

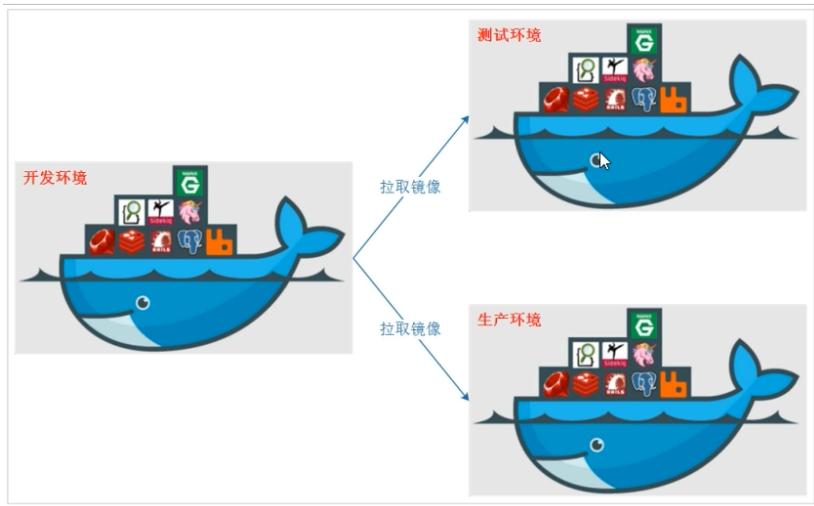
Docker

Docker简介

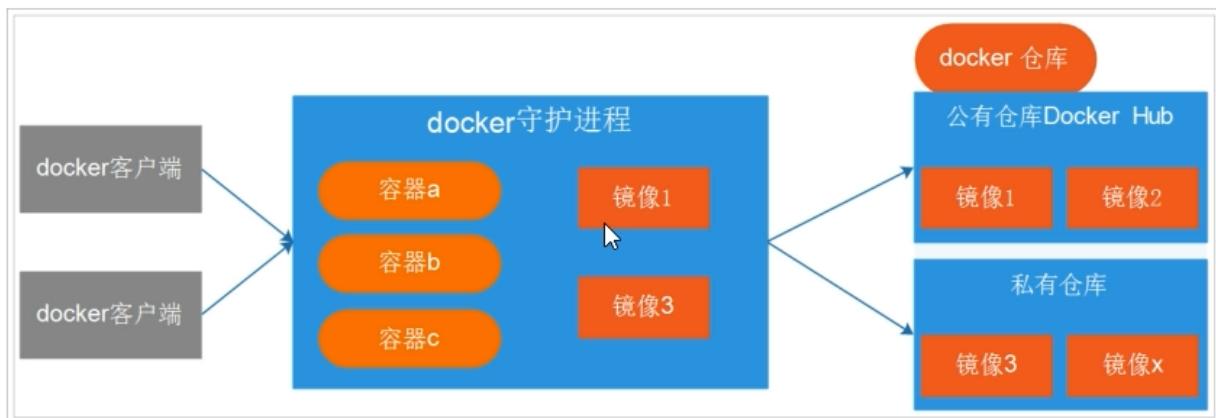
Docker是一个开源的应用容器引擎

Docker优势

启动速度快，占用体积小（通过复用主操作系统文件）



Docker组成部分



名称	说明
Docker 镜像 (Images)	Docker 镜像是用于创建 Docker 容器的模板。镜像是基于联合文件系统的一种层式结构，由一系列指令一步一步构建出来。
Docker 容器 (Container)	容器是独立运行的一个或一组应用。镜像相当于类，容器相当于类的实例
Docker 客户端 (Client)	Docker 客户端通过命令行或者其他工具使用 Docker API 与 Docker 的守护进程通信。
Docker 主机 (Host)	一个物理或者虚拟的机器用于执行 Docker 守护进程和容器。
Docker 守护进 程	是Docker服务器端进程，负责支撑Docker 容器的运行以及镜像的管理。
Docker 仓库 DockerHub (Registry)	Docker 仓库用来保存镜像，可以理解为代码控制中的代码仓库。Docker Hub提供了庞大的镜像集合供使用。用户也可以将自己本地的镜像推送到Docker仓库供其他人下载。

Docker的安装

1、更新yum源为ustc

```

1 # 1、yum 包更新到最新
2 sudo yum update
3
4 # 2、作用：安装需要的软件包， yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的
5 sudo yum install -y yum-utils device-mapper-persistent-data lvm2
6
7 # 3、设置yum源
8 # 3.1、方案一：使用ustc的（推荐）
9 sudo yum-config-manager --add-repo http://mirrors.ustc.edu.cn/docker-
ce/linux/centos/docker-ce.repo
10 # 3.2、方案二：使用阿里云（可能失败）
11 sudo yum-config-manager --add-repo http://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo
12
13 # 4、安装docker；出现输入的界面都按 y
14 sudo yum install -y docker-ce

```

2、安装docker

sudo yum install -y docker-ce

3. 安装后查看docker版本

docker -v

4、设置ustc的docker镜像源

vi /etc/docker/daemon.json

输入以下内容：

```
{
  "registry-mirrors": ["https://docker.mirrors.ustc.edu.cn"]
}
```

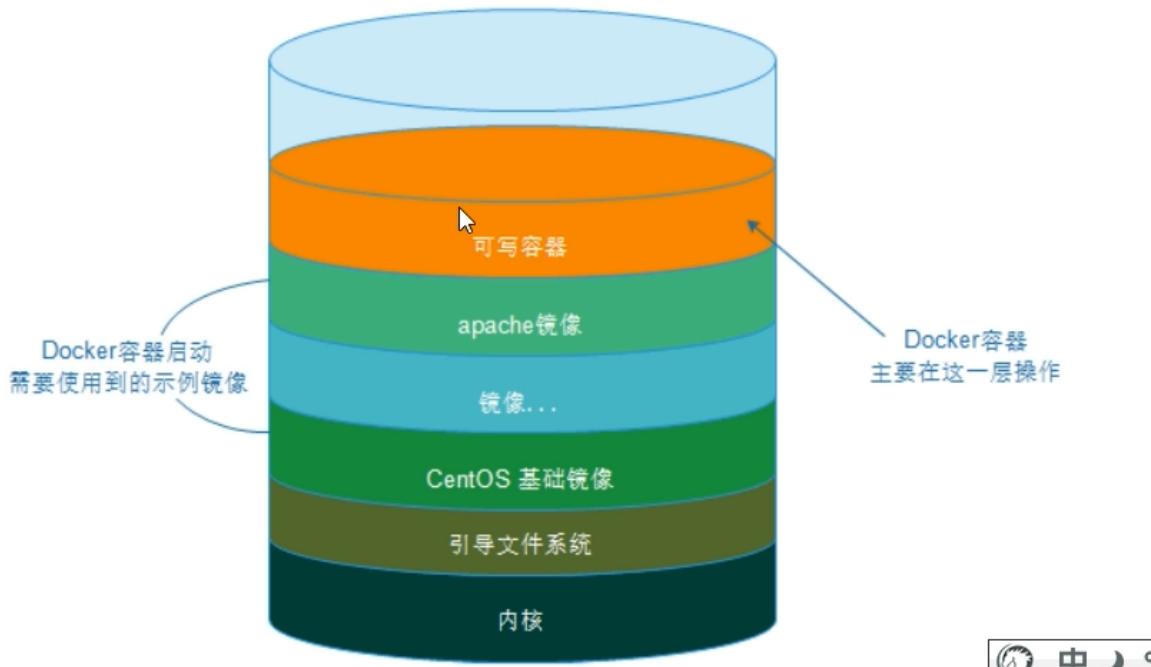
5、docker的启动停止

```
systemctl start docker  
systemctl stop docker  
systemctl restart docker  
systemctl status docker
```

6、设置docker开机自启动

```
systemctl enable docker
```

镜像：Docker镜像是由文件系统叠加而成（是一种文件的存储形式）；是docker中的核心概念，可以认为镜像就是对某些运行环境或者软件打的包，用户可以从docker仓库中下载基础镜像到本地，比如开发人员可以从docker仓库拉取（下载）一个只包含centos7系统的基础镜像，然后在这个镜像中安装jdk、mysql、Tomcat和自己开发的应用，最后将这些环境打成一个新的镜像。开发人员将这个新的镜像提交给测试人员进行测试，测试人员只需要在测试环境下运行这个镜像就可以了，这样就可以保证开发人员的环境和测试人员的环境完全一致。



1、查看镜像

```
docker images
```

2、搜索镜像

```
docker search centos7
```

3、拉取(最新或指定版本)镜像

```
docker pull 镜像名称:版本号 注：若不指定版本号，默认拉取最新的镜像
```

4、删除镜像

```
docker rmi [image id]
```

查看，创建，启动交互式容器

使用查看容器命令，拉取centos的镜像之后结合容器启动命令和选项`-it` 启动交互式容器
容器是由镜像运行产生的运行实例，镜像与容器的关系就像java类和对象实例的关系

容器相关命令：

1、查看容器

查看正在运行的容器

docker ps

查看所有容器

docker ps -a

2、创建容器

docker run 参数

参数说明：

-i：表示运行容器

-t：表示容器启动后会进入其命令行。加入这两个参数后，容器创建就能登录进去。即分配一个伪终端。

--name :为创建的容器命名。

-v：表示目录映射关系（前者是宿主机目录，后者是映射到宿主机上的目录），可以使用多个 - v做多个目录或文件映射。注意：最好做目录映射，在宿主机上做修改，然后共享到容器上。

-d：在run后面加上-d参数，则会创建一个**守护式容器**在后台运行（这样创建容器后不会自动登录容器，如果只加-i -t两个参数，创建后就会自动进去容器）。

-p：表示端口映射，前者是宿主机端口，后者是容器内的映射端口。可以使用多个-p做多个端口映射

3、启动交互式容器

创建并启动交互式容器

docker run -it --name=容器名称 镜像 指定解析shell脚本的解析器

如： docker run -it --name=mycentos1 centos /bin/bash

启动交互式容器之后，会直接进入容器终端（容器的命令行），此时可以查看容器的文件结构

4、停止容器

进入交互式容器终端之后退出终端，并停止容器

exit

5、创建并启动守护式容器

守护式容器在启动之后，会一直在后台运行，适用于需要长期运行容器的情况

创建并启动守护式容器，启动之后不会直接进入

docker run -di --name=mycentos2 centos:7

启动容器后，进入容器

docker exec -it mycentos2 /bin/bash

退出容器(并不会停止容器)

exit

6、停止容器

可以通过一下命令停止守护式容器

停止容器

docker stop mycentos1或者容器id

#再次启动容器

docker start mycentos1

7、文件拷贝

```
# 将文件从宿主机拷贝到容器中  
docker cp 文件名 容器名:容器目录  
# 将容器中的文件拷贝到宿主机中  
docker cp 容器名:容器目录 宿主机目录
```

8、目录挂载

可以在[创建容器的时候](#)，将宿主机的目录与容器内的目录进行映射，这样我们就可以通过修改宿主机某个目录的文件从而去影响容器

```
1 # 创建linux宿主机器要挂载的目录  
2 mkdir /usr/local/test  
3  
4 # 创建并启动容器mycentos3，并挂载linux中的/usr/local/test目录到容器的/usr/local/test；也就是在  
5 # linux中的/usr/local/test中操作相当于对容器相应目录操作  
6 docker run -di -v /usr/local/test:/usr/local/test --name=mycentos3 centos:7  
7  
8 # 在linux下创建文件  
9 touch /usr/local/test/default.txt  
10  
11 # 进入容器  
12 docker exec -it mycentos3 /bin/bash
```

注意：如果你共享的是多级的目录，可能会出现权限不足的提示。这是因为CentOS7中的安全模块selinux把权限禁掉了，需要添加参数 `--privileged=true` 来解决挂载的目录没有权限的问题。

9、查看容器ip地址

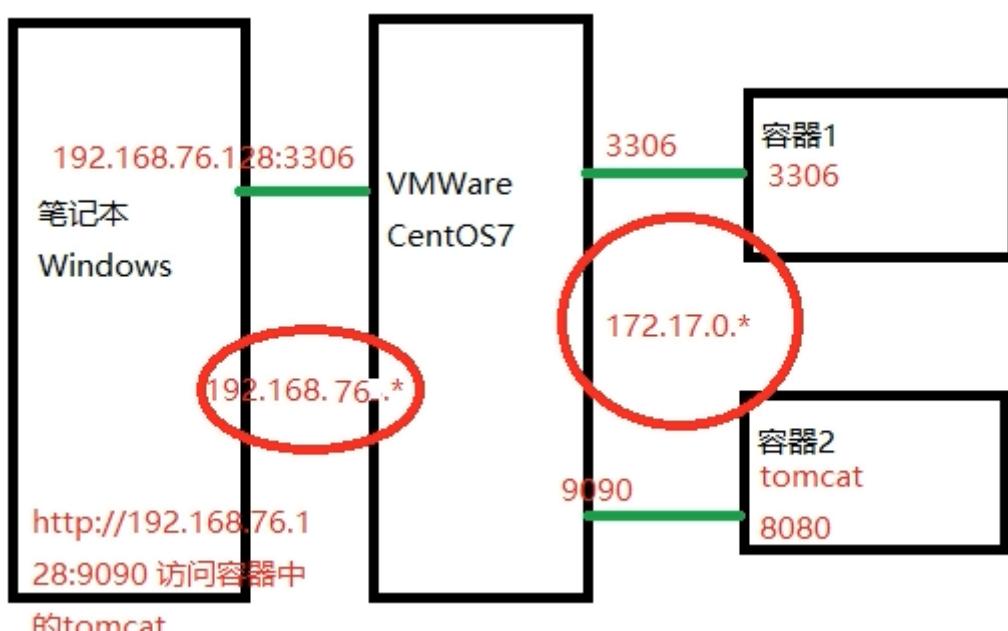
一般，容器和宿主机在同一个网段

`docker inspect 容器名`

10、删除容器

`# 只能删除停止状态的容器`

`docker rm 容器名`



MySQL容器部署

1、拉取mysql镜像

```
# 拉取MySQL 5.7镜像  
docker pull centos/mysql-57-centos
```

2、创建并启动守护式容器

```
docker run -di --name=mysql5.7 -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root mysql
```

-p 代表端口映射，格式为 宿主机映射端口:容器运行端口

-e 代表添加环境变量 MYSQL_ROOT_PASSWORD 是root用户的远程登陆密码（如果是在容器中使用root登录的话，那么其密码为空）

```
1 # 创建mysql5.7容器  
2 docker run -di --name=mysql5.7 -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root centos/mysql-  
57-centos7
```

docker run 后台运行 容器名 -p 映射端口 -e 远程连接root密码 镜像名

```
docker run -di --name=mysql5.7 -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root  
centos/mysql-57-centos7
```

3、在容器中操作mysql

```
# 启动并进入容器
```

```
docker exec -it mysql容器名 /bin/bash
```

刚创建的mysql容器没有密码

查看mysql镜像

```
● ● ●  
1 docker images | grep mysql
```

启动mysql容器

```
● ● ●  
1 docker run -p 3306:3306 --name mymysql -v $PWD/conf:/etc/mysql/conf.d -v $PWD/logs:/logs -v $PWD/data:/var/lib/mysql  
2 -e MYSQL_ROOT_PASSWORD=password123 -d mysql:5.6  
3  
4 //不挂载  
docker run --name mymysql -e MYSQL_ROOT_PASSWORD=mysqlpassword123 -p 3306:3306 -d mysql:5.6
```

ps:参数解释

- -p 3306:3306: 将容器的 3306 端口映射到主机的 3306 端口。
- -v \$PWD/conf:/etc/mysql/conf.d: 将主机当前目录下的 conf/my.cnf 挂载到容器的 /etc/mysql/my.cnf。
- -v \$PWD/logs:/logs: 将主机当前目录下的 logs 目录挂载到容器的 /logs。
- -v \$PWD/data:/var/lib/mysql: 将主机当前目录下的 data 目录挂载到容器的 /var/lib/mysql。
- -e MYSQL_ROOT_PASSWORD=123456: 初始化 root 用户的密码。

4、使用图形界面工具(windows中的navicat)操作在docker中安装的mysql

由于windows和docker容器不在同一网段，所以需要在创建容器的时候，使用-p指定端口映射。

五、添加远程登录用户

```
CREATE USER 'newroot'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
```

```
mysql> CREATE USER 'newroot'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
Query OK, 0 rows affected (0.00 sec)
```

六、给予远程用户所有表所有权限

```
GRANT ALL PRIVILEGES ON *.* TO 'newroot'@'%';
```

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'newroot'@'%';
Query OK, 0 rows affected (0.00 sec)
```

七、刷新权限

```
FLUSH PRIVILEGES;
```

mysql8+版本的授权语句会有所不同

Tomcat容器部署

目标：拉取tomcat镜像，启动容器，操作容器中的tomcat

分析：

- 将项目文件上传到容器中的tomcat目录（ webapps ）
创建容器的时候可以指定-v进行目录挂载，tomcat在容器中的目录（ /usr/local/tomcat/webapps ）
- 可以通过外部浏览器访问容器中的项目
创建容器的时候可以指定-p进行端口映射

1. 拉取tomcat镜像；`docker pull tomcat`

2. 创建并启动tomcat容器；

```
docker run -di --name=mytomcat -p 9000:8080 -v
/usr/local/tomcat/webapps:/usr/local/tomcat/webapps tomcat
```

3. 访问容器中tomcat

<http://192.168.76.128:9000>

1、拉取镜像

```
# 拉取tomcat镜像
docker pull tomcat
```

```

1 # 创建tomcat容器;并挂载了webapps目录
2 docker run -di --name=mymtomcat -p 9000:8080 -v
   /usr/local/tomcat/webapps:/usr/local/tomcat/webapps tomcat
3
4 # 如果出现 WARNING: IPv4 forwarding is disabled. Networking will not work.
5 #执行如下操作
6 # 1、编辑 sysctl.conf
7 vi /etc/sysctl.conf
8
9 # 2、在上述打开的文件中后面添加
10 net.ipv4.ip_forward=1 ↵
11
12 # 3、重启network
13 systemctl restart network

```

小结：上传项目文件可以使用容器中的目录挂载功能，外部访问可以使用端口映射，也就是说：项目可以直接上传到centOS系统中的目录，并直接通过访问系统的端口访问到容器。

Nginx容器部署

目标：拉取nginx镜像，启动容器，访问nginx

分析：

nginx的默认访问端口是：80

在创建容器的时候需要进行端口映射，指定-p，映射的端口80

1. 拉取nginx镜像；
2. 创建并启动nginx容器；
3. 在浏览器上访问容器中nginx，<http://192.168.76.128>

`docker pull nginx`

`docker run -di --name=nginx -p 81:80 nginx`

`docker ps`

如果宿主机的端口号被占用了，需要改变宿主机的端口映射，容器不用改，不存在端口占用（沙箱机制），在访问的时候需要带上宿主机的ip和端口号。

```
[root@VM-16-6-centos ~]# docker start nginx
nginx
[root@VM-16-6-centos ~]# docker exec -di nginx /bin/bash
[root@VM-16-6-centos ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            NAMES
STATUS              PORTS              NAMES
ebdd795100a2        redis              "docker-entrypoint.s..."   2 hours ago       redis
Up 2 minutes        0.0.0.0:6390->6379/tcp, :::6390->6379/tcp
11f5f03ee645        nginx              "/docker-entrypoint...."  2 hours ago       nginx
Up 53 seconds       0.0.0.0:81->80/tcp, :::81->80/tcp
1f2ddcbb9521e       centos/mysql-57-centos7  "container-entrypoin..."  4 hours ago       mysql
Up 2 minutes        0.0.0.0:3307->3306/tcp, :::3307->3306/tcp
[root@VM-16-6-centos ~]#
```

Redis容器部署

目标：拉取redis镜像，启动容器，操作容器中的redis

分析：

1. 拉取redis镜像；
2. 创建并启动redis容器；默认端口是6379，如果需要外部访问则可以使用端口映射；
3. 连接redis: ①使用命令行客户端 ②使用图形界面工具

小结：

```
# 创建容器
docker run -di --name=myredis -p 6379:6379 redis

# 进入容器
docker exec -it myredis /bin/bash
```

```
[root@itheima ~]# docker pull redis
Using default tag: latest
latest: Pulling from library/redis
Digest: sha256:e549a30b3c31e6305b973e0d9113a3d38d60566708137af9ed7cbdce5650c5cc
Status: Image is up to date for redis:latest
[root@itheima ~]# docker run -di --name=myredis -p 6379:6379 redis
ed04805c3f9cd8408e5154a89391f890c00b045021e00c108af04155b94ab
[root@itheima ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
ed04805c3f9c        redis              "docker-entrypoint.s..."   5 seconds ago     Up 3 seconds      0.0.0.0:6379
1bcd0d9fe0a        nginx              "nginx -g 'daemon of..."  8 minutes ago    Up 8 minutes      0.0.0.0:80
0ea446849fa1       tomcat             "catalina.sh run"      17 minutes ago   Up 17 minutes     0.0.0.0:9000
c13c8e6420be       centos/mysql-57-centos7 "container-entrypoint..." About an hour ago Up About an hour  0.0.0.0:3306
6025fef36038       centos:7          "/bin/bash"            2 hours ago     Up 2 hours       0.0.0.0:22
b248f0c51202       centos:7          "/bin/bash"            4 hours ago     Up 2 hours       0.0.0.0:22
[root@itheima ~]# docker exec -it myredis /bin/bash
root@ed04805c3f9c:/data# cd /usr/local/bin
root@ed04805c3f9c:/usr/local/bin# ls
docker-entrypoint.sh  gosu  redis-benchmark  redis-check-aof  redis-check-rdb  redis-cli  redis-sentinel  redis-server
root@ed04805c3f9c:/usr/local/bin# ./redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> set str heima
OK
127.0.0.1:6379> get str
"heima"
127.0.0.1:6379>
```

进入redis容器之后还需要启动redis-cli才能操作redis

Docker-Compose

5.1. Compose简介

5.1.1. 概念

Compose项目是Docker官方的开源项目，负责实现对Docker容器集群的快速编排。它是一个定义和运行多容器的**docker应用工具**。使用compose，你能通过YAML文件配置你自己的服务，然后通过一个命令，你能使用配置文件创建和运行所有的服务。

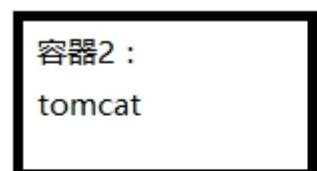
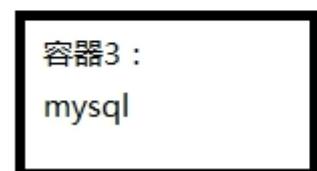
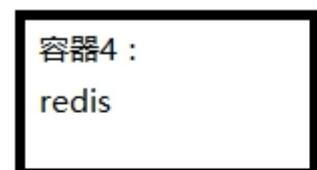
5.1.2. 组成

Docker-Compose将所管理的容器分为三层，分别是工程（project），服务（service）以及容器（container）。Docker-Compose运行目录下的所有文件（docker-compose.yml, extends文件或环境变量文件等）组成一个工程，若无特殊指定工程名即为当前目录名。一个工程当中可包含多个服务，每个服务中定义了容器运行的镜像，参数，依赖。一个服务当中可包括多个容器实例。

- **服务（service）**：一个应用的容器，实际上可以包括若干运行相同镜像的容器实例。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、依赖哪些其他服务等等，即以容器为粒度，用户需要Compose所完成的任务。
- **项目（project）**：由一组关联的应用容器组成的一个完整业务单元，在docker-compose.yml中定义。即是Compose的一个配置文件可以解析为一个项目，Compose通过分析指定配置文件，得出配置文件所需完成的所有容器管理与部署操作。

Docker-Compose的工程配置文件默认为docker-compose.yml，可通过环境变量COMPOSE_FILE或-f参数自定义配置文件，其定义了多个有依赖关系的服务及每个服务运行的容器。

使用一个Dockerfile模板文件，可以让用户很方便的定义一个单独的应用容器。在工作中，经常会碰到需要多个容器相互配合来完成某项任务的情况。例如：要部署一个Web项目，除了Web服务容器，往往还需要再加上后端的数据仓库服务容器，甚至还包括负载均衡容器等。



docker-compose项目配置
docker-compose.yml可以同时启动容器和指定
容器之间的依赖关系

5.2. 安装与卸载

Compose目前已经完全支持Linux、Mac OS和Windows，在我们安装Compose之前，需要先安装Docker。下面我们将以编译好的二进制包方式安装在Linux系统中。

5.2.1. 安装

```
1 curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
2 # 设置文件可执行权限
3 chmod +x /usr/local/bin/docker-compose
4
5 # 查看版本信息
6 docker-compose --version
7
```

docker-compose是一个应用工具；可以通过配置docker-compose.yml文件同时启动多个容器。

部署项目时可以编写一个docker-compose.yml文件作为启动项目单位，同时启动项目相关的那些容器。

使用Compose前，可以通过执行 docker-compose --help|-h 来查看Compose基本命令用法。

也可以通过执行 docker-compose [COMMAND] --help 或者 docker-compose --help [COMMAND] 来查看某个具体的使用格式。

可以知道Compose命令的基本的使用格式为：

```
1 docker-compose [-f 参数...] [options] [COMMAND] [ARGS...]
```

命令选项如下：

```
1 -f, --file FILE指定使用的Compose模板文件，默认为docker-compose.yml，可以多次指定。
2 -p, --project-name NAME指定项目名称，默认将使用所在目录名称作为项目名。
3 -x-network-driver 使用Docker的可拔插网络后端特性（需要Docker 1.9 及以后版本）
4 -x-network-driver DRIVER指定网络后端的驱动，默认为bridge（需要Docker 1.9 及以后版本）
5 -verbose输出更多调试信息
6 -v, --version打印版本并退出
```

5.3.1. up

格式为：

```
1 docker-compose up [options] [--scale SERVICE=NUM...] [SERVICE...]
```

`up`命令十分强大，它尝试自动完成包括构建镜像，（重新）创建服务，启动服务，并关联服务相关容器的一些列操作。链接的服务都将会被自动启动，除非已经处于运行状态。

多数情况下我们可以直接通过该命令来启动一个项目。

选项包括：

- 1 `-d` 在后台运行服务容器
- 2 `-no-color` 不使用颜色来区分不同的服务的控制输出
- 3 `-no-deps` 不启动服务所链接的容器
- 4 `-force-recreate` 强制重新创建容器，不能与`-no-recreate`同时使用
- 5 `-no-recreate` 如果容器已经存在，则不重新创建，不能与`-force-recreate`同时使用
- 6 `-no-build` 不自动构建缺失的服务镜像
- 7 `-build` 在启动容器前构建服务镜像
- 8 `-abort-on-container-exit` 停止所有容器，如果任何一个容器被停止，不能与`-d`同时使用
- 9 `-t, --timeout TIMEOUT` 停止容器时候的超时（默认为10秒）
- 10 `-remove-orphans` 删除服务中没有在compose文件中定义的容器
- 11 `-scale SERVICE=NUM` 设置服务运行容器的个数，将覆盖在compose中通过`scale`指定的参数

5.3.2. ps

格式为：

```
1 | docker-compose ps [options] [SERVICE...]
```

列出项目中目前的所有容器。

选项包括：



```
1 | -q 只打印容器的ID信息
```

5.3.3. stop

格式为：

```
1 | docker-compose stop [options] [SERVICE...]
```

停止已经处于运行状态的容器，但不删除它。

选项包括：

```
1 | -t, --timeout TIMEOUT 停止容器时候的超时（默认为10秒）
```

5.3.4down

停止和删除容器、网络、卷、镜像，这些内容是通过docker-compose up命令创建的。默认值删除容器网络，可以通过指定 rmi、volumes参数删除镜像和卷。

选项包括：

- 1 -rmi type 删除镜像，类型必须是：‘all’：删除compose文件中定义的所有镜像；‘local’：删除镜像名为空的镜像
- 2 -v, --volumes 删除已经在compose文件中定义的和匿名的附在容器上的数据卷
- 3 -remove-orphans 删除服务中没有在compose中定义的容器

5.3.5. restart

格式为：

```
1 | docker-compose restart [options] [SERVICE...]
```

5.3.6. rm

格式为：

```
1 | docker-compose rm [options] [SERVICE...]
```

删除所有（停止状态的）服务容器。

选项包括：

- 1 -f, --force 强制直接删除，包括非停止状态的容器
- 2 -v 删除容器所挂载的数据卷

5.3.7. start

格式为：

```
1 | docker-compose start [SERVICE...]
```

启动已经存在的服务容器。

5.3.8. run

格式为：

5.3.12. exec

格式为：

```
1 | docker-compose exec [options] SERVICE COMMAND [ARGS...]
```

与 `docker exec` 命令功能相同，可以通过 service name 登陆到容器中。

选项包括：

```
1 | -d 分离模式，后台运行命令。
2 | -privileged 获取特权。
3 | -user USER 指定运行的用户。
4 | -T 禁用分配TTY。By default docker-compose exec分配 a TTY。
5 | -index=index 当一个服务拥有多个容器时，可通过该参数登陆到该服务下的任何服务，例如：docker-compose
   | exec --index=1 web /bin/bash , web服务中包含多个容器
```

5.4. Compose模板文件

模板文件是使用Compose的核心，涉及的指令关键字也比较多，大部分指令与 `docker run` 相关参数的含义都是类似的。默认的模板文件名称为 `docker-compose.yml`，格式为 YAML 格式。

比如一个Compose模板文件：

```
1 | version: "2"
2 | services:
3 |   web:
4 |     images: nginx
5 |     ports:
6 |       - "8080:80"
7 |     volumes:
8 |       - /usr/local/abc:/usr/local/cba
9 | #volumes:
10 |
11 | #networks:
```

Docker Compose的模板文件主要分为3个区域，如下：

- **services**

服务，在它下面可以定义应用需要的一些服务，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、依赖哪些其他服务等等。

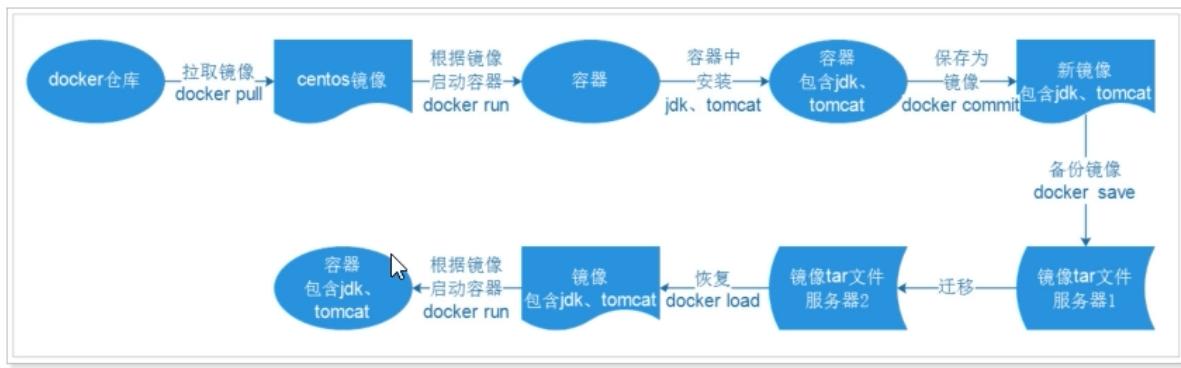
- **volumes**

数据卷，在它下面可以定义的数据卷（名字等等），然后挂载到不同的服务下去使用。

- **networks**

应用的网络，在它下面可以定义应用的名字、使用的网络类型等等。

6. 迁移与备份



其中涉及到的命令有：

- docker commit 将容器保存为镜像
- docker save 将镜像备份为tar文件
- docker load 根据tar文件恢复为镜像

```
[root@itheima ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
83b188764180        tomcat              "catalina.sh run"   26 minutes ago    Up 26 minutes      0.0.0.0:9090->8080/tcp   web1
3a6a3da8c051        centos/mysql-57-centos7 "container-entrypoint..." 26 minutes ago    Up 26 minutes      0.0.0.0:3306->3306/tcp   mysql1
a9c1cd8ce624        redis               "docker-entrypoint.s..." 26 minutes ago    Up 26 minutes      0.0.0.0:6379->6379/tcp   redis1
ed04805c3f9c        redis               "docker-entrypoint.s..." 2 hours ago       Exited (0) 27 minutes ago
1bcd0d9fe0a         nginx              "nginx -g 'daemon of..." 2 hours ago       Exited (0) 27 minutes ago
0ea446849fa1        tomcat              "catalina.sh run"   2 hours ago       Exited (143) 27 minutes ago
c1393656            centos/mysql-57-centos7 "container-entrypoint..." 2 hours ago       Exited (0) 27 minutes ago
6025fef36038        centos7           "/bin/bash"        3 hours ago       Exited (137) 27 minutes ago
b248f0c51202        centos7           "/bin/bash"        3 hours ago       Exited (137) 27 minutes ago
[root@itheima ~]# docker commit mynginx mynginx
sha256:fe0065fe7c824365c7518649a8ccaa81c18ce92793a7a9669dccaf10b9776fb0
[root@itheima ~]# docker images
REPOSITORY          TAG      IMAGE ID      CREATED             SIZE
mynginx             latest   fe0065fe7c82   5 seconds ago   109MB
nginx               latest   62c261073ecf   12 hours ago    109MB
tomcat              latest   894b39cf2fa1   6 days ago     522MB
redis               latest   d3e3588af517   2 weeks ago    95MB
centos/mysql-57-centos7  latest   e35b3f74ae0   2 months ago   452MB
centos              7        9f38484d220f   2 months ago   202MB
[root@itheima ~]# docker save -o mynginx.tar mynginx
[root@itheima ~]# ll
总用量 110356
-rw-r--r--. 1 root root      0 6月  5 15:18 abc.txt
-rw-----. 1 root root 1227 5月 28 12:15 anaconda-ks.cfg
-rw-r--r--. 1 root root      0 6月  5 15:19 cba.txt
-rw-----. 1 root root 112996864 6月  5 18:37 mynginx.tar
[root@itheima ~]# docker rm mynginx
mynginx
[root@itheima ~]# docker rmi mynginx
Untagged: mynginx:latest
Deleted: sha256:fe0065fe7c824365c7518649a8ccaa81c18ce92793a7a9669dccaf10b9776fb0
Deleted: sha256:24410b42e07e917658ecf170a3dc34bf02d4cf5d3075f2e062de60843d076bc
[root@itheima ~]# docker images
REPOSITORY          TAG      IMAGE ID      CREATED             SIZE
nginx               latest   62c261073ecf   12 hours ago    109MB
tomcat              latest   894b39cf2fa1   6 days ago     522MB
redis               latest   d3e3588af517   2 weeks ago    95MB
centos/mysql-57-centos7  latest   e35b3f74ae0   2 months ago   452MB
centos              7        9f38484d220f   2 months ago   202MB
```

```
[root@itheima ~]# docker load -i mynginx.tar
8782ed8918c: Loading layer [=====>] 5.632kB/5.632kB
Loaded image: mynginx:latest
[root@itheima ~]# docker images
REPOSITORY          TAG      IMAGE ID      CREATED             SIZE
mynginx             latest   fe0065fe7c82   3 minutes ago   109MB
nginx               latest   62c261073ecf   12 hours ago    109MB
tomcat              latest   894b39cf2fa1   6 days ago     522MB
redis               latest   d3e3588af517   2 weeks ago    95MB
centos/mysql-57-centos7  latest   e35b3f74ae0   2 months ago   452MB
centos              7        9f38484d220f   2 months ago   202MB
[root@itheima ~]# docker run -d --name=mynginx -p 80:80 mynginx
e3656eadf0e564d2151f98ab2a757c75ce9773c42b312df4759ab750c1d31f2
[root@itheima ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
e3656eadf0e        mynginx             "nginx -g 'daemon of..." 7 seconds ago    Up 6 seconds      0.0.0.0:80->80/tcp   mynginx
83b188764180        tomcat              "catalina.sh run"   31 minutes ago   Up 31 minutes     0.0.0.0:9090->8080/tcp   web1
3a6a3da8c051        centos/mysql-57-centos7 "container-entrypoint..." 31 minutes ago   Up 31 minutes     0.0.0.0:3306->3306/tcp   mysql1
a9c1cd8ce624        redis               "docker-entrypoint.s..." 31 minutes ago   Up 31 minutes     0.0.0.0:6379->6379/tcp   redis1
[root@itheima ~]#
```

7.1. 什么是Dockerfile文件

前面的课程中已经知道了，要获得镜像，可以从Docker仓库中进行下载。那如果我们想自己开发一个镜像，那该如何做呢？答案是：Dockerfile

Dockerfile其实就是一个文本文件，由一系列命令和参数构成，Docker可以读取Dockerfile文件并根据Dockerfile文件的描述来构建镜像。 ↴

Dockerfile文件内容一般分为4部分：

- 基础镜像信息
- 维护者信息
- 镜像操作指令
- 容器启动时执行的指令

7.2. Dockerfile常用命令

命令	作用
FROM image_name:tag	定义了使用哪个基础镜像启动构建流程
MAINTAINER user_name	声明镜像的创建者
ENV key value	设置环境变量(可以写多条)
RUN command	是Dockerfile的核心部分(可以写多条)
ADD source_dir/file dest_dir/file	将宿主机的文件复制到容器内，如果是一个压缩文件，将会在复制后自动解压
COPY source_dir/file dest_dir/file	和ADD相似，但是如果有压缩文件并不能解压
WORKDIR path_dir	设置工作目录

```

1 # 1、创建目录
2 mkdir -p /usr/local/dockerjdk8
3 cd /usr/local/dockerjdk8
4
5 # 2、下载jdk-8u202-linux-x64.tar.gz并上传到服务器（虚拟机）中的/usr/local/dockerjdk8目录
6
7 # 3、在/usr/local/dockerjdk8目录下创建Dockerfile文件，文件内容如下：
8 vi Dockerfile
9
10 FROM centos:7
11 MAINTAINER ITCAST
12 WORKDIR /usr
13 RUN mkdir /usr/local/java
14 ADD jdk-8u202-linux-x64.tar.gz /usr/local/java/
15 ENV JAVA_HOME /usr/local/java/jdk1.8.0_202
16 ENV JRE_HOME $JAVA_HOME/jre
17 ENV CLASSPATH $JAVA_HOME/lib/rt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib:$CLASSPATH
18 ENV PATH $JAVA_HOME/bin:$PATH
19
20 # 4、执行命令构建镜像；不要忘了后面的那个 .
21 docker build -t='jdk1.8' .
22
23 # 5、查看镜像是否建立完成
24 docker images

```

Docker私有仓库搭建

私有仓库搭建步骤：

```

1 # 1、拉取私有仓库镜像
2 docker pull registry
3
4 # 2、启动私有仓库容器
5 docker run -di --name=registry -p 5000:5000 registry
6
7 # 3、打开浏览器 输入地址http://宿主机ip:5000/v2/_catalog，看到{"repositories":[]}
8 # 表示私有仓库搭建成功
9
10 # 4、修改daemon.json
11 vi /etc/docker/daemon.json
12
13 # 在上述文件中添加一个key，保存退出。此步用于让 docker 信任私有仓库地址；注意将宿主机ip修改为自己宿主
14 # 机真实ip
15 {"insecure-registries":["宿主机ip:5000"]}
16
17 # 5、重启docker 服务
18 systemctl restart docker
19
20 docker start registry

```

将本地镜像打标签（标记本地镜像为一个私有仓库中的镜像）；将打了标签的镜像推送到私有仓库。

8.2. 将镜像上传至私有仓库

操作步骤：

```
1 # 1、标记镜像为私有仓库的镜像
2 docker tag jdk1.8 宿主机IP:5000/jdk1.8
3
4 # 2、再次启动私有仓库容器
5 docker restart registry
6
7 # 3、上传标记的镜像
8 docker push 宿主机IP:5000/jdk1.8
9
10 # 4、输入网址查看仓库效果
11
```

```
[root@itcast dockerjdk8]# docker tag jdk1.8 192.168.12.135:5000/jdk1.8
[root@itcast dockerjdk8]# docker start registry
registry
[root@itcast dockerjdk8]# docker push 192.168.12.135:5000/jdk1.8
The push refers to repository [192.168.12.135:5000/jdk1.8]
2d7e8c06b38c: Pushed
f388e0699b47: Pushed
```

8.3.1. 私有仓库所在服务器拉取镜像

若是在私有仓库所在的服务器上去拉取镜像；那么直接执行如下命令：

```
1 # 因为私有仓库所在的服务器上已经存在相关镜像；所以先删除；请指定镜像名，不是id
2 docker rmi 服务器ip:5000/jdk1.8
3
4
5
6 #拉取镜像
7 docker pull 服务器ip:5000/jdk1.8
8
9 #可以通过如下命令查看 docker 的信息；了解到私有仓库地址
10 docker info
```

```
[root@itcast ~]# docker images
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
```

| 在仓库所在服务器上去拉取镜像是比较少有的操作；作为了解即可。

8.3.2. 其它服务器拉取私有仓库镜像

大多数情况下，都是某台服务器部署了私有镜像仓库之后；到其它服务器上从私有仓库中拉取镜像，若要拉取私有仓库镜像需要去修改docker的配置文件，设置启动时候的仓库地址。

```
1 # 打开配置文件
2 vi /usr/lib/systemd/system/docker.service
3
4 # 在打开的上述文件中按照下面的图，添加如下的内容；注意修改下面内容中的ip地址
5 --add-registry=192.168.12.135:5000 --insecure-registry=192.168.12.135:5000 \
6
7 # 修改完后需要重新加载docker配置文件并重启docker
8 systemctl daemon-reload
9 systemctl restart docker
```

在重启之后；那么则可以去拉取私有仓库中的镜像：

```
1 # 执行拉取镜像命令并查看
2 docker pull jdk1.8
3
4 docker images
```