docker run -d --name random\_nginx --publish 80 nginx

docker port random\_nginx

docker run -d --name randomall\_nginx --publish-all nginx

docker run -d --name randomall\_nginx --P nginx

### **17.6 创建自定义网络**

* 可以创建多个网络，每个网络IP范围均不相同
* docker的自定义网络里面有一个DNS服务，可以通过容器名称访问主机

# 创建自定义网络

docker network create --driver bridge myweb

# 查看自定义网络中的主机

docker network inspect myweb

# 创建容器的时候指定网络

docker run -d --name mynginx1 --net myweb nginx

docker run -d --name mynginx2 --net myweb nginx

docker exec -it mynginx2 bash

ping mynginx1

### **17.7 连接到指定网络**

docker run -d --name mynginx3 nginx

docker network connect myweb mynginx3

docker network disconnect myweb mynginx3

### **17.8 移除网络**

docker network rm myweb

## **18.compose**

* Compose 通过一个配置文件来管理多个Docker容器
* 在配置文件中，所有的容器通过services来定义，然后使用docker-compose脚本来启动、停止和重启应用和应用中的服务以及所有依赖服务的容器
* 步骤：
  + 最后，运行docker-compose up，Compose 将启动并运行整个应用程序 配置文件组成
  + services 可以定义需要的服务，每个服务都有自己的名字、使用的镜像、挂载的数据卷所属的网络和依赖的其它服务
  + networks 是应用的网络，在它下面可以定义使用的网络名称，类性
  + volumes是数据卷，可以在此定义数据卷，然后挂载到不同的服务上面使用

### **18.1 安装compose**

yum -y install epel-release

yum -y install python-pip

yum clean all

pip install docker-compose

### **18.2 编写docker-compose.yml**

* 在 docker-compose.yml 中定义组成应用程序的服务，以便它们可以在隔离的环境中一起运行
* 空格缩进表示层次
* 冒号空格后面有空格

version: '2'services:

nginx1:

image: nginx

ports:

- "8080:80"

nginx2:

image: nginx

ports:

- "8081:80"

### **18.3 启动服务**

* docker会创建默认的网络

| **命令** | **服务** |
| --- | --- |
| docker-compose up | 启动所有的服务 |
| docker-compose up -d | 后台启动所有的服务 |
| docker-compose ps | 打印所有的容器 |
| docker-compose stop | 停止所有服务 |
| docker-compose logs -f | 持续跟踪日志 |
| docker-compose exec nginx1 bash | 进入nginx1服务系统 |
| docker-compose rm nginx1 | 删除服务容器 |
| docker network ls | 查看网络网络不会删除 |
| docker-compose down | 删除所有的网络和容器 |

删除所有的容器 docker container rm docker container ps -a -q

### **18.4 网络互ping**

docker-compose up -d

docker-compose exec nginx1 bash

apt update && apt install -y inetutils-ping

#可以通过服务的名字连接到对方

ping nginx2

### **18.5 配置数据卷**

* networks 指定自定义网络
* volumes 指定数据卷
* 数据卷在宿主机的位置 /var/lib/docker/volumes/nginx-compose\_data/\_data

version: '3'services:

nginx1:

image: nginx

ports:

- "8081:80"

networks:

- "newweb"

volumes:

- "data:/data"

- "./nginx1:/usr/share/nginx/html"

nginx2:

image: nginx

ports:

- "8082:80"

networks:

- "default"

volumes:

- "data:/data"

- "./nginx2:/usr/share/nginx/html"

nginx3:

image: nginx

ports:

- "8083:80"

networks:

- "default"

- "newweb"

volumes:

- "data:/data"

- "./nginx3:/usr/share/nginx/html"networks:

newweb:

driver: bridgevolumes:

data:

driver: local

docker exec nginx-compose\_nginx1\_1 bash

cd /data

touch 1.txt

exit

cd /**var**/lib/docker/volumes/nginx-compose\_data/\_data

ls

## **19. node项目**

* nodeapp 是一个用 Docker 搭建的本地 Node.js 应用开发与运行环境。

### **19.1 服务分类**

* db：使用 mariadb 作为应用的数据库
* node：启动node服务
* web：使用 nginx 作为应用的 web 服务器

### **19.2 app目录结构**

| **文件** | **说明** |
| --- | --- |
| docker-compose.yml | 定义本地开发环境需要的服务 |
| images/nginx/config/default.conf | nginx 配置文件 |
| images/node/Dockerfile | node的Dockfile配置文件 |
| images/node/web/package.json | 项目文件 |
| images/node/web/public/index.html | 静态首页 |
| images/node/web/server.js | node服务 |

├── docker-compose.yml

└── images

├── nginx

│ └── config

│ └── **default**.conf

└── node

├── Dockerfile

└── web

├── package.json

├── public

│ └── index.html

└── server.js

#### **19.2.1 docker-compose.yml**

version: '2'services:

node:

build:

context: ./images/node

dockerfile: Dockerfile

depends\_on:

- db

web:

image: nginx

ports:

- "8080:80"

volumes:

- ./images/nginx/config:/etc/nginx/conf.d

- ./images/node/web/public:/public

depends\_on:

- node

db:

image: mariadb

environment:

MYSQL\_ROOT\_PASSWORD: "root"

MYSQL\_DATABASE: "node"

MYSQL\_USER: "zfpx"

MYSQL\_PASSWORD: "123456"

volumes:

- db:/var/lib/mysqlvolumes:

db:

driver: local

#### **19.2.2 server.js**

images/node/web/server.js

**let** http=require('http');**var** mysql = require('mysql');**var** connection = mysql.createConnection({

host : 'db',

user : 'zfpx',

password : '123456',

database : 'node'

});

connection.connect();

**let** server=http.createServer(**function** (req,res) {

connection.query('SELECT 2 + 2 AS solution', **function** (error, results, fields) {

**if** (error) **throw** error;

res.end(''+results[0].solution);

});

});

server.listen(3000);

#### **19.2.3 package.json**

images/node/web/package.json

{"scripts": {

"start": "node server.js"

},"dependencies": {

"mysql": "^2.16.0"

}

#### **19.2.4 images/node/Dockerfile**

FROM node

MAINTAINER zhangrenyang <zhang\_renyang@126.com>

COPY ./web /web

WORKDIR /web

RUN npm install

CMD npm start

#### **19.2.5 images/nginx/config/default.conf**

upstream backend {

server node:3000;

}

server {

listen 80;

server\_name localhost;

root /public;

index index.html index.htm;

location /api {

proxy\_pass http://backend;

}

}

## **20. 搭建LNMP网站**

### **20.1 关闭防火墙**

| **功能** | **命令** |
| --- | --- |
| 停止防火墙 | systemctl stop firewalld.service |
| 永久关闭防火墙 | systemctl disable firewalld.service |

# 关闭防火墙之后docker需要重启

/bin/systemctl restart docker.service

### **20.2 创建自定义网络**

docker network create lnmp

### **20.3 创建mysql数据库容器**

docker run -itd --name lnmp\_mysql --net lnmp -p 3306:3306 --mount src=mysql-vol,dst=/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=123456 mysql:5.6 --character-**set**-server=utf8

### **20.4 创建数据库**

docker exec lnmp\_mysql bash -c 'exec mysql -uroot -p"$MYSQL\_ROOT\_PASSWORD" -e"create database wordpress"'

mysql -h127.0.0.1 -uroot -p123456

### **20.5 创建nginx+PHP容器**

mkdir -p /app/wwwroot

docker run -itd --name lnmp\_web --net lnmp -p 8888:80 --mount type=bind,src=/app/wwwroot,dst=/var/www/html richarvey/nginx-php-fpm

### **20.6 wordpress**

cd /usr/local/src

wget https://wordpress.org/latest.tar.gz

tar -xzvf latest.tar.gz -C /app/wwwroothttp://192.168.20.136:8888/wordpress/wp-admin/setup-config.php

手工添加 /app/wwwroot/wordpress/wp-config.php

## **21. 制作个性化镜像**

### **21.1 搭建nginx镜像**

#### **21.1.1 下载nginx**

mkdir /usr/local/src

cd /usr/local/src

wget http://nginx.org/download/nginx-1.12.1.tar.gz

#### **21.1.2 重启docker**

#如果容器内无法联网可以重启docker

systemctl restart docker.service

#### **21.1.3 编写default.conf**

server {

listen 80;

server\_name localhost;

#charset koi8-r;

#access\_log /var/log/nginx/host.access.log main;

location / {

root /usr/local/nginx/html;

index index.html index.htm;

}

location /status{

stub\_status on;

}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000

location ~ \.php$ {

root /usr/local/nginx/html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

include fastcgi\_params;

}

}

#### **21.1.4 编写Dockerfile**

Dockerfile

FROM centos

MAINTAINER zhufengjiagou

RUN yum install -y gcc gcc-c++ make openssl-devel pcre-devel

ADD nginx-1.12.1.tar.gz /tmp

RUN cd /tmp/nginx-1.12.1 && \

./configure --prefix=/usr/local/nginx && \

make -j 2 && \

make install

RUN rm -rf /tmp/nginx-1.12.1 && yum clean all

COPY **default**.conf /usr/local/nginx/conf

WORKDIR /usr/local/nginx

EXPOSE 80

CMD ["./sbin/nginx","-g","daemon off;"]

#### **21.1.5 打包镜像**

docker image build -t zfnginx:v1 -f Dockerfile .

#### **21.1.6 使用镜像**

docker container run -it zfnginx:v1 bash

### **21.2 搭建php镜像**

#### **21.2.1 下载php**

cd /usr/local/src

wget http://img.zhufengpeixun.cn/libmcrypt-2.5.8.tar.gz

wget http://img.zhufengpeixun.cn/php-5.6.31.tar.gz

#### **21.2.2 Dockerfile**

FROM centos

MAINTAINER zhufengjiagou

RUN yum -y install gcc gcc-c++ make automake autoconf libtool openssl-devel pcre-devel libxml2 libxml2-devel bzip2 bzip2-devel libjpeg-turbo libjpeg-turbo-devel libpng libpng-devel freetype freetype-devel zlib zlib-devel libcurl libcurl-devel

ADD libmcrypt-2.5.8.tar.gz /tmp

RUN cd /tmp/libmcrypt-2.5.8 && \

./configure && \

make -j 2 && \

make install

ADD php-5.6.31.tar.gz /tmp

RUN cd /tmp/php-5.6.31 && \

./configure --prefix=/usr/local/php --**with**-pdo-mysql=mysqlnd --**with**-mysqli=mysqlnd --**with**-mysql=mysqlnd --**with**-openssl --enable-mbstring --**with**-freetype-dir --**with**-jpeg-dir --**with**-png-dir --**with**-mcrypt --**with**-zlib --**with**-libxml-dir=/usr --enable-xml --enable-sockets --enable-fpm --with-config-file-path=/usr/local/php/etc --**with**-bz2 --**with**-gd && \

make -j 2 && \

make install

RUN cp /usr/local/php/etc/php-fpm.conf.default /usr/local/php/etc/php-fpm.conf

RUN sed -i 's/127.0.0.1/0.0.0.0/g' /usr/local/php/etc/php-fpm.conf

RUN sed -i '89a daemonize = no' /usr/local/php/etc/php-fpm.conf

RUN rm -rf /tmp/php-5.6.30 && yum clean all

WORKDIR /usr/local/php

EXPOSE 9000

CMD ["/usr/local/php/sbin/php-fpm","-c","/usr/local/php/etc/php-fpm.conf"]

#### **21.1.3 打包镜像**

docker image build -t zfphp:v1 -f Dockerfile .

#### **21.1.4 使用镜像**

docker container run -it zfphp:v1 bash

## **22. 布署Java网站**

### **22.1 下载安装包**

* [java](https://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase10-4425482.html)
* [tomcat](http://tomcat.apache.org/download-70.cgi)

cd /usr/local/src/java

wget http://img.zhufengpeixun.cn/jdk1.8.0\_211.tar.gz

wget http://img.zhufengpeixun.cn/apache-tomcat-7.0.94.tar.gz

### **22.2 镜像文件**

Dockerfile

FROM centos

MAINTAINER zhufengjiagou

ADD jdk1.8.0\_211.tar.gz /usr/local

ENV JAVA\_HOME /usr/local/jdk1.8.0\_211

ADD apache-tomcat-7.0.94.tar.gz /usr/local

RUN rm -f /usr/local/\*.tar.gz

WORKDIR /usr/local/apache-tomcat-7.0.94

EXPOSE 8080

ENTRYPOINT ["./bin/catalina.sh","run"]

### **22.3 打包镜像**

docker image build -t zftomcat:v1 -f Dockerfile .

### **22.4 启动容器**

docker run -itd \

--name=tomcat \

-p 8080:8080 \

--mount type=bind,src=/app/webapps,dst=/usr/local/apache-tomcat-7.0.94/webapps \zftomcat:v1