# Docker

## Docker简介

### 背景

开发和运维之间因为环境不同而导致的矛盾

集群环境下每台机器部署相同的应用

DevOps(Development and Operations)

### 简介

Docker是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。

Docker是世界领先的软件容器平台。开发人员利用 Docker 可以消除协作编码时“在我的机器上可正常工作”的问题。运维人员利用 Docker 可以在隔离容器中并行运行和管理应用，获得更好的计算密度。企业利用 Docker 可以构建敏捷的软件交付管道，以更快的速度、更高的安全性和可靠的信誉为 Linux 和 Windows Server 应用发布新功能。

### Docker优点

简化程序： Docker 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，便可以实现虚拟化。Docker改变了虚拟化的方式，使开发者可以直接将自己的成果放入Docker中进行管理。方便快捷已经是 Docker的最大优势，过去需要用数天乃至数周的 任务，在Docker容器的处理下，只需要数秒就能完成

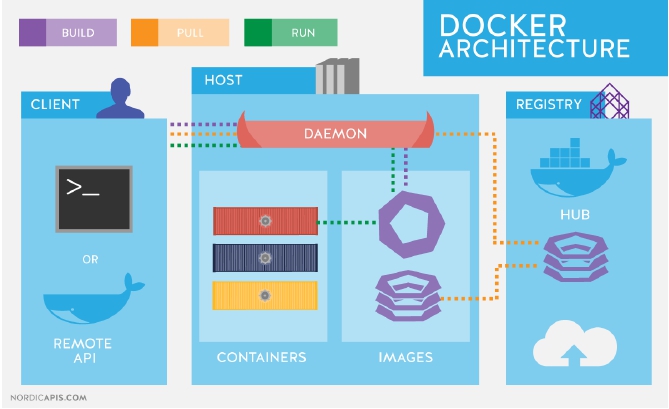
避免选择恐惧症： 如果你有选择恐惧症，还是资深患者。Docker 帮你 打包你的纠结！比如 Docker 镜像；Docker镜像中包含了运行环境和配置，所以 Docker 可以简化部署多种应用实例工作。比如 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

节省开支： 一方面，云计算时代到来，使开发者不必为了追求效果而配置高额的硬件，Docker 改变了高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化

的方式。

## Docker 架构

Docker使用C/S架构，Client通过接口与Server进程通信实现容器的构建，运行和发布，如图：



### Host（宿主机）

安装了Docker程序，并运行了Docker daemon的主机。

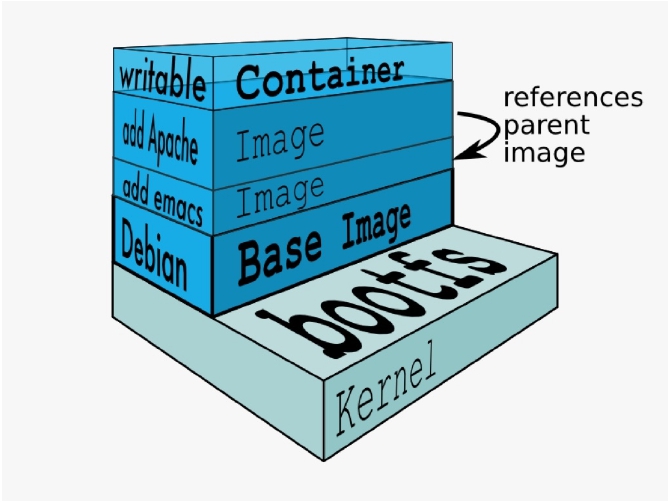
* Docker daemon(Docker 守护进程)：

运行在宿主机上，Docker守护进程，用户通过Docker client(Docker命令)与Docker daemon交互。

* Images(镜像)：

将软件环境打包好的模板，用来创建容器的，一个镜像可以创建多个容器。

* 镜像分层结构：



* 位于下层的镜像称为父镜像(Parent Image)，最底层的称为基础镜像(Base Image)。
* 最上层为“可读写”层，其下的均为“只读”层。
* AUFS:

advanced multi-layered uniﬁcation ﬁlesystem：高级多层统一文件系统

用于为Linux文件系统实现“联合挂载”

AUFS是之前的UnionFS的重新实现

Docker最初使用AUFS作为容器文件系统层

AUFS的竞争产品是overlayFS，从3.18开始被合并入Linux内核

Docker的分层镜像，除了AUFS，Docker还支持btrfs，devicemapper和vfs等

* Containers(容器)：

Docker的运行组件，启动一个镜像就是一个容器，容器与容器之间相互隔离，并且互不影响。

### Docker Client（客户端）

Docker命令行工具，用户是用Docker Client与Docker daemon进行通信并返回结果给用户。也可以使用其他工具通过Docker Api 与Docker daemon通信。

### Registry（仓库服务注册）

经常会和仓库(Repository)混为一谈，实际上Registry上可以有多个仓库，每个仓库可以看成是一个用户，一个用户的仓库放了多个镜像。仓库分为了公开仓库(Public Repository)和私有仓库(Private Repository)，最大的公开仓库是官方的Docker Hub，国内也有如阿里云、时速云等，可以给国内用户提供稳定快速的服务。用户也可以在本地网络内创建一个私有仓库。当用户创建了自己的镜像之后就可以使用 push 命令将它上传到公有或者私有仓库，这样下次

在另外一台机器上使用这个镜像时候，只需要从仓库上 pull 下来就可以了。

## Docker安装

Docker 提供了两个版本：社区版 (CE) 和企业版 (EE)。

### 操作系统要求

以Centos7为例，且Docker 要求操作系统必须为64位，且centos内核版本为3.1及以上。

查看系统内核版本信息：

uname -r

### 准备

卸载旧版本：

yum remove docker docker-common docker-selinux docker-engine

yum remove docker-ce

卸载后将保留 /var/lib/docker 的内容（镜像、容器、存储卷和网络等）。

rm -rf /var/lib/docker

1.安装依赖软件包

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

#安装前可查看device-mapper-persistent-data和lvm2是否已经安装

rpm -qa|grep device-mapper-persistent-data

rpm -qa|grep lvm2

2.设置yum源

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

3.更新yum软件包索引

yum makecache fast

### 安装

安装最新版本docker-ce:

yum install docker-ce –y

#安装指定版本docker-ce可使用以下命令查看

yum list docker-ce.x86\_64 --showduplicates | sort –r

# 安装完成之后可以使用命令查看

docker version

### 配置镜像加速

这里使用阿里云的免费镜像加速服务，也可以使用其他如时速云、网易云等

1.注册登录开通阿里云[容器镜像服务](https://cr.console.aliyun.com/cn-hangzhou/repositories)

2.查看控制台，招到镜像加速器并复制自己的加速器地址

3.找到/etc/docker目录下的daemon.json文件，没有则直接 vi daemon.json

4.加入以下配置

#填写自己的加速器地址

{

"registry-mirrors": ["https://zfzbet67.mirror.aliyuncs.com"]

}

5.通知systemd重载此配置文件；

systemctl daemon-reload

6.重启docker服务

systemctl restart docker

## Docker常用操作

输入 docker 可以查看Docker的命令用法，输入 docker COMMAND --help 查看指定命令详细用法。

### 镜像常用操作

查找镜像：

docker search 关键词

#搜索docker hub网站镜像的详细信息

下载镜像：

docker pull 镜像名:TAG

# Tag表示版本，有些镜像的版本显示latest，为最新版本

查看镜像

docker images

# 查看本地所有镜像

删除镜像：

docker rmi -f 镜像ID或者镜像名:TAG

# 删除指定本地镜像

# -f 表示强制删除

获取元信息：

docker inspect 镜像ID或者镜像名:TAG

# 获取镜像的元信息，详细信息

### 容器常用操作

运行：

docker run --name 容器名 -i -t -p 主机端口:容器端口 -d -v 主机目录:容器目录:ro 镜像ID或镜像名:TAG

# --name 指定容器名，可自定义，不指定自动命名

# -i 以交互模式运行容器

# -t 分配一个伪终端，即命令行，通常-it组合来使用

# -p 指定映射端口，讲主机端口映射到容器内的端口

# -d 后台运行容器

# -v 指定挂载主机目录到容器目录，默认为rw读写模式，ro表示只读

容器列表：

docker ps -a -q

# docker ps查看正在运行的容器

# -a 查看所有容器（运行中、未运行）

# -q 只查看容器的ID

启动容器：

docker start 容器ID或容器名

停止容器：

docker stop 容器ID或容器名

删除容器：

docker rm -f 容器ID或容器名

# -f 表示强制删除

查看日志

docker logs 容器ID或容器名

进入正在运行容器：

docker exec -it 容器ID或者容器名 /bin/bash

# 进入正在运行的容器并且开启交互模式终端

# /bin/bash是固有写法，作用是因为docker后台必须运行一个进程，否则容器就会退出，在这里表示启动容器后启动bash。

# 也可以用docker exec在运行中的容器执行命令

拷贝文件：

docker cp 主机文件路径 容器ID或容器名:容器路径 #主机中文件拷贝到容器中

docker cp 容器ID或容器名:容器路径 主机文件路径 #容器中文件拷贝到主机中

获取容器元信息：

docker inspect 容器ID或容器名

### 操作实例

docker pull mysql:5.7

#创建三个要挂载的目录

mkdir -p /my/mysql/conf

mkdir -p /my/mysql/data

mkdir -p /my/mysql/logs

#复制文件 并修改字符

docker cp mysql:/etc/mysql/mysql.conf.d/mysqld.cnf /my/mysql/conf/

vi /my/mysql/conf/mysqld.conf

character-set-server=utf8

#最终启动命令

docker run \

--name mysql \

-p 3306:3306 \

-v /my/mysql/conf:/etc/mysql/mysql.conf.d/ \

-v /my/mysql/data:/var/lib/mysql \

-v /my/mysql/logs:/logs \

-e MYSQL\_ROOT\_PASSWORD=root \

-d mysql:5.7

## Docker 生成镜像

有时候从Docker镜像仓库中下载的镜像不能满足要求，我们可以基于一个基础镜像构建一个自己的镜像

构建两种方式：

更新镜像：使用 docker commit 命令

构建镜像：使用 docker build 命令，需要创建Dockerﬁle文件

### 更新镜像

先使用基础镜像创建一个容器，然后对容器内容进行更改，然后使用 docker commit 命令提交为一个新的镜像（以tomcat为例）。

1.根据基础镜像，创建容器

docker run --name mytomcat -p 80:8080 -d tomcat

2.修改容器内容

docker exec -it mytomcat /bin/bash

cd webapps/ROOT

rm -f index.jsp

echo hello world > index.html

exit

3.提交为新镜像

docker commit -m="描述消息" -a="作者" 容器ID或容器名 镜像名:TAG

# 例:

# docker commit -m="修改了首页" -a="User01" mytomcat user01/tomcat:v1.0

4.使用新镜像运行容器

docker run --name tom -p 8080:8080 -d user01/tomcat:v1.0

### Dockerfile构建镜像

#### 什么是DockerFile

Dockerﬁle is nothing but the source code for building Docker images

Docker can build images automatically by reading the instructions from a Dockerﬁle

A Dockerﬁle is a text document that contains all the commands a user could call on the command lineto assemble an image

Using docker build users can create an automated build that executes several command-line instructionsin succession



#### Dockerfile格式

Format：

#Comment

INSTRUCTION arguments

The instruction is not case-sensitive

However,convention is for them to be UPPERCASE to distinguish them from arguments more easily

Docker runs instructions in a Dockerﬁle in order

The ﬁrst instruction must be 'FROM' in order to specify the Base Image from which you are building

#### 使用Dockerfile构建镜像实例

一、准备

1.把你的springboot项目打包成可执行jar包

2.把jar包上传到Linux服务器

二、构建

1.在jar包路径下创建Dockerﬁle文件 vi Dockerfile

# 指定基础镜像，本地没有会从dockerHub pull下来

FROM java:8

#作者

MAINTAINER user01

# 把可执行jar包复制到基础镜像的根目录下

ADD luban.jar /luban.jar

# 镜像要暴露的端口，如要使用端口，在执行docker run命令时使用-p生效

EXPOSE 80

# 在镜像运行为容器后执行的命令

ENTRYPOINT ["java","-jar","/luban.jar"]

2.使用 docker build 命令构建镜像，基本语法

docker build -t user01/mypro:v1 .

# -f指定Dockerfile文件的路径

# -t指定镜像名字和TAG

# .指当前目录，这里实际上需要一个上下文路径

三、运行

运行自己的SpringBoot镜像

docker run --name pro -p 80:80 -d 镜像名:TAG

#### Dockerfile常用命令

##### FROM

FROM指令是最重要的一个并且必须为Dockerﬁle文件开篇的第一个非注释行，用于为镜像文件构建过程指定基础镜像，后续的指令运行于此基础镜像提供的运行环境。

这个基础镜像可以是任何可用镜像，默认情况下docker build会从本地仓库找指定的镜像文件，如果不存在就会从Docker Hub上拉取

语法：

FROM <image>

FROM <image>:<tag>

FROM <image>@<digest>

##### MAINTAINER(depreacted)

Dockerﬁle的制作者提供的本人详细信息

Dockerﬁle不限制MAINTAINER出现的位置，但是推荐放到FROM指令之后

语法：

MAINTAINER <name>

name可以是任何文本信息，一般用作者名称或者邮箱

##### LABEL

给镜像指定各种元数据

语法：

LABEL <key>=<value> <key>=<value> <key>=<value>..

一个Dockerﬁle可以写多个LABEL，但是不推荐这么做，Dockerﬁle每一条指令都会生成一层镜像，如果LABEL太长可以使用\符号换行。构建的镜像会继承基础镜像的LABEL，并且会去掉重复的，但如果值不同，则后面的值会覆盖前面的值。

##### COPY

用于从宿主机复制文件到创建的新镜像文件

语法：

COPY <src>...<dest>

COPY ["<src>",..."<dest>"]

# <src>：要复制的源文件或者目录，可以使用通配符

# <dest>：目标路径，即正在创建的image的文件系统路径；建议<dest>使用绝对路径，否则COPY指令则以WORKDIR为

其起始路径

注意：如果你的路径中有空白字符，通常会使用第二种格式

规则：

<src> 必须是build上下文中的路径，不能是其父目录中的文件

如果 <src> 是目录，则其内部文件或子目录会被递归复制，但 <src> 目录自身不会被复制

如果指定了多个 <src> ，或在 <src> 中使用了通配符，则 <dest> 必须是一个目录，则必须以/符号结尾

如果 <dest> 不存在，将会被自动创建，包括其父目录路径

##### ADD

基本用法和COPY指令一样，ADD支持使用TAR文件和URL路径

语法：

ADD <src>...<dest>

ADD ["<src>",..."<dest>"]

规则：

和COPY规则相同

如果 <src> 为URL并且 <dest> 没有以/结尾，则 <src> 指定的文件将被下载到 <dest>

如果 <src> 是一个本地系统上压缩格式的tar文件，它会展开成一个目录；但是通过URL获取的tar文件不会自动展开

如果 <src> 有多个，直接或间接使用了通配符指定多个资源，则 <dest> 必须是目录并且以/结尾

##### WORKDIR

用于为Dockerﬁle中所有的RUN、CMD、ENTRYPOINT、COPY和ADD指定设定工作目录，只会影响当前WORKDIR

之后的指令。

语法：

WORKDIR <dirpath>

在Dockerﬁle文件中，WORKDIR可以出现多次，路径可以是相对路径，但是它是相对于前一个WORKDIR指令指定的路径另外，WORKDIR可以是ENV指定定义的变量

##### VOLUME

用来创建挂载点，可以挂载宿主机上的卷或者其他容器上的卷

语法：

VOLUME <mountpoint>

VOLUME ["<mountpoint>"]

不能指定宿主机当中的目录，宿主机挂载的目录是自动生成的

##### EXPOSE

用于给容器打开指定要监听的端口以实现和外部通信

语法：

EXPOSE <port>[/<protocol>] [<port>[/<protocol>]...]

<protocol> 用于指定传输层协议，可以是TCP或者UDP，默认是TCP协议

EXPOSE可以一次性指定多个端口，例如： EXPOSE 80/tcp 80/udp

##### ENV

用来给镜像定义所需要的环境变量，并且可以被Dockerﬁle文件中位于其后的其他指令(如ENV、ADD、COPY等)所调用，调用格式：$variable\_name或者${variable\_name}

语法：

ENV <key> <value>

ENV <key>=<value>...

第一种格式中， <key> 之后的所有内容都会被视为 <value> 的组成部分，所以一次只能设置一个变量

第二种格式可以一次设置多个变量，如果 <value> 当中有空格可以使用\进行转义或者对 <value> 加引号进行标识；

另外\也可以用来续行

##### ARG

用法同ENV

语法：

ARG <name>[=<default value>]

指定一个变量，可以在docker build创建镜像的时候，使用 --build-arg <varname>=<value> 来指定参数

##### RUN

用来指定docker build过程中运行指定的命令

语法：

RUN <command>

RUN ["<executable>","<param1>","<param2>"]

第一种格式里面的参数一般是一个shell命令，以 /bin/sh -c 来运行它

第二种格式中的参数是一个JSON格式的数组，当中 <executable> 是要运行的命令，后面是传递给命令的选项或者参数；但是这种格式不会用 /bin/sh -c 来发起，所以常见的shell操作像变量替换和通配符替换不会进行；如果你运行的命令依赖shell特性，可以替换成类型以下的格式

RUN ["/bin/bash","-c","<executable>","<param1>"]

##### CMD

容器启动时运行的命令

语法：

CMD <command>

CMD ["<executable>","<param1>","<param2>"]

CMD ["<param1>","<param2>"]

前两种语法和RUN相同

第三种语法用于为ENTRYPOINT指令提供默认参数

RUN和CMD区别：

RUN指令运行于镜像文件构建过程中，CMD则运行于基于Dockerﬁle构建出的新镜像文件启动为一个容器的时候CMD指令的主要目的在于给启动的容器指定默认要运行的程序，且在运行结束后，容器也将终止；不过，CMD命令可以被docker run的命令行选项给覆盖

Dockerﬁle中可以存在多个CMD指令，但是只有最后一个会生效

##### ENTRYPOINT

类似于CMD指令功能，用于给容器指定默认运行程序

语法：

ENTRYPOINT<command>

ENTRYPOINT["<executable>","<param1>","<param2>"]

和CMD不同的是ENTRYPOINT启动的程序不会被docker run命令指定的参数所覆盖，而且，这些命令行参数会被当做参数传递给ENTRYPOINT指定的程序(但是，docker run命令的--entrypoint参数可以覆盖ENTRYPOINT)

docker run命令传入的参数会覆盖CMD指令的内容并且附加到ENTRYPOINT命令最后作为其参数使用.同样，Dockerﬁle中可以存在多个ENTRYPOINT指令，但是只有最后一个会生效.

Dockerﬁle中如果既有CMD又有ENTRYPOINT，并且CMD是一个完整可执行命令，那么谁在最后谁生效

##### ONBUILD

用来在Dockerﬁle中定义一个触发器

语法：

ONBUILD <instruction>

Dockerﬁle用来构建镜像文件，镜像文件也可以当成是基础镜像被另外一个Dockerﬁle用作FROM指令的参数

在后面这个Dockerﬁle中的FROM指令在构建过程中被执行的时候，会触发基础镜像里面的ONBUILD指令

ONBUILD不能自我嵌套，ONBUILD不会触发FROM和MAINTAINER指令

在ONBUILD指令中使用ADD和COPY要小心，因为新构建过程中的上下文在缺少指定的源文件的时候会失败

## 本地镜像发布到阿里云

有时候需要共享镜像或者习惯使用自己定义的镜像，可以注册私有仓库，国内推荐使用阿里云

步骤：

1.登录阿里云容器镜像服务：<https://cr.console.aliyun.com/cn-hangzhou/repositories>

2.将镜像推送到阿里云

# 登录阿里云的docker仓库

$ sudo docker login --username=[用户名] registry.cn-hangzhou.aliyuncs.com

# 创建指定镜像的tag，归入某个仓库

$ sudo docker tag [镜像ID] registry.cn-hangzhou.aliyuncs.com/user01/user01:[镜像版本号]

# 讲镜像推送到仓库

$ sudo docker push registry.cn-hangzhou.aliyuncs.com/user01/user01:[镜像版本号]

3.拉取镜像

docker pull registry.cn-hangzhou.aliyuncs.com/coldest7/mytom:v1

## Docker网络

Docker允许通过外部访问容器或容器互联的方式来提供网络服务。

安装Docker时，会自动安装一块Docker网卡称为docker0，用于Docker各容器及宿主机的网络通信，网段为172.0.0.1。

Docker网络中有三个核心概念：沙盒（Sandbox）、网络（Network）、端点（Endpoint）。

沙盒，提供了容器的虚拟网络栈，也即端口套接字、IP路由表、防火墙等内容。隔离容器网络与宿主机网络，形成了完全独立的容器网络环境。

网络，可以理解为Docker内部的虚拟子网，网络内的参与者相互可见并能够进行通讯。Docker的虚拟网络和宿主机网络是存在隔离关系的，其目的主要是形成容器间的安全通讯环境。

端点，位于容器或网络隔离墙之上的洞，主要目的是形成一个可以控制的突破封闭的网络环境的出入口。当容器的端点与网络的端点形成配对后，就如同在这两者之间搭建了桥梁，便能够进行数据传输了。

### Docker的四种网络

Docker服务在启动的时候会创建三种网络，bridge、host和none，还有一种共享容器的模式container

* Bridge

桥接模式，主要用来对外通信的，docker容器默认的网络使用的就是bridge。

使用bridge模式配置容器自定的网络配置

# 配置容器的主机名

docker run --name t1 --network bridge -h [自定义主机名] -it --rm busybox

# 自定义DNS

docker run --name t1 --network bridge --dns 114.114 -it --rm busybox

# 给host文件添加一条

docker run --name t1 --network bridge --add-host [hostname]:[ip] -it --rm busybox

* Host

host类型的网络就是主机网络的意思，绑定到这种网络上面的容器，内部使用的端口直接绑定在主机上对应的端口，

而如果容器服务没有使用端口，则无影响。

* None

从某种意义上来说，none应该算不上网络了，因为它不使用任何网络，会形成一个封闭网络的容器

* container

共享另外一个容器的network namespace，和host模式差不多，只是这里不是使用宿主机网络，而是使用的容器网络

## 开放端口

Docker0为NAT桥，所以容器一般获得的是私有网络地址

给docker run命令使用-p选项即可实现端口映射，无需手动添加规则

* -p 选项的使用

-p <containerPort>

将指定的容器端口映射到主机所有地址的一个动态端口

-p <hostPort>:<containerPort>

将容器端口 <containerPort> 映射到指定的主机端口 <hostPort>

-p <ip>::<containerPort>

将指定的容器端口 <containerPort> 映射到主机指定 <ip> 的动态端口

-p <ip>:<hostPort>:<containerPort>

将指定的容器端口 <containerPort> 映射至主机指定 <ip> 的端口 <hostPort>

* 动态端口指随机端口，可以使用docker port命令查看具体映射结果
* -P 暴露所有端口（所有端口指构建镜像时EXPOSE的端口）

自定义docker0桥的网络属性信息：/etc/docker/daemon.json文件

{

"bip": "192.168.1.5/24",

"fixed-cidr": "10.20.0.0/16",

"fixed-cidr-v6": "2001:db8::/64",

"mtu": 1500,

"default-gateway": "10.20.1.1",

"default-gateway-v6": "2001:db8:abcd::89",

"dns": ["10.20.1.2","10.20.1.3"]

}

核心选项为bip，即bridge ip之意，用于指定docker0桥自身的IP地址；其它选项可通过此地址计算得出

远程连接

创建自定义的桥

docker network create -d bridge --subnet "172.26.0.0/16" --gateway "172.26.0.1" mybr0

## Docker compose

从上一节课我们了解到可以使用一个Dockerﬁle模板文件来快速构建一个自己的镜像并运行为应用容器。但是在平时工作的时候，我们会碰到多个容器要互相配合来使用的情况，比如数据库加上咱们Web应用等等。这种情况下，每次都要一个一个启动容器设置命令变得麻烦起来，所以Docker Compose诞生了。

### Dockercompse简介

Compose的作用是“定义和运行多个Docker容器的应用”。使用Compose，你可以在一个配置文件（yaml格式）中配置你应用的服务，然后使用一个命令，即可创建并启动配置中引用的所有服务。

Compose中两个重要概念：

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。

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文件中定义。

### Dockercompse安装

Compose支持三平台Windows、Mac、Linux，安装方式各有不同。我这里使用的是Linux系统，其他系统安装方法

可以参考官方文档和开源GitHub链接：

Docker Compose官方文档链接：<https://docs.docker.com/compose>

Docker Compose GitHub链接：<https://github.com/docker/compose>

Linux上有两种安装方法，Compose项目是用Python写的，可以使用Python-pip安装，也可以通过GitHub下载二进制文件进行安装。

* 通过Python-pip安装

1.安装Python-pip

um install -y epel-release

yum install -y python-pip

2.安装docker-compose

pip install docker-compose

3.验证是否安装

docker-compose version

4.卸载

pip uninstall docker-compose

* 通过GitHub链接下载安装

非ROOT用户记得加sudo

1.通过GitHub获取下载链接，以往版本地址：<https://github.com/docker/compose/releases>

curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname-s)-$(uname -m)" -o /usr/local/bin/docker-compose

2.给二进制下载文件可执行的权限

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

3.可能没有启动程序，设置软连接，比如:

ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

4.验证是否安装

docker-compose version

5.卸载

如果是二进制包方式安装的，删除二进制文件即可。

rm /usr/local/bin/docker-compose

简单实例

Compose的使用非常简单，只需要编写一个docker-compose.yml，然后使用docker-compose 命令操作即可。

docker-compose.yml描述了容器的配置，而docker-compose 命令描述了对容器的操作

1.我们使用一个微服务项目先来做一个简单的例子，首先创建一个compose的工作目录，然后创建一个eureka文件夹，里面放可执行jar包和编写一个Dockerﬁle文件，目录结构如下：

compose

eureka

Dockerfile

eureka-server-2.0.2.RELEASE.jar

2.在compose目录创建模板文件docker-compose.yml文件并写入以下内容

version: '1'

services:

eureka:

build: ./eureka

ports:

- 3000:3000

expose:

- 3000

### DockerCompose常用命令

##### image

指定镜像名称或者镜像id，如果该镜像在本地不存在，Compose会尝试pull下来。

示例：

image: java:8

##### build

指定Dockerﬁle文件的路径。可以是一个路径，例如：build: ./dir

也可以是一个对象，用以指定Dockerﬁle和参数，例如：

build: context: ./dir dockerﬁle: Dockerﬁle-alternate args: buildno: 1

##### command

覆盖容器启动后默认执行的命令。

示例：

command: bundle exec thin -p 3000

也可以是一个list，类似于Dockerﬁle总的CMD指令，格式如下：

command: [bundle, exec, thin, -p, 3000]

##### links

链接到其他服务中的容器。可以指定服务名称和链接的别名使用SERVICE:ALIAS 的形式，或者只指定服务名称，示

例：

web: links: - db - db:database – redis

##### external\_links

表示链接到docker-compose.yml外部的容器，甚至并非Compose管理的容器，特别是对于那些提供共享容器或共同

服务。格式跟links类似，示例：

external\_links: - redis\_1 - project\_db\_1:mysql - project\_db\_1:postgresql

##### ports

暴露端口信息。使用宿主端口:容器端口的格式，或者仅仅指定容器的端口（此时宿主机将会随机指定端口），类似于docker run -p ，示例：

ports:

"3000"

"3000-3005"

"8000:8000"

"9090-9091:8080-8081"

"49100:22"

"127.0.0.1:8001:8001"

"127.0.0.1:5000-5010:5000-5010"

##### expose

暴露端口，只将端口暴露给连接的服务，而不暴露给宿主机，示例：

expose: - "3000" - "8000"

##### volumes

卷挂载路径设置。可以设置宿主机路径 （HOST:CONTAINER） 或加上访问模式 （HOST:CONTAINER:ro）。示例：

volumes:

Just specify a path and let the Engine create a volume

/var/lib/mysql

Specify an absolute path mapping

/opt/data:/var/lib/mysql

Path on the host, relative to the Compose ﬁle

./cache:/tmp/cache

User-relative path

~/conﬁgs:/etc/conﬁgs/:ro

Named volums

datavolume:/var/lib/mysql

##### volumes\_from

从另一个服务或者容器挂载卷。可以指定只读或者可读写，如果访问模式没有指定，则默认是可读写。示例：

volumes\_from:

service\_name

service\_name:ro

container:container\_name

container:container\_name:rw

##### environment

设置环境变量。可以使用数组或者字典两种方式。只有一个key的环境变量可以在运行Compose的机器上找到对应的

值，这有助于加密的或者特殊主机的值。示例：

environment: RACK\_ENV: development SHOW: 'true' SESSION\_SECRET: environment: -

RACK\_ENV=development - SHOW=true - SESSION\_SECRET

##### env\_ﬁle

从文件中获取环境变量，可以为单独的文件路径或列表。如果通过 docker-compose -f FILE 指定了模板文件，则env\_ﬁle 中路径会基于模板文件路径。如果有变量名称与 environment 指令冲突，则以envirment 为准。示例：

env\_ﬁle: .env env\_ﬁle: - ./common.env - ./apps/web.env - /opt/secrets.env

##### extends

继承另一个服务，基于已有的服务进行扩展。

##### net

设置网络模式。示例：

net: "bridge" net: "host" net: "none" net: "container:[service name or container name/id]"

##### dns

配置dns服务器。可以是一个值，也可以是一个列表。示例：

dns: 8.8.8.8 dns: - 8.8.8.8 - 9.9.9.9

##### dns\_search

配置DNS的搜索域，可以是一个值，也可以是一个列表，示例：

dns\_search: example.com dns\_search: - dc1.example.com - dc2.example.com

##### 其它

docker-compose.yml 还有很多其他命令，可以参考docker-compose.yml文件官方文档：

[https://docs.docker.com/compose/compose-ﬁle/](https://docs.docker.com/compose/compose-file/)

### 使用DockerCompose构建实例

使用docker-compose一次性来编排三个微服务:eureka服务(eureka-server-2.0.2.RELEASE.jar)、user服务(user-2.0.2.RELEASE.jar)、power服务(power-2.0.2.RELEASE.jar)

1.创建一个工作目录和docker-compose模板文件

2.工作目录下创建三个文件夹eureka、user、power，并分别构建好三个服务的镜像文件

以eureka的Dockerﬁle为例:

# 基础镜像

FROM java:8

# 作者

MAINTAINER user01

# 把可执行jar包复制到基础镜像的根目录下

ADD eureka-server-2.0.2.RELEASE.jar /eureka-server-2.0.2.RELEASE.jar

# 镜像要暴露的端口，如要使用端口，在执行docker run命令时使用-p生效

EXPOSE 3000

# 在镜像运行为容器后执行的命令

ENTRYPOINT ["java","-jar","/eureka-server-2.0.2.RELEASE.jar"]

目录文件结构：

compose

docker-compose.yml

eureka

Dockerfile

eureka-server-2.0.2.RELEASE.jar

user

Dockerfile

user-2.0.2.RELEASE.jar

power

Dockerfile

power-2.0.2.RELEASE.jar

3.编写docker-compose模板文件：

version: '1'

services:

eureka:

image: eureka:v1

ports:

- 8080:8080

user:

image: user:v1

ports:

- 8081:8081

power:

image: power:v1

ports:

- 8082:8082

4.启动微服务，可以加上参数-d后台启动

docker-compose up -d