第一章：Docker容器的介绍与整套课程大纲

1.1 课程大纲总览

简介：介绍课程知识目录大纲

1.2 Docker容器化技术的介绍和使用场景

简介：介绍docker容器化技术

- 什么是Dokcer？
  - 百科:一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口；
    使用go语言编写，在LCX（linux容器）基础上进行的封装
    简单来说分三点：
    1）就是可以快速部署启动应用
    2）实现虚拟化，完整资源隔离
    3）一次编写，四处运行（有一定的限制，比如Docker是基于Linux 64bit的，无法在32bit的linux/Windows/unix环境下使用）
- 为什么要用？
  - 1、提供一次性的环境，假如需要安装Mysql，则需要安装很多依赖库、版本等，如果使用Docker则通过镜像就可以直接启动运行

    2、快速动态扩容，使用docker部署了一个应用，可以制作成镜像，然后通过Dokcer快速启动

    3、组建微服务架构，可以在一个机器上模拟出多个微服务，启动多个应用

    4、更好的资源隔离和共享
- 总结一句话：开箱即用，快速部署，可移植性强，环境隔离

第二章：Docker容器基础入门实战

2.1 Window 10 环境下安装Docker

简介：如何在Win10 环境下安装docker容器

- Docker for Win10 安装包下载地址：
  - https://store.docker.com/editions/community/docker-ce-desktop-windows
- 国内镜像加速
  - https://registry.docker-cn.com　#Docker中国区
  - http://hub-mirror.c.163.com　　#网易

2.2 Linux Centos7环境下安装Docker

简介：如何在Linux环境下安装docker

- 安装环境：Centos 7
- 安装条件：docker官方要求至少3.8以上，建议3.10以上
- Docker 版本：
  - docker EE 企业版本
  - docker CE 社区版本
- 关闭防火墙：systemctl stop firewalld.service vi /etc/selinux/config
- 安装Docker Ce 社区版本：
- 安装wget命令：
- 下载阿里云docker社区版 yum源

```
[root@localhost ~]# cd  /etc/yum.repos.d/
[root@localhost yum.repos.d]#
[root@localhost yum.repos.d]# wget http://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo
```

- 查看docker安装包：yum list | grep docker
- 安装Docker Ce 社区版本：yum install -y docker-ce.x86_64
- 设置开机启动：systemctl enable docker
- 更新xfsprogs：yum -y update xfsprogs
- 启动docker：systemctl start docker
- 查看版本：docker version
- 查看详细信息：docker info

2.3 Docker镜像的搜索下载以及查看删除实战

简介：如何搜索docker镜像

- 什么是镜像？
- 查看本地镜像：**docker images**
- 搜索镜像：**docker search centos**
- 搜索镜像并过滤是官方的： **docker search --filter "is-official=true" centos**
- 搜索镜像并过滤大于多少颗星星的：**docker search --filter stars=10 centos**
- 下载centos7镜像：**docker pull centos:7**
- 修改本地镜像名字（小写）：**docker tag centos:7 mycentos:1**
- 本地镜像的删除：**docker rmi centos:7**

2.4 Docker核心基础之配置阿里云镜像加速

简介：配置阿里云镜像加速

- 阿里云镜像加速器配置地址：https://cr.console.aliyun.com/cn-shenzhen/instances/mirrors
- 配置步骤：**vi /etc/docker/daemon.json**

```
{
"registry-mirrors": ["https://5xok66d4.mirror.aliyuncs.com"]
}
```

- 重启：**systemctl daemon-reload && systemctl restart docker**

2.5 Docker的体系结构之镜像与容器

简介：讲解镜像与容器之间的关系

- 一个镜像可以启动无数台容器（机器性能允许的情况下）
- 容器也容器之间的操作互不影响，处于隔离的环境

2.6 Docker核心基础之容器的构建等基本操作

简介：Docker容器的创建，查看，停止，重启等

- 构建容器：**docker run -itd --name=mycentos centos:7**
  - **-i** ：表示以交互模式运行容器（让容器的标准输入保持打开）
  - **-d**：表示后台运行容器，并返回容器ID
  - **-t**：为容器重新分配一个伪输入终端

    - **--name**：为容器指定名称
- 查看本地所有的容器：**docker ps -a**
- 查看本地正在运行的容器：**docker ps**
- 停止容器：**docker stop CONTAINER_ID / CONTAINER_NAME**
- 一次性停止所有容器：**docker stop $(docker ps -a -q)**
- 启动容器：**docker start CONTAINER_ID / CONTAINER_NAME**
- 重启容器：**docker restart CONTAINER_ID / CONTAINER_NAME**
- 删除容器：**docker rm CONTAINER_ID / CONTAINER_NAME**
- 强制删除容器：**docker rmi -f CONTAINER_ID / CONTAINER_NAME**
- 查看容器详细信息：**docker inspect CONTAINER_ID / CONTAINER_NAME**
- 进入容器：**docker exec -it 0ad5d7b2c3a4 /bin/bash**

2.7 Docker核心基础之容器的文件复制与挂载

简介：容器与宿主机之间文件复制与挂载

- 从宿主机复制到容器：**docker cp 宿主机本地路径 容器名字/ID：容器路径**
    - **docker cp /root/123.txt mycentos:/home/**

- 从容器复制到宿主机：**docker cp 容器名字/ID：容器路径 宿主机本地路径**
    - **docker cp mycentos:/home/456.txt /root**

- 宿主机文件夹挂载到容器里：**docker run -itd -v 宿主机路径:容器路径 镜像ID**
    - **docker run -itd -v /root/xdclass/:/home centos:7**

第三章：Docker核心必备之自定义镜像实战

3.1 构建自定义镜像的意义与应用场景

简介：介绍自定义镜像的重要性

- docker目前镜像的制作有俩种方法：
  - 基于Docker Commit制作镜像
  - 基于dockerfile制作镜像，Dockerfile方式为主流的制作镜像方式

3.2 Commit构建自定义镜像

简介：对容器的修改以及保存

- 简介：对容器的修改以及保存
  - 启动并进入容器：**docker run -it centos:7 /bin/bash**
  - 在/home 路径下创建xdclass文件夹：**mkdir /home/xdclass**
  - 安装ifconfig命令：**yum -y install net-tools**
  - 重启容器，查看容器的xdclass文件夹还在不在：**docker restart 67862569d4f7**
  - 删除容器，再重新启动一个容器进入查看有没有xdclass文件夹：**docker rm 67862569d4f7 && docker run -it centos:7 /bin/bash**
  - 构建镜像：
    - docker commit 4eb9d14ebb18 mycentos:7
    - docker commit -a "XD" -m "mkdir /home/xdclass" 4eb9d14ebb18 mcentos:7
    - -a：标注作者
    - -m：说明注释
    - 查看详细信息：docker inspect 180176be1b4c
  - 启动容器：docker run -itd 180176be1b4c /bin/bash
  - 进入容器查看：docker exec -it 2a4d38eca64f /bin/bash

3.3 核心必备知识之Dockerfile构建镜像实战

简介：Dockerfile构建镜像实战

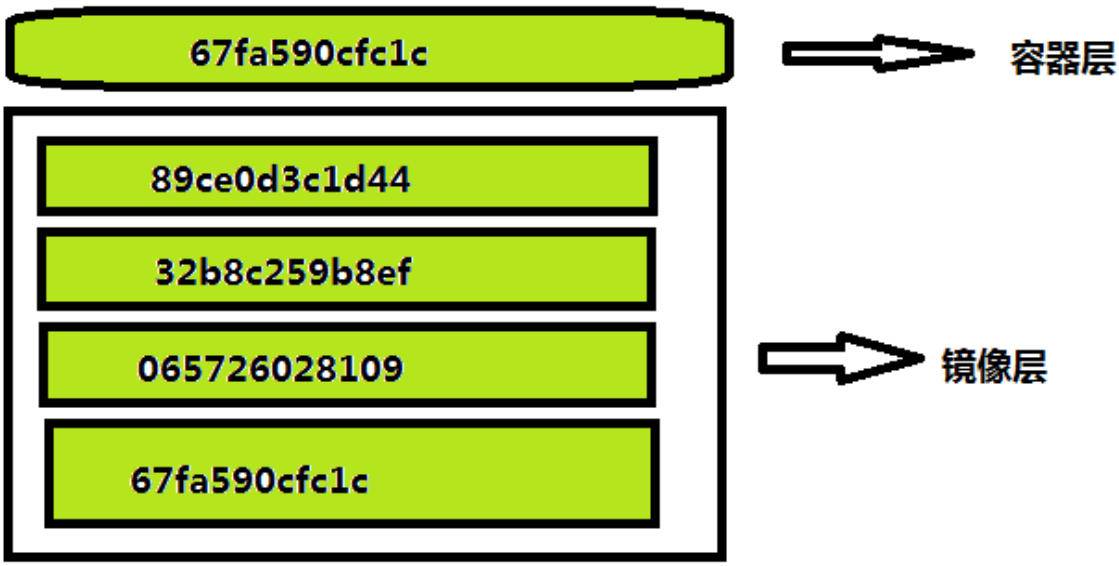
- **Dockerfile**

```
# this is a dockerfile
FROM centos:7
MAINTAINER XD 123456@qq.com
RUN echo "正在构建镜像！！！"
WORKDIR /home/xdclass
COPY 123.txt /home/xdclass
RUN yum install -y net-tools
```

- 构建：docker build -t mycentos:v2 .
- 查看：docker images
- 进入验证：验证成功

3.4 Docker核心知识之镜像分层结构剖析

简介：介绍镜像的分层结构



- 总结：
  - 共享资源
  - 对容器的任何改动都是发生容器层
  - 容器层是可写可读，而镜像层只读

3.5 不得不掌握的Dockerfile基础指令

简介：介绍一些常用的Dockerfile指令

- FROM
  - 基于哪个镜像
- MAINTAINER
  - 注明作者
- COPY
  - 复制文件进入镜像（只能用相对路径，不能用绝对路径）
- ADD
  - 复制文件进入镜像（假如文件是.tar.gz文件会解压）
- WORKDIR：
  - 指定工作目录，假如路径不存在会创建路径
- ENV
  - 设置环境变量
- EXPOSE
  - 暴露容器端口
- RUN

    - 在构建镜像的时候执行，作用于镜像层面
- ENTRYPOINT
    - 在容器启动的时候执行，作用于容器层，dockerfile里有多条时只允许执行最后一条
- CMD
    - 在容器启动的时候执行，作用于容器层，dockerfile里有多条时只允许执行最后一条
    - 容器启动后执行默认的命令或者参数，允许被修改
- 命令格式：
    - shell命令格式：RUN yum install -y net-tools
    - exec命令格式：RUN ["yum","install" ,"-y" ,"net-tools"]

- dockerfile

```
#第一个
FROM centos:7
RUN echo "images building!"
CMD ["echo","container","starting..."]
ENTRYPOINT ["echo","container","starting ！！！"]

#第二个
FROM centos:7
RUN echo "images building!"
CMD ["echo","containe1r","starting..."]
CMD ["echo","container2","starting..."]
ENTRYPOINT ["echo","container2","starting ！！！"]
ENTRYPOINT ["echo","container2","starting ！！！"]

#第三个
FROM centos:7
CMD ["-ef"]
ENTRYPOINT ["ps"]
```

3.6 实战系列之Dockerfile构建JAVA网站镜像

简介：Dockerfile构建java环境

- 本地宿主机配置jdk

```
export JAVA_HOME=/usr/local/jdk
export JRE_HOME=$JAVA_HOME/jre
export CLASSPATH=$JAVA_HOME/lib:$JRE_HOME/lib:$CLASSPATH
export PATH=$JAVA_HOME/bin:$JRE_HOME/bin:$PATH
```

- vi /etc/profile

- source /etc/profile
- 检验：java -version
- **dockerfile**

```
FROM centos:7
ADD jdk-8u211-linux-x64.tar.gz /usr/local
RUN mv /usr/local/jdk1.8.0_211 /usr/local/jdk
ENV JAVA_HOME=/usr/local/jdk
ENV JRE_HOME=$JAVA_HOME/jre
ENV CLASSPATH=$JAVA_HOME/lib:$JRE_HOME/lib:$CLASSPATH
ENV PATH=$JAVA_HOME/bin:$JRE_HOME/bin:$PATH
ADD apache-tomcat-8.5.35.tar.gz /usr/local
RUN  mv /usr/local/apache-tomcat-8.5.35  /usr/local/tomcat
EXPOSE 8080
ENTRYPOINT ["/usr/local/tomcat/bin/catalina.sh","run"]
```

- 启动容器：

```
docker run -itd -p 80:8080 -v /root/test/ROOT:/usr/local/tomcat/webapps/ROOT
mycentos:jdk /bin/bash
```

- 网页进行访问验证

3.7 实战系列之Dockerfile构建nginx镜像

简介：Dockerfile构建nginx

- dockerfile

```
FROM  centos:7
ADD nginx-1.16.0.tar.gz /usr/local
COPY nginx_install.sh /usr/local
RUN sh /usr/local/nginx_install.sh
EXPOSE 80
```

- 安装nginx的shell脚本

```
#!/bin/bash
yum install -y gcc gcc-c++ make pcre pcre-devel zlib zlib-devel
cd /usr/local/nginx-1.16.0
./configure --prefix=/usr/local/nginx && make && make install
```

- 制作Nginx镜像：
  - docker build -t mycentos:nginx .

- Nginx镜像启动注意
  - 在容器里nginx是以daemon方式启动，退出容器时，nginx程序也会随着停止：/usr/local/nginx/sbin/nginx
  - 使用前台方式永久运行：/usr/local/nginx/sbin/nginx -g "daemon off;"

- 检查验证：
  - docker run -itd -p 80:80 mycentos:nginx /usr/local/nginx/sbin/nginx -g "daemon off;"

3.8 实战系列之Dockerfile构建redis镜像

简介：Dockerfile构建redis

- 编写redis编译安装shell脚本redis_install.sh

```
#!/bin/bash
yum install -y gcc gcc-c++ make openssl openssl-devel
cd /home/redis-4.0.9
make && make PREFIX=/usr/local/redis install
mkdir -p /usr/local/redis/conf/
cp /home/redis-4.0.9/redis.conf  /usr/local/redis/conf/
sed -i  '69s/127.0.0.1/0.0.0.0/' /usr/local/redis/conf/redis.conf
sed -i  '88s/protected-mode yes/protected-mode no/' /usr/local/redis/conf/redis.conf
```

- 编写Dockerfile

```
FROM centos:7
ADD redis-4.0.9.tar.gz /home
COPY redis_install.sh /home
RUN sh /home/redis_install.sh
ENTRYPOINT /usr/local/redis/bin/redis-server /usr/local/redis/conf/redis.conf
```

- 测试redis：
  - 启动容器：docker run -itd -p 6380:6379 mycentos:redis #6380是宿主机端口，6379是容器的端口
  - 进入容器：docker exec -it 9b402baeaba7 /bin/bash
  - 宿主机连接redis： /usr/local/redis/bin/redis-cli -p 6380
  - 验证：

```
[root@localhost home]# /usr/local/redis/bin/redis-cli  -p 6380
127.0.0.1:6380> set name xdclass
OK
127.0.0.1:6380> get name
"xdclass"
127.0.0.1:6380>
```

3.9 实战系列之docker快速部署mysql数据库并初始化

简介：Dockerfile快速部署mysql数据库并初始化

- 官方网址：
  - https://hub.docker.com/
- 启动命令：
  - docker run --name some-mysql -p 3307:3306 -e MYSQL_ROOT_PASSWORD=abc123456 -d mysql:5.7
- 进入容器命令：
  - docker exec -it 4336ae28fbfa env LANG=C.UTF-8 /bin/bash
- dockerfile

```
FROM mysql:5.7
WORKDIR /docker-entrypoint-initdb.d
ENV LANG=C.UTF-8
ADD init.sql .
```

- 初始化sql语句：

```
-- 建库
create database  `db_student`;
SET character_set_client = utf8;
use db_student;

-- 建表
drop table if exists  `user`;
CREATE TABLE user (
id tinyint(5) zerofill auto_increment not null comment '学生学号',
name varchar(20) default null comment '学生姓名',
age tinyint default null comment '学生年龄',
class varchar(20) default null comment '学生班级',
sex char(5) not null comment '学生性别',
unique key (id)
)engine=innodb charset=utf8;

-- 插入数据

insert into user values('1','小明','15','初三','男');
insert into user values('2','小红','13','初二','女');
insert into user values('3','小东','14','初一','男');
```

```
insert into user values('4','小西','12','初二','男');
```


第四章：Docker核心知识之网络模式与特权指令

4.1 Docker 容器的网络模式介绍

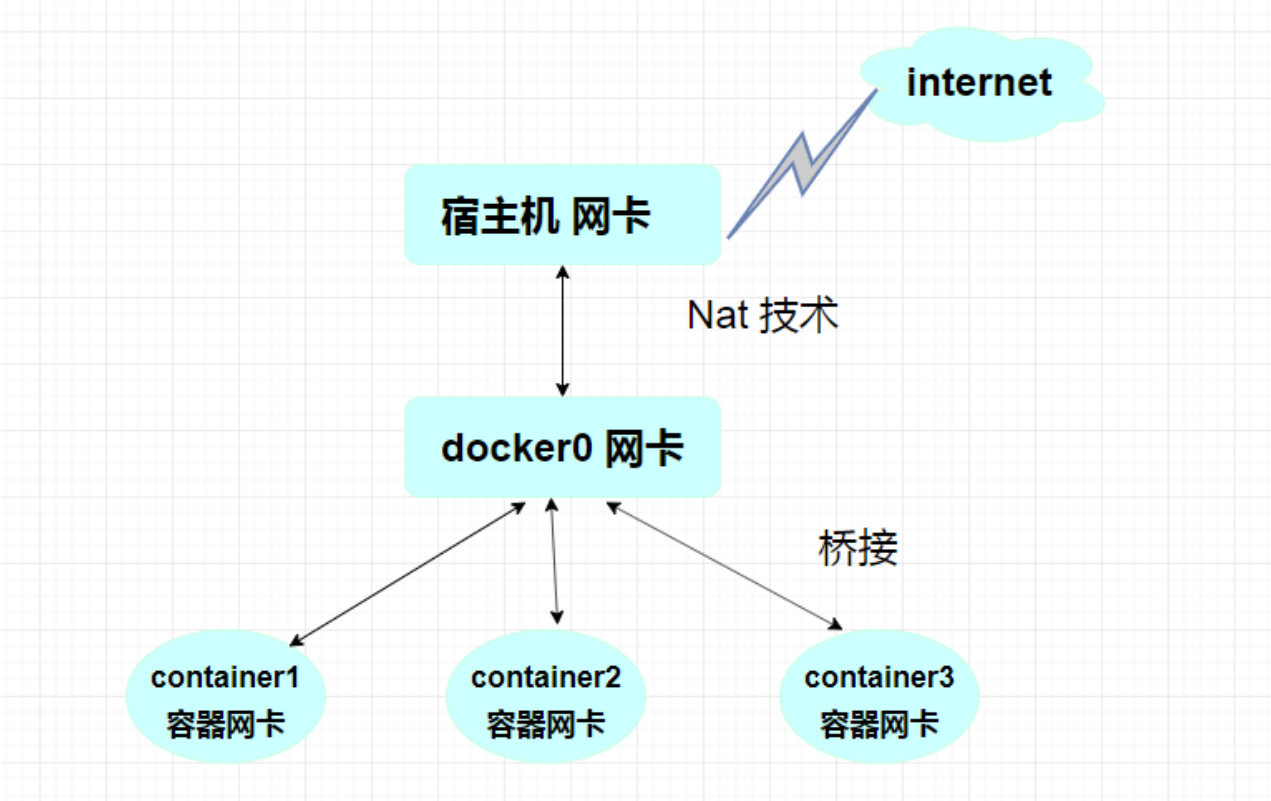
简介：介绍Docker网络模式

- 默认的三种网络模式：
  - **bridge**：桥接模式
  - **host**：主机模式
  - **none**：无网络模式
- 查看网络模式：
  - docker network ls

4.2 Docker 容器的bridge模式实战演练

简介：介绍Docker桥接网络模式

- 桥接模式是docker 的默认网络设置，当Docker服务启动时，会在主机上创建一个名为docker0的虚拟网桥，并选择一个和宿主机不同的IP地址和子网分配给docker0网桥
- 桥接拓扑图：



- 安装工具：
  - yum -y install net-tools
  - yum install -y bridge-utils

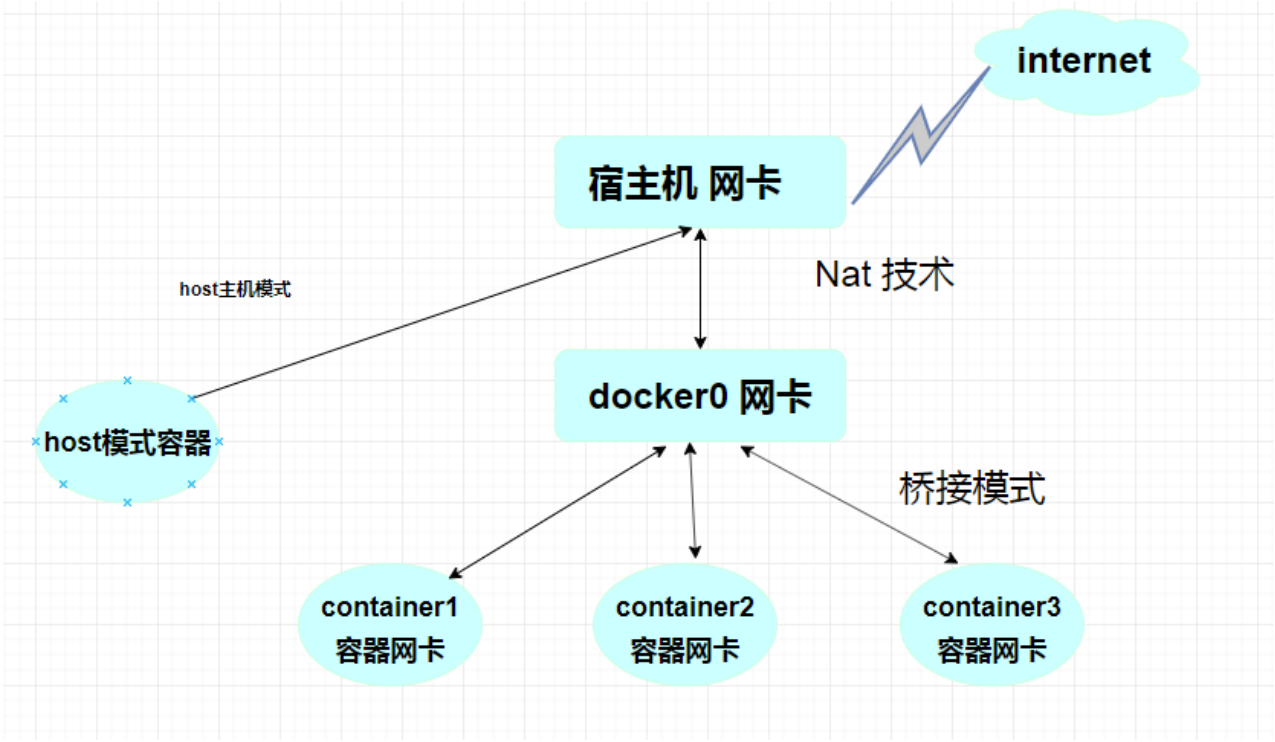
- 查看桥接情况：
  - **brctl show**

```
[root@localhost ~]# brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242055f4af6       no              veth60d950c
                                                        veth827b88f
                                                        vethaabef5c
                                                        vethd22be73
                                                        vethe1396d2
```

4.3 Docker 容器的host模式实战演练

简介：介绍Docker主机网络模式

- host 模式：该模式下容器是不会拥有自己的ip地址，而是使用宿主机的ip地址和端口。

- 

- 启动nginx容器命令并防火墙放开80端口：
  - docker run -d centos:nginx /usr/local/nginx/sbin/nginx -g "daemon off;"
  - firewall-cmd --zone=public --add-port=80/tcp --permanent
  - firewall-cmd --reload

4.4 Docker 容器的none模式介绍

简介：介绍Docker关闭网络模式介绍

- none模式：关闭模式
- 无法连外网

4.5 Docker 容器间基于Link实现单向通信

简介：介绍基于link单向通信

- 启动mysql数据库容器：
  - docker run --name mydb -e MYSQL_ROOT_PASSWORD=abc123456 -d mysql:5.7
- 启动tomcat应用容器并link到mysql数据库：
  - docker run -itd --name tomcat1 --link mydb tomcat:tag

    - 注意：mydb 这个容器一定要存在！
- 官方版的mysql 5.7 需要安装工具才有ping命令:
    - apt-get update && apt-get install iputils-ping

4.6 Docker容器间利用brige网桥实现双向通信

简介：介绍基于Bridge网桥实现双向通信

- 执行：**docker network ls**

```
[root@localhost ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
b6875f7a18dc        bridge              bridge              local
aa7e5dc89e49        host                host                local
b6ac4684d54d        my_bridge           bridge              local
2d1450c05ce0        none                null                local
```

- 创建一个新的网桥：**docker network create -d bridge my_bridge**
- 启动第一个容器：**docker run -itd --name tomcat centos:7**
- 启动第二个容器：**docker run -itd --name redis centos:7**
- 把第一个容器加入网桥：**docker network connect my_bridge tomcat**
- 把第二个容器加入网桥：**docker network connect my_bridge redis**
- 最后分别进入俩个容器中进行验证

4.7 Docker容器的特权模式介绍

简介：介绍特权模式

- 启动一个普通的容器
    - docker run -itd --name mycentos centos:7 /bin/bash
- 安装网络工具：
    - yum -y install net-tools
- 执行route -n
- 删除网关：
    - route del default gw 172.17.0.1
- 启动拥有特权模式的容器：

    - docker run -itd --privileged=true --name mycentos1 centos:7 /bin/bash
- 进入容器：
    - docker exec -it ef /bin/bash
- 删除网关
    - route del default gw 172.17.0.1
- 成功
- 备注：特权模式用的比较少

4.8 Docker核心知识之Volume数据共享

简介：Volume的介绍使用

- dockerfile

```
FROM centos:7
VOLUME ["/usr/local"]
```

- 注意：在dockerfile里设置volume是无法修改宿主机的挂载路径的

- 使用volume容器共享创建nginx集群
    - 使用**--volumes-from** 实现容器与容器之间volume共享
    - 创建nginx1
        - docker run -itd -p 8080:80 -v /usr/local/nginx/html:/usr/local/nginx/html --name nginx1 mycentos:nginx /usr/local/nginx/sbin/nginx -g "daemon off;"
    - 创建nginx2
        - docker run -itd -p 8081:80 --volumes-from nginx1 --name nginx2 mycentos:nginx /usr/local/nginx/sbin/nginx -g "daemon off;"
    - 创建nginx3
        - docker run -itd -p 8082:80 --volumes-from nginx1 --name nginx3 mycentos:nginx /usr/local/nginx/sbin/nginx -g "daemon off;"
    - 对/usr/local/nginx/html/index.html进行修改
    - 打开浏览器进行访问测试
- 使用docker inspect 容器ID 可以查看详细的挂载信息

第五章：实战系列之利用Compose操作容器

5.1 实用工具Docker-Compose的介绍与安装

简介：Compose的介绍

- docker-compose：是一个用于定义和运行多容器 Docker 的应用程序工具，可以帮助我们可以轻松、高效的管理容器
- compose的安装
  - 安装pip工具
    - yum install -y epel-release
    - yum install -y python-pip
  - 安装docker-compose
    - pip install -i https://pypi.tuna.tsinghua.edu.cn/simple docker-compose==1.24.1
  - 查看docker-compose版本：
    - docker-compose version
- 安装pip报错解决：
  - ```
    [root@localhost yum.repos.d]# yum install python-pip -y
    已加载插件：fastestmirror

     One of the configured repositories failed (未知),
     and yum doesn't have enough cached data to continue. At this point the only
     safe thing yum can do is fail. There are a few ways to work "fix" this:

    1. Contact the upstream for the repository and get them to fix the problem.

     2. Reconfigure the baseurl/etc. for the repository, to point to a working
        upstream. This is most often useful if you are using a newer
        distribution release than is supported by the repository (and the
        packages for the previous distribution release still work).

     3. Disable the repository, so yum won't use it by default. Yum will then
        just ignore the repository until you permanently enable it again or use
        --enablerepo for temporary usage:

            yum-config-manager --disable <repoid>

     4. Configure the failing repository to be skipped, if it is unavailable.
        Note that yum will try to contact the repo. when it runs most commands,
        so will have to try and fail each time (and thus. yum will be be much
        slower). If it is a very temporary problem though, this is often a nice
        compromise:

            yum-config-manager --save --setopt=<repoid>.skip_if_unavailable=true

     Cannot retrieve metalink for repository: epel/x86_64. Please verify its path and
    try again
    ```
    - 解决：`vi /etc/yum.repos.d/epel.repo` 修改配置文件，注释掉**metalink** ，取消注释 **baseurl**

      修改前如图：
    ```

```
[epel]
name=Extra Packages for Enterprise Linux 7 - $basearch
#baseurl=http://download.fedoraproject.org/pub/epel/7/$basearch
metalink=https://mirrors.fedoraproject.org/metalink?repo=epel-7&arch=$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7

[epel-debuginfo]
name=Extra Packages for Enterprise Linux 7 - $basearch - Debug
#baseurl=http://download.fedoraproject.org/pub/epel/7/$basearch/debug
metalink=https://mirrors.fedoraproject.org/metalink?repo=epel-debug-7&arch=$base
failovermethod=priority
enabled=0
```

改后如下图：

```
[epel]
name=Extra Packages for Enterprise Linux 7 - $basearch
baseurl=http://download.fedoraproject.org/pub/epel/7/$basearch
#metalink=https://mirrors.fedoraproject.org/metalink?repo=epel-7&arch=$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7

[epel-debuginfo]
name=Extra Packages for Enterprise Linux 7 - $basearch - Debug
#baseurl=http://download.fedoraproject.org/pub/epel/7/$basearch/debug
```

5.2 实用工具Docker-Compose的快速上手

简介：Compose快速上手

- 编写一个最最简单的yml

```
version: '3'
services:
  redis:
    image: mycentos:redis
```

- compose操作容器（一定要进入配置文件目录）
  - 后台启动容器：docker-compose up -d
  - 查看容器运行情况：docker-compose ps
  - 停止并删除容器：docker-compose down
  - 停止并删除容器并删除volume：docker-compose down --volumes
  - 停止启动容器：docker-compose stop；docker-compose start
  - docker-compose exec的使用：docker-compose exec redis bash
- 总结：
  - 操作docker-compose一定要在配置文件docker-compose.yml文件路径下操作
  - 格式一定要注意，该空格要空格

5.3 实用工具Docker-Compose核实用技能

简介：Compose核心实用技能掌握

- docker-compose.yml的三大部分：version，services，networks，最关键是services和networks两个部分
- compose设置网络模式
- compose使用端口映射
- compose设置文件共享
- compose管理多个容器
- docker-compose.yml

```
version: '3'
services:
  nginx:
    image: mycentos:nginx
    network_mode: "host"
    volumes:
    - /home:/usr/local/nginx/html
    - /var/logs/nginx/logs:/usr/local/nginx/logs
    command: /usr/local/nginx/sbin/nginx -g "daemon off;"


  redis:
    image: mycentos:redis
    ports:
    - "6380:6379"
```

5.4 实战项目篇之利用Docker-Compose快速搭建个人博客

简介：Compose快速搭建个人博客wordpress

- 官网：https://docs.docker.com/compose/wordpress/
- docker-compose.yml

```
version: '3.3'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
```

```
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {}
```

- 启动wordpress：docker-compose up -d
- 打开浏览器访问：IP:8000
- 进行安装配置
- 将删除容器和默认网络，但会保留WordPress数据库：`docker-compose down`
- 将删除容器，默认网络和WordPress数据库：`docker-compose down --volumes`

5.5 实战项目篇之Docker-Compose 详细分析

简介：分析上节课docker-compose.yml

- docker-compose.yml

```
version: '3.3'
services:
   db:
     image: mysql:5.7       #docker run -itd mysql:5.7
     volumes:
       - db_data:/var/lib/mysql     #采用的是卷标的形式挂载（注意：- db_data是参数，可以变，自定义，必须与下面对应）
     restart: always        #自动重启，保证服务在线
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress   #指定环境变量  docker -itd  -e MYSQL_ROOT_PASSWORD= somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db     # - db 是参数，合起来的意思是只有当上面的mysql数据库安装成功后，这个wordpress才可以被安装，还有一个功能，就是docker --link 将上面的mysql数据库，与这个wordpress应用连起来
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
```

```
        WORDPRESS_DB_PASSWORD: wordpress
        WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {}
```

- docker-compose中有两种方式可以设置volumes
  - 使用具体路径直接挂载到本地，特点就是直观
  - 使用卷标的形式，特点就是简洁，但是不知道数据到底在本地的什么位置。需要通过卷标查看
    - `docker volume ls`
    - `docker volume inspect` wordpress_db_data