Docker容器化技术(下)

一、Dockerfile基础命令

1.1.FROM - 基于基准镜像

- FROM centos #制作基准镜像(基于centos)
- FROM scratch #不依赖任何基准镜像base image
- FROM tomcat:9.022-jdk8-openjdk
- 尽量使用官方的Base Image

1.2.LABEL&MAINTAINER - 说明信息

- MAINTAINER xxx.com
- LABEL version = "1.0"
- LABEL description = "xxx啥作用"

1.3.WORKDIR - 设置工作目录

- WORKDIR /usr/local
- WORKDIR /usr/local/newdir #自动创建
- 尽量使用绝对路径

1.4.ADD&COPY - 复制文件

- ADD hello / #f复制到根路径
- ADD test.tar.gz / #添加根目录并解压
- ADD 除了复制，还具备添加远程文件的功能，+网址，类似wget

1.5.ENV - 设置环境常量

- ENV JAVA_HOME /usr/local/openjdk8
- RUN ${JAVA_HOME}/bin/java -jar test.jar
- 尽量使用环境常量，可提高程序维护性

二、Dockerfile执行指令

RUN&CMD&ENTRYPOINT

- RUN：在Build构建时执行 相当于shell的执行方式
- ENTRYPOINT：容器启动时执行
- CMD：容器启动后执行 CMD[“ps”,"ef"] 相当于exec的执行方式

执行方式

```
RUN yum install -y vim #shell命令格式
RUN ["yum","install","-y","vim"] #Exec命令格式
```

为什么要提供两种不同的执行方式呢?

- shell执行

使用Shell执行，当前shell是父进程，生成一个子进程

在子shell中执行脚本，脚本执行完毕，退出子shell，回到当前shell

- exec运行

当前进程执行

实战

```
FROM centos
RUN ["echo","image building!!!"]//执行在子进程了
CMD ["echo","container starting..."]//只有这句能看到的
```

注意 : CMD 如果增加了，则会取代CMD命令，CMD命令不一定执行

ENTRYPOINT 一定会执行

三、构建Redis镜像

- Redis是一个NoSQL数据库
- 2010.3.15开始，Redis开发工作，由VMWare主持

书写Dockerfile

```
FROM centos
RUN ["yum","install","-y","gcc","gcc-c++","net-tools","make"]
WORKDIR /usr/local
ADD redis-4.0.14.tar.gz .   //会自动解压
WORKDIR /usr/local/redis-4.0.14/src
RUN make && make install
WORKDIR /usr/local/redis-4.0.14
ADD redis-7000.conf .
EXPOSE 7000 //暴露7000端口
CMD ["redis-server","redis-7000.conf"]
```

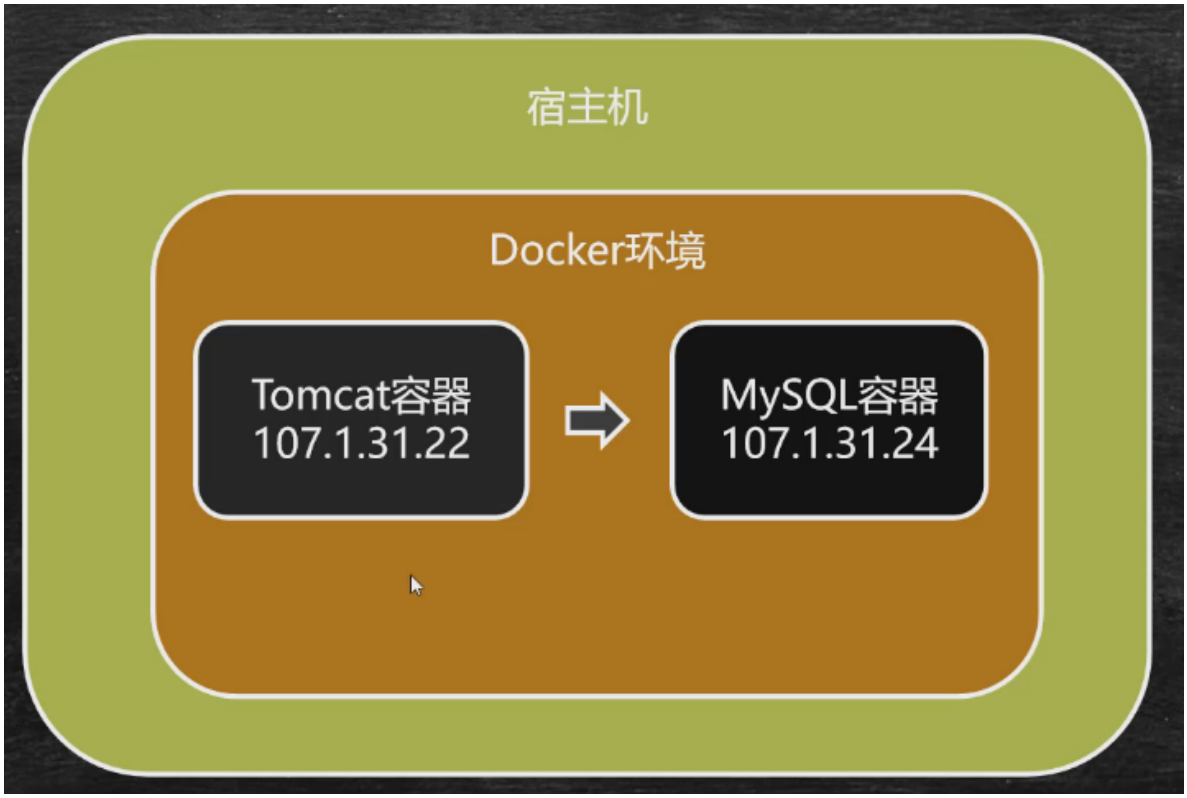
之后执行镜像构建就行

```
docker build -t xxx/docker-redis:1.0 .
docker run -p 7000:7000 xxx/docker-redis:1.0
```

这边只是讲述Dockerfile的书写，实际工作中，直接使用redis镜像即可

四、容器间Link单向通信

- 容器创建后，存在一个虚拟IP



容器单向访问

原理：虽然有内虚拟IP，容器中进行通讯，我们不采用IP通讯，采用容器名称进行通讯

我们使用--name指定名称

```
docker run -d --name web tomcat
docker run -d --name database -it centos /bin/bash
```

查看虚拟IP

```
docker inspect [containerID]
```

我们使用ping

```
ping 172.17.0.3 是可以ping通的
但是我们ping名称是ping不通的
```

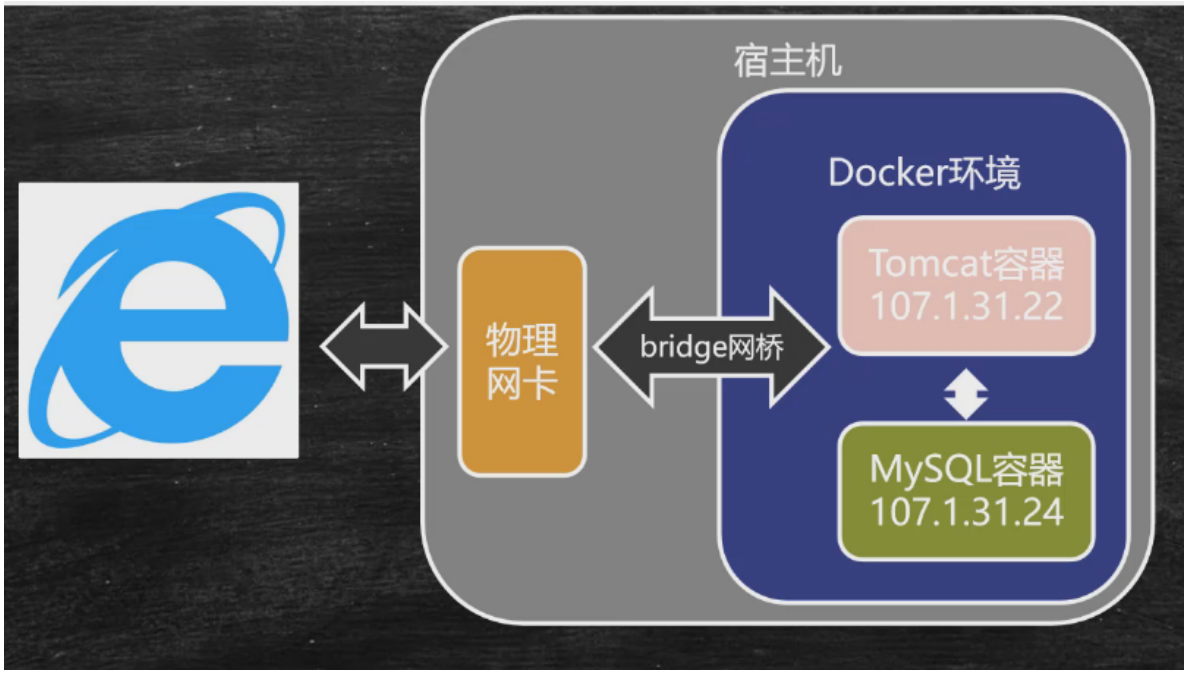
所以我们创建第二个tomcat的时候，要进行链接数据库

```
docker run --name web --link database tomcat
```

这时候我们进入tomcat，然后进行ping，可以自然联通

```
ping database 可以自动ping通
```

五、基于Bridge网桥进行双向通信

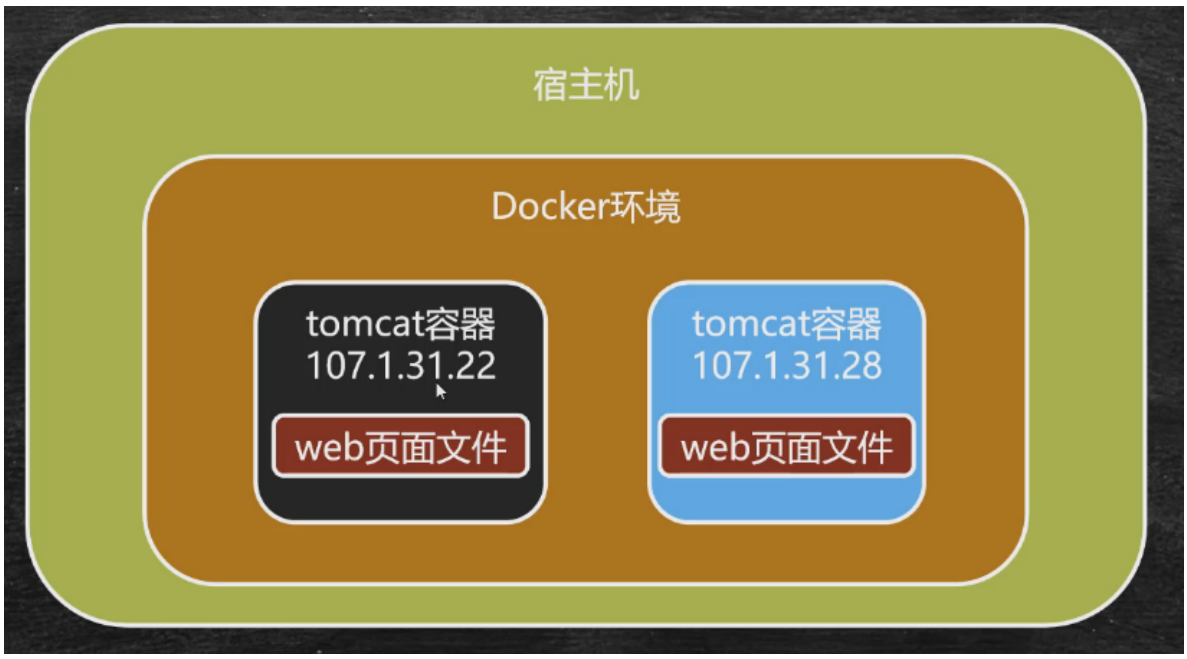


网桥双向通信原理

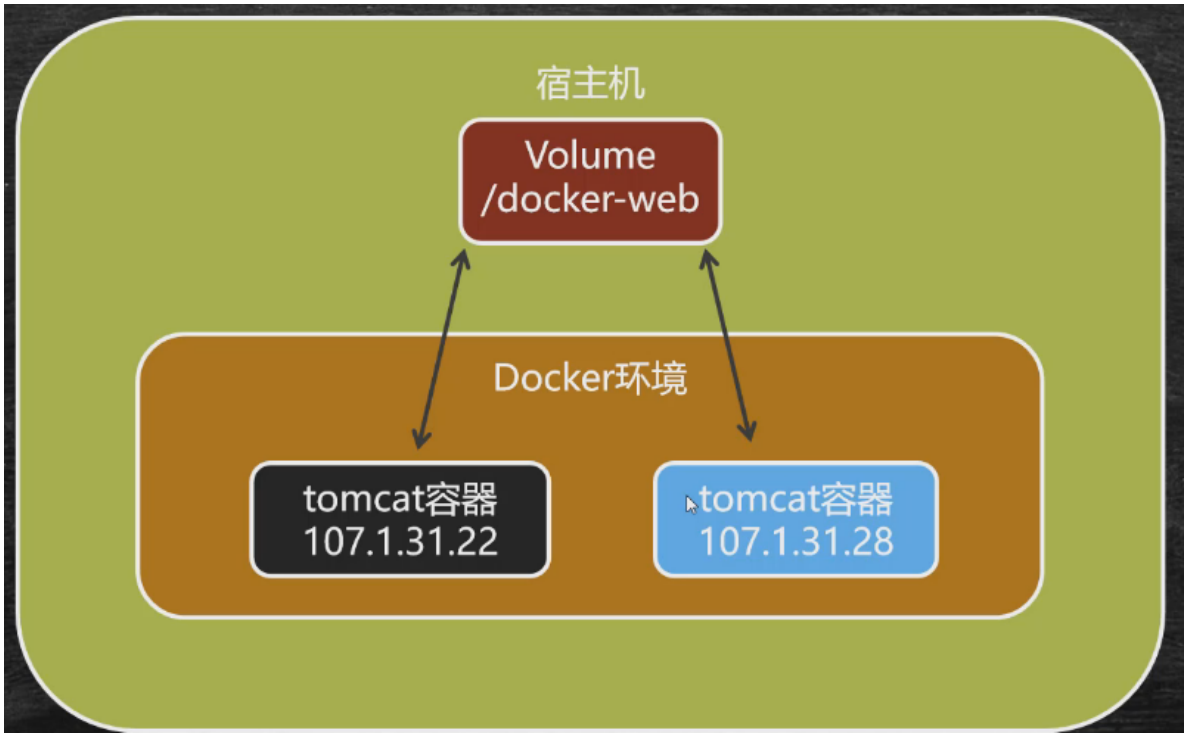
具体操作：绑定tomcat和database

```
docker run -d --name web tomcat
docker run -d --name database centos /bin/bash
docker network ls
docker network create -d bridge my-bridge
docker network connect my-bridge web
docker network connect my-bridge database
```

六、Volume容器间共享数据



容器未使用volume



Volume容器原理

方法：

1.通过设置-v挂载宿主机目录

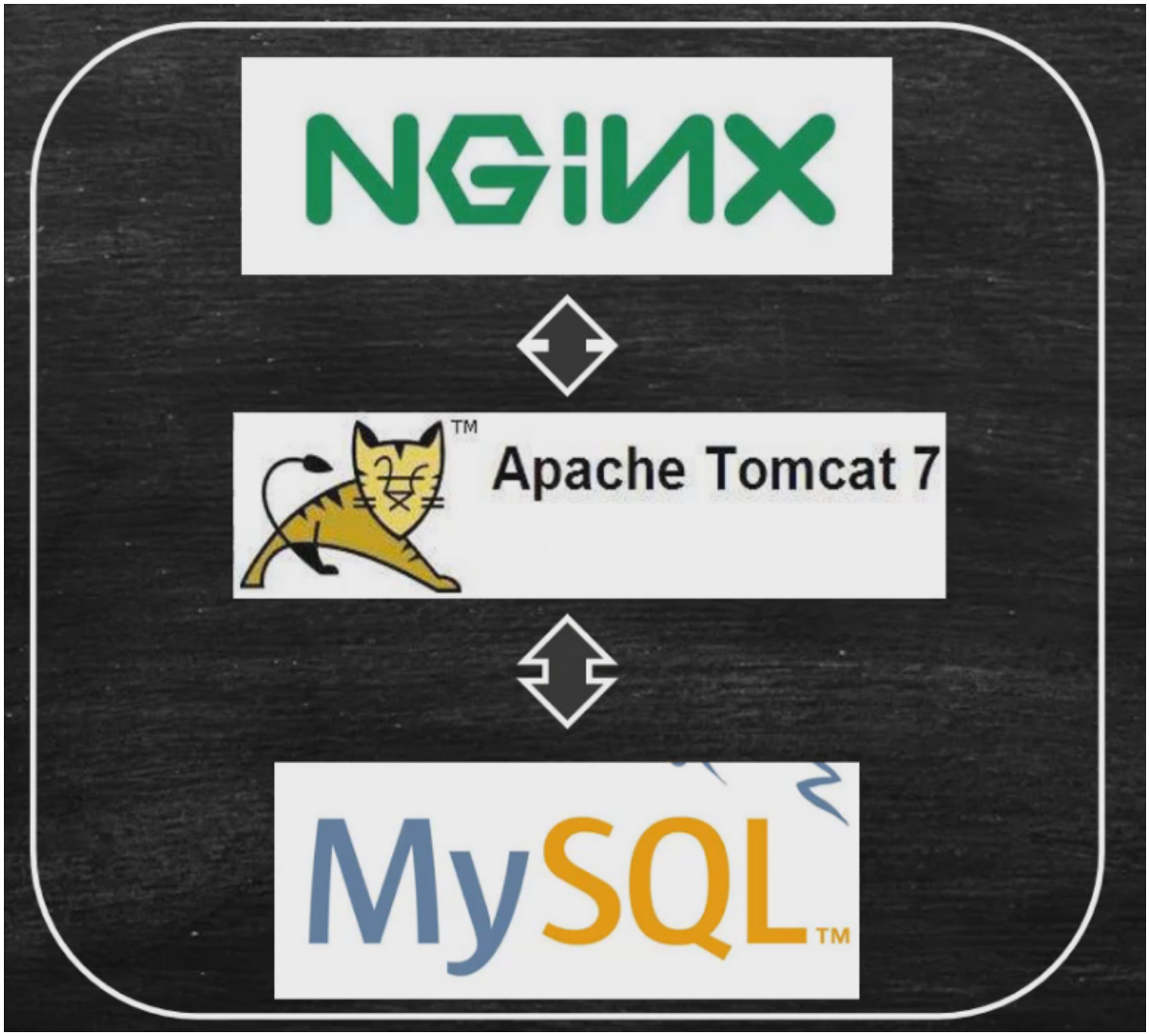
- 格式：
- docker run --name 容器名 -v 宿主机路径:容器内挂载路径 镜像名
- 实例：
- docker run --name t1 -v /usr/webapps:/usr/local/tomcat/webapps tomcat

2.通过共享容器

- 格式：
- docker create --name webpage -v /webapps:/tomcat/webapps tomcat /bin/true
- 共享容器挂载点
- docker run --volumes-from webpage --name t1 -d tomcat

七、Docker Compose

多容器部署会遇到很多麻烦，所以我们的Docker compose出来了



案例

- Docker Compose 单机多容器部署工具
- 通过yml文件定义多容器如何部署
- WIN/MAC默认提供Docker Compose，Linux则需要安装

安装步骤：

1.获取自动安装

```
pip install -U docker-compose==1.23.2
```

2.进行执行权限

```
sudo chmod +x /usr/local/bin/docker-compose
```

安装WordPress

1.创建目录wordpress

```
mkdir wordpress
```

2.vim docker-compose.yml

```
复制官网上的
```

3.build the project

```
docker-compose up -d
```

八、Docker-compose应用实战

案例、两个SpringBoot项目构建docker-compose

SpringBoot打包

文件 applicaion-dev.yml application.yml bsbdj.jar

```
vim Dockerfile

FROM openjdk:8u222-jre
WORKDIR /usr/local/bsbdj   //上述肯定没有这个目录，所以会创建
ADD bdbdj.jar    //加入jar
ADD application.yml .
ADD application-dev.yml .
EXPOSE 80        //暴露端口80
CMD ["java","-jar","bsbdj.jar"]

docker build -t msb.com/bsbdj-app .
docker run msb.com/bsbdj-app
```

数据库打包

```
vim Dockerfile

FROM mysql:5.7
WORKDIR /docker-entrypoint-initdb.d
ADD init-db.sql .

docker build -t msb.com/bsbdj-db .
docker run -e -d MYSQL_ROOT_PASSWORD=root msb.com/bsbdj-db

docker exec -it aae73fa77d75 /bin/bash
```

Docker-Compose进行关联和发布

```
vim docker-compose.yml

version: '3.3'
services:
    db:
        build: ./bsbdj-db/
        restart: always  //容错，自动重启
        environment:
            MYSQL_ROOT_PASSWORD: root
    app:
        build: ./bsbdj-app/
        depends_on:
            - db
```

```
        ports:
            - "80:80"
        restart: always


docker-compose up
docker-compose up -d
docker-compose logs
docker-compose down
```

我们连接数据库，那么yml中jdbc:mysql://db:3306/xxx即可