link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=249 sentences=229, words=2105

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

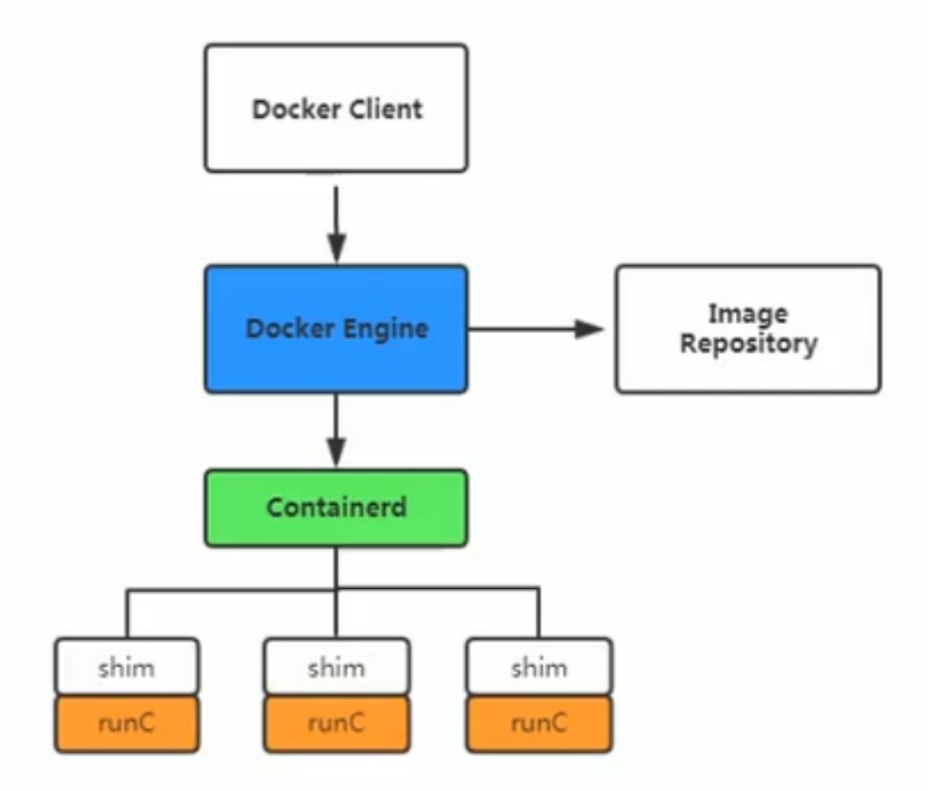
- 启动时间
    - Docker秒级启动
    - KVM分钟级启动
- 轻量级 容器镜像通常以M为单位，虚拟机以G为单位，容器资源占用小，要比虚拟要部署更快速
    - 容器共享宿主机内核，系统级虚拟化，占用资源少，容器性能基本接近物理机
    - 虚拟机需要虚拟化一些设备，具有完整的OS,虚拟机开销大，因而降低性能，没有容器性能好
- 安全性
    - 由于共享宿主机内核，只是进程隔离，因此隔离性和稳定性不如虚拟机，容器具有一定权限访问宿主机内核，存在一下安全隐患
- 使用要求
    - KVM基于硬件的完全虚拟化，需要硬件CPU虚拟化技术支持
    - 容器共享宿主机内核，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-config-manager --enable docker-ce-nightly #要每日构建版本的 Docker CE
yum-config-manager --enable docker-ce-test
yum install docker-ce docker-ce-cli containerd.io
```

9.2 启动

```
systemctl start docker
```

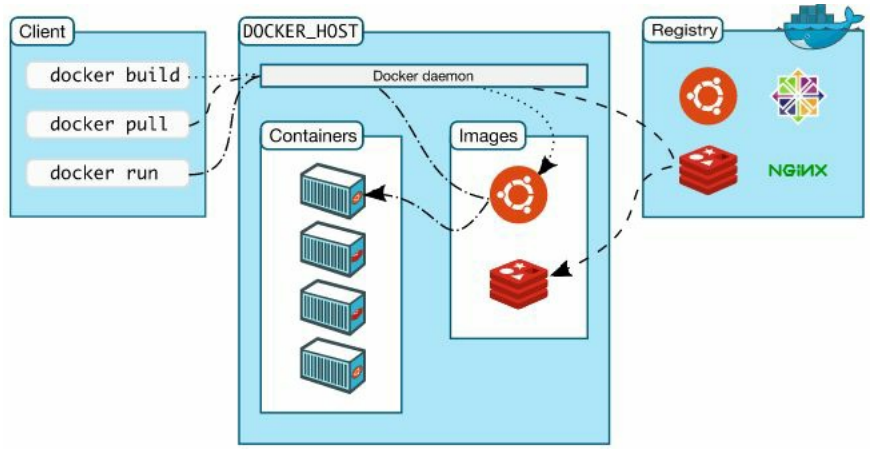
9.3 查看docker版本

```
$ 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
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></id>` 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 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，没有被标记或补任何容器引用 docker image prune tag 标记本地镜像，将其归入某一仓库 docker image tag [imageName] [username]/[repository]:[tag] export 导出容器文件系统tar归档文件创建镜像 docker export -o mysqlv1.tar a404c6c174a2 import 导入容器快照文件系统tar归档文件创建镜像 docker import mysqlv1.tar zf/mysql:v2 save 保存一个或多个镜像到一个tar归档文件 docker save -o mysqlv2.tar zf/mysqlv2:v3 load 加载镜像存储文件来自tar归档或标准输入 docker load -i mysqlv2.tar build 根据Dockerfile构建镜像

> 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），而镜像存储文件将保存完整记录，体积也要大。此外，从容器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 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 显示一个容器运行的进程 docker container update port 更新一个或多个容器配置 docker container port 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] 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 [containerID] /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 [containID]:app/package.json . commit [containerId] 创建一个新镜像来自一个容器 docker commit -a "zhufeng" -m "mysql" a404c6c174a2 mynginx:v1

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

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 -p --publish list 发布容器端口到主机 -P --publish-all --mount mount 挂载宿主机分区到容器 -v,--volumn list 挂载宿主机分区到容器

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. 制作个性化镜像

```
docker commit -m"hello" -a "zhangrenyang" [containerId] zhangrenyang/hello:latest
docker images
docker run zhangrenyang/hello /bin/bash
docker rm b2839066c362
docker 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

命令 含义 案例 FROM 继承的镜像 FROM node COPY 拷贝 COPY ./app /app WORKDIR 指定工作路径 WORKDIR /app RUN 编译打包阶段运行命令 RUN npm install EXPOSE 暴露端口 EXPOSE 3000 CMD 容器运行阶段运行命令 CMD npm run start

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:
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

```
CMD npm start
```

- 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

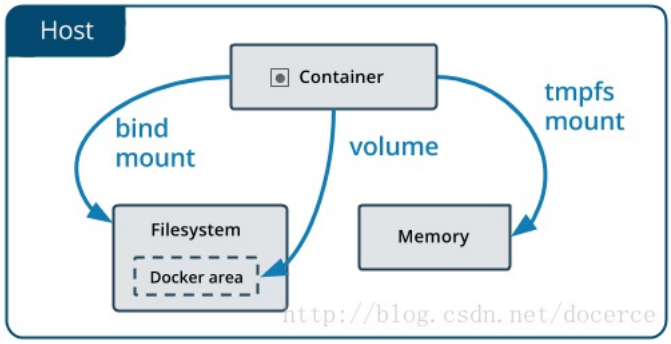
- 注册账户 (https://hub.docker.com/)
- docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

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

docker tag express-demo zhangrenyang/express-demo:1.0.0
docker push zhangrenyang/express-demo:1.0.0
```

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
docker run -d -it --name=nginx1 --mount src=nginx-vol,dst=/usr/share/nginx/html nginx
docker run -d -it --name=nginx2 -v nginx-vol:/usr/share/nginx/html 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 -ti --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 -v /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

docker里面有一个DNS服务，可以通过容器名称访问主机 网络类型

- none 无网络，对外界完全隔离
- host 主机网络
- bridge 桥接网络，默认网络

17.1 bridge

```
docker network ls
docker inspect bridge
docker run -d --name server1 nginx
docker run -d --name server2 nginx
docker exec -it server1 bash
ping server2
```

17.2 none

```
docker run -d --name server_none --net none nginx
docker inspect none
docker exec -it server_none bash
ip addr
```

17.3 host

```
docker run -d --name server_host --net host nginx
docker inspect none
docker exec -it server_host bash
ip addr
```

17.4 访问桥接网络里面的服务

```
docker inspect nginx
443 80
docker run -d --name server_nginx -p "8080:80"  nginx
```

- 访问主机的8080端口会被定向到容器的80端口

17.5 查看主机绑定的端口

```
docker inspect [容器名称]
docker port server_nginx
```

17.6 指向主机的随机端口

```
docker run -d --name webserver --publish 80 nginx
docker port webserver

docker run -d --name webserver --publish-all nginx
docker run -d --name webserver --P nginx
`
```

17.7 创建自定义网络

```
docker network create --driver bridge web
docker network inspect web
```

17.8 制定网络

```
docker run -d --name webserver  --net web nginx
docker network connect  web webserver1
docker network disconnect web webserver2
```

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，非常适合组合使用多个容器进行开发的场景 步骤：
- 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。
- 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。

```
pip install docker-compose
```

18.1 docker-compose.yml

- 空格缩进表示层次
- 冒号空格后面有空格

```
version: '2'
services:
  zfpx1:
    image: nginx
    port:
      - "8080:80"
  zfpx2:
    image: nginx
    port:
      - "8081:80"
```

18.2 nginx工具包

```
apt update
#ping
apt install inetutils-ping
#nslookup
apt install dnsutils
#ifconfig
apt install net-tools
#ip
apt install iproute2
#curl
apt install curl
```

18.3 启动

```
docker-compose up 启动所有的服务
docker-compose -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 zfpx1 bash 进入zfpx服务系统
docker-compose rm 删除服务容器
docker network ls 网络不会删除
docker-compose down 删除网路
```

- docker会创建默认的网络

18.4 配置网络

```
docker-compose up -d
docker-compose exec zfpx1 bash
ping zfpx2 可以通过服务的名字连接到对方
```

18.5 配置数据卷

```
version: '2'
services:
  zfpx1:
    image: nginx
    ports:
      - "8080:80"
    networks:
      - "zfpx"
    volumes:
      - "access:/mnt"
  zfpx2:
    image: nginx
    ports:
      - "8081:80"
    networks:
      - "zfpx"
    volumes:
      - "access:/mnt"
  zfpx3:
    image: nginx
    ports:
      - "8082:80"
    networks:
      - "default"
      - "zfpx"
networks:
  zfpx:
    driver: bridge
volumes:
  access:
    driver: local
```

18.6 配置根目录

```
version: '2'
services:
  zfpx1:
    image: nginx
    ports:
      - "8080:80"
    networks:
      - "zfpx"
    volumes:
      - "access:/mnt"
      - "./zfpx1:/usr/share/nginx/html"
  zfpx2:
    image: nginx
    ports:
      - "8081:80"
    networks:
      - "zfpx"
    volumes:
      - "access:/mnt"
      - "./zfpx2:/usr/share/nginx/html"
  zfpx3:
    image: nginx
    ports:
      - "8082:80"
    networks:
      - "default"
      - "zfpx"
networks:
  zfpx:
    driver: bridge
volumes:
  access:
    driver: local
```

19. nodeapp

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

- 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 结构

- app：这个目录存储应用
    - web 放应用的代码
- services： 环境里定义的服务需要的一些服务
- images: 方式一些贬义的脚本和镜像
- docker-compose.yml：定义本地开发环境需要的服务
- images/nginx/config/default.conf 放置nginx 配置文件
- node 的Dockfile配置文件

19.3 docker-compose.yml

```
version: '2'
services:
 node:
  build:
    context: ./images/node
    dockerfile: Dockerfile
  volumes:
    - ./app/web:/web
  depends_on:
   - db
 web:
  image: nginx
  ports:
   - "8080:80"
  volumes:
   - ./images/nginx/config:/etc/nginx/conf.d
   - ./app/web/views:/mnt/views
  depends_on:
   - node
 db:
  image: mariadb
  environment:
   MYSQL_ROOT_PASSWORD: "root"
   MYSQL_DATABASE: "node"
   MYSQL_USER: "zfpx"
   MYSQL_PASSWORD: "123456"
  volumes:
    - db:/var/lib/mysql
volumes:
 db:
  driver: local
```

19.4 其它文件

19.4.1 app/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);
```

19.4.2 package.json

```
"scripts": {
    "start": "node server.js"
  },
"dependencies": {
    "mysql": "^2.16.0"
}
```

19.4.3 images/node/Dockerfile

```
FROM node
MAINTAINER zhangrenyang 126.com>
WORKDIR /web
RUN npm install
CMD npm start
```

19.4.4 images/nginx/config/default.conf

```
upstream backend {
    server node:3000;
}
server {
    listen 80;
    server_name localhost;
    root /mnt/views;
    index index.html index.htm;

    location /api {
        proxy_pass http:
    }
}
```

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

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
 yum install -y mysql
```

20.4 创建所需数据库

```
docker exec lnmp_mysql bash -c 'exec mysql -uroot  -p"$MYSQL_ROOT_PASSWORD" -e"create database wordpress"'
```

20.5 创建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 /opt/src
wget https:
tar -xzvf latest.tar.gz -C /app/wwwroot
http:
http:
```

21. Dockerfile

21.1 语法

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 (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 http://localhost exit 1 ARG 在构建镜像时指定一些参数 ARG user

21.2 build镜像命令

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

21.3 搭建nginx镜像

```
mkdir /usr/local/src
cd /usr/local/src
wget http://nginx.org/download/nginx-1.12.1.tar.gz
#如果容器内无法联网可以重启docker
systemctl restart docker
```

```
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 nginx.conf /usr/local/nginx/conf
WORKDIR /usr/local/nginx
EXPOSE 80
CMD ["./sbin/nginx","-g","daemon off;"]
```

```
docker image build -t nginx:v1 -f Dockerfile .
```

21.4 搭建php镜像

```
wget http:
wget http:
```

```
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-5.6.30.tar.gz /tmp
RUN cd /tmp/php-5.6.30 && \
    ./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"]
```

```
docker image build -t php:v1 -f Dockerfile .
```

22. 布署PHP网站

22.1 自定义网络

```
docker network create lnmp
```

22.2 创建PHP容器

```
docker run -itd \
--name lnmp_php
--net lnmp \
--mount type=bind,src=/app/wwwroot,dst=/usr/local/nginx/html \
php:v1
```

22.3 创建nginx容器

```
docker run -itd \
--name lnmp_nginx
--net lnmp \
-p 8888:80 \
--mount type=bind,src=/app/wwwroot,dst=/usr/local/nginx/html \
nginx:v1
```

22.4 创建mysql容器

```
docker run -itd \
--name lnmp_mysql
--net lnmp \
--mount type=bind,src=mysql-vol,dst=/usr/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456
mysql --character-set-server=utf8
```

23. 布署Java网站

```
wget  https:

wget http:
```

tomcat (http://tomcat.apache.org/download-70.cgi)

```
FROM centos
MAINTAINER zhufengjiagou
ADD  jdk-7u80-linux-x64.tar.gz /usr/local
ENV JAVA_HOME /usr/local/jdk1.7.0_80
ADD apache-tomcat-7.0.94.tar.gz /usr/local
COPY server.xml /user/local/apache/conf
RUN rm -f /usr/local
```

```
docker image build -t tomcat:v1 -f Dockerfile .
```

```
docker run -itd \
--name=tomcat
-p 8080:8080
--mount type=bind,src=/app/webapps,dst=/usr/local/apache-tomcat-7.0.94/webapps \
tomcat:v1
```

23. 发布

注册账号 https://hub.docker.com (https://hub.docker.com)

登录Docker Hub docker login

docker login --username=zhangrenyang --password=123456

镜像打标签 docker tag wordpress:v1 zhangrenyang/wordpress:v1

上传 docker push zhangrenyang/wordpress:v1

搜索测试 docker search zhangrenyang

下载 docker pull zhangrenyang/wordpress:v1

20. 参考

- yaml (http://www.ruanyifeng.com/blog/2016/07/yaml.html)
- mysql (https://www.npmjs.com/package/mysql)