镜像的搜索

杨翼臣

# 镜像的搜索

## 搜索命令

* docker search 名称

## 镜像的搜索命令

* 限制数量 docker images –limit 数量 镜像名称

# 镜像的下载

命令：docker pull 镜像名称[:TAG] tag不写默认拉最新。加时不用中括号

# 镜像推送

* 命令：docker

# 镜像—

## 镜像分层

docker的镜像是分层的。支持通过扩展现有镜像，创建新的镜像。 ### （联合文件系统）UniosFS： 分层、轻量级、高性能的文件系统，支持对文件系统的修改作为一次提交来一层层叠加。（相当于git？），支持将不同的目录挂载在同一个虚拟文件系统下。是docker镜像的基础。镜像可以通过分层来进行继承。基于基础镜像，可以制作各种具体的应用镜像。 把本来镜像需要的多个文件系统加载到一个文件系统中，把原镜像系统内需要的各种文件系统叠加起来。

### 镜像分层的好处

可以共享资源，方便复制迁移，为了资源能够在其他地方复用。像搭积木一样 比如说如果有多个镜像从相同的base镜像构建而来，那么docker host只需要在磁盘中保存一份base镜像；同时内存中也只需要加载一份base镜像，就可以为所有容器服务，而且镜像的每一层都可以被共享。

### 镜像层

镜像层是只能读的

## 镜像加载原理

bootfs(boot file system)主要包含bpotloader和kernel, bootloader主要是引导加载kernel, Linux刚启动时会加载bootfs文件系统，在Docker镜像的最底层是引导文件系统bootfs。这一层与我们典型的Linux/Unix 系统是一样的，包含boot加载器和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权己由bootfs转交给内核，此时系统也会卸载bootfs。 rootfs (root file system)，在bootfs之上。包含的就是典型 Linux 系统中的/dev, /proc,/bin, /etc 等标准目录和文件。rootfs就是各种不同的操作系统发行版，比如Ubuntu， Centos等等。

# 容器的操作

前提知识：docker只能建立在linux内核上运行。（还需要拉一个系统镜像） ## 新建容器 基本命令： docker run [OPTIONSJ IMAGE [COMMANDJ [ARG..]

### 新建容器中需要用到启动交互式容器

^576f87 操作指令：（options） - 为容器命名：–name=“容器名” - 让容器后台运行并返回容器id：-d，没有交互。 后台运行需要注意 - 让容器以交互模式运行：-i（interactive） - 为容器分配一个伪输入终端:-t（tyy） -i和-t合用，前台有伪终端，等待交互 前台交互的主目录接口一般用bash或者/bin/bash接到镜像名后面如： docker run - it ubuntu /bin/bash - 让容器随机映射端口：-P（大写） - 指定容器在本机上的应用端口映射： -p（小写）（通过哪个端口能访问他，放在哪个端口） - -p 6000:6000 前者是宿主机的端口，后面是容器内的端口

## 列出容器列表

^0349b9 - 基本命令：docker ps (正在运行的) - 其他命令： - 列出所有容器，保括历史、已停止的：-a - 显示最近创建的容器：-l - 显示最近创建的n个容器：-n - 静默模式，只显示容器编号：-q

## 退出容器

* 我要退出让容器停止: exit，退出后容器也停止。
* 我要退出容器容器仍然运行: ctrl+p+q ，mac自行摸索

## 启动容器

* 命令：docker start 容器id

## 在新建容器时后台模式启动

-d后会发现容器没有正在运行，而是退出。 原因：docker容器后台运行，必须要有一个前台进程。如果容器的命令并不是一直挂起的命令如（top，tail），就是会自动退出。机制问题。

## 停止容器

* 正常停止：docker stop
* 强制停止：docker kill 容器id 或者容器名

## 删除容器

先停止：docker stop 然后再docker rm 强制删除：docker rm -f 容器id 删除所有容器：docker rm -f $(docker ps -a -q)

## 查看容器日志

基本命令：docker log 容器id

## 查看容器内的系统进程

基本命令：docker top 容器id

## 查看容器内部细节

基本命令：docker inspect 容器id

## 进入容器内部

基本命令： docker exec -it 容器id - 重新进入：docker attach 容器id

区别： - attach是直接进入容器启动命令的终端，不会启动新的进程。当用exit退出时，容器会停止。 - exec 是在容器中打开一个新的终端，相当于新的窗口，用exit退出是不会导致之前容器本来运行的窗口停止的。

## 将容器内的文件拷贝到主机上

基本命令：docker cp 容器id：要拷贝的目录路径 宿主机的目地路径

## 容器的导入和导出

* 导出为镜像：docker export 容器id > 文件名.tar
* 导入为镜像： cat 文件名.tar | docker import -镜像用户名/镜像名：镜像版本号(自定义)

## 提交容器为镜像新副本

基本命令：docker commit-m=“提交的描述信息” -a=“作者” 容器id 要创建的目标镜像的名称:[标签名]

## 容器的数据挂载

[[Docker大笔记#^4a20c0|查看挂载在宿主机的哪]]

# 层

[[Pasted image 20230403164739.png|600]]

## 容器层

当容器层被启动后，一个新的可写层被加载到镜像的顶部，这一层通常被叫做“容器层”，容器层之下的都叫做“镜像层”，容器层是可写的。

# 镜像的发布

## 将本地镜像发布到阿里云

阿里云开发者平台，aliyun.com 找到容器和镜像服务； - 找到容器镜像服务 - 选择个人实例 - 创建命名空间和仓库名称 - 进入管理页面获得脚本 通过脚本修改内容进行推送操作

## 将本地镜像发布到私人仓库

### docker registry

#### 配置registry

1、 下载registry镜像 2、 运行私有仓库registry，相当于本地的docker hub 3、 docker run -d -p 5000:5000 -v /zyyuse/myregistry/:/tmp/registry –privileged=true registry 默认情况，仓库被创建在容器的var/lib/registry目录下，建议自行用容器卷映射，方便于宿主机联调

#### 如何取消docker默认不允许http推送的限制

在宿主机中cd到/etc/docker/daemon.json文件 新增代码： `“insecure-registries”:[“192.168.111.162:5000”]

json文件中那部分的registry-mirrors是镜像地址；可以用官方地址也能用国内的加速地址；

#### 查找私服上的镜像文件

4、 验证私服库上有什么镜像 curl-XGET 本地ip:加registry映射端口/v2/ \_catalog

#### 上传镜像到registry

##### 将镜像按标准命名规则进行规范

* docker tag 要规范的镜像名称:tag(可以使版本也可以是文字英文 ) Host:Port(自己宿主机的ip地址)/Repository（仓库）:Tag（要打的tag）

##### 推送

基本命令：docker push 规范命名后的镜像名称:tag

# 数据卷（持久化备份容器的数据）

## 提醒

Docker挂载主机目录访问如果出现cannot open directory .: Permission denied 解诀办法：在挂载日录后多加一个 –privileged=true 参数即可 如果是Centos7安全模块会比之前系统版本加强，不安全的会先禁止，所以目录挂载的情况被默认为不安全的行为，在SELinux里面挂载目录被禁止掉了，如果要开启，我们一般使用 –privileged=true 命令，扩大容器的权限解决挂载目录没有权限的问题，使用该参数，container内的root拥有真正的root权限，否则，container内的root只是外部的一个普通用户权限。

## 参数v

添加自定义的容器卷； 可以挂载多个硬盘；

## 卷

目录或者文件，存在与一个或多个容器中，被docker挂载到容器上，不属于联合文件系统，因此能够绕过Union File System提供一些用于持续存储或共享数据的特性。 为了让数据能够持久恶化，完全独立出容器的生存周期，相当于硬件换了，软件还没有坏，docker 不会在容器删除时删除其挂载的数据卷。

## 特点

1：数据卷可在容器之间共享或重用数据 2：卷中的更改可以直接实时生效， 3：数据卷中的更改不会包含在镜像的更新中 4：数据卷的生命周期一直持续到没有容器使用它为止

## 启动一个映射目录的容器

命令：docker run -it –privileged=true-v /目标挂载宿主机绝对路径目录:/容器内要挂载的目录 镜像名 （目录可以自定义，没有的话机器会自动创建） 这样在主机上相应目录的文件和容器中目录文件能互相共享和共用。

## 查看容器挂载在哪个宿主目录

命令：docker inspect； 查看mounts就能看到挂载的详细数据。 ^4a20c0

## 数据权限控制

* 默认权限是可读可写的:默认的挂载映射的容器命令后面是省略掉docker run -it –privileged=true-v /宿主机绝对路径目录:/容器内目录 **:rw** 镜像名m，省略掉了rw。
* 让容器只可读不可写，宿主机可读可写，在宿主机上读写完让容器只能读：

将：rw改成ro（read only）；

## 卷的继承和共享

继承： docker run - it – privileged=true **–volumes -from u1** –name u2 ubuntu 让该容器直接用某个容器的映射在宿主机的目录

# 常用服务安装

## mysql的简单安装(需要学习mysql)

**此方法没有使用容器卷，删容器实例数据全没。** ### 拉取镜像 docker pull mysql ### 运行镜像

$ docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:tag

### 进入mysql

mysql -uroot -p

根据提示输入镜像启动时设置的密码

## 正常带容器卷的mysql安装

### 创建容器实例

docker run -d -p 3306:3306 --privileged=true   
-v 主机目录/mysql/log:/var/log/mysql  
-v 主机目录/mysql/data:/var/lib/mysql  
-v 主机目录/mysql/conf:/etc/mysql/conf.d  
-e MYSQL\_ROOT\_PASSWORD=密码  
-- name 名字  
mysql:tag

### 解决中文乱码问题

进入主机目录：主机目录/mysql/conf下创建my.cnf文件，添加文本内容：

[client]  
default\_character\_set=utf8  
[mysqld]  
colloation\_server = utf8\_general\_ci  
character\_set\_server = utf8

这个文件会通过容器卷同步到容器内的mysql容器实例 #### 重启容器检查是否正常 SHOW VARIABLES LIKE ‘character%’; 看输出是否为utf8而不是latin；

### 容器删除后如何重新挂载使用主机目录数据

使用相同run命令即可，确保容器卷的映射配置是对的即可。

docker run -d -p 3306:3306 --privileged=true   
-v 主机目录/mysql/log:/var/log/mysql  
-v 主机目录/mysql/data:/var/lib/mysql  
-v 主机目录/mysql/conf:/etc/mysql/conf.d  
-e MYSQL\_ROOT\_PASSWORD=密码  
-- name 名字  
mysql:tag

## mysql主从服务器的搭建

### 新建一个主（master）服务器容器实例端口映射给3307

docker run -d -p 3307:3306 --privileged=true   
-v 主机目录/mysql/log:/var/log/mysql  
-v 主机目录/mysql/data:/var/lib/mysql  
-v 主机目录/mysql/conf:/etc/mysql/conf.d  
-e MYSQL\_ROOT\_PASSWORD=密码  
-- name 名字  
mysql:tag

#### 配置my.cnf文件

* 进入宿主机目录：/mydata/mysql-master/conf 新建my.cnf
* 粘贴文本：
* [[Pasted image 20230404175122.png|600]]

[mysqld]  
##设置server\_id，同一局城网中需要唯一  
server\_id =101  
##指定不需要同步的数据库名称  
binlog-ignore-db=mysql  
##开启二进制日志功能  
log-bin=mall-mysql-bin |  
##设置二进制日志使用内存大小（事务）  
binlog\_cache\_size=1M  
##设置使用的二进制日志格式 (mixed,statement,row)  
binlog\_format=mixed  
##二进制日志过期清理时间。默认值为0，表示不自动清理。  
expire\_logs\_days=7  
##跳过主从复制中遇到的所有错误或指定类型的错误，避免slave端复制中断。  
##如：1062错误是指一些主键重复，1032错误是因为主从数据库数据不一致  
slave\_skip\_errors=1062  
  
  
?????以下内容是主机需要的？  
##relay\_log配置中继日志  
relay\_log=mall-mysql-relay-bin  
##log\_slave\_updates表示slave将复制事件写进自己的二进制日志  
log\_slave\_updates=1  
##slave设置为只读（具有super权限的用户除外）  
read\_only=1

#### 重启主服务器实例

#### 进入主服务器容器实例创建数据同步用户

* 建立一个用户，并给这个用户授权

CREATE USER 'slave'@'%' IDENTIFIED BY '123456';  
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO'slave' @ '%'

### 新建一个从服务器,端口映射给3308

docker run -d -p 3308:3306 --privileged=true   
-v 主机目录/mysql/log:/var/log/mysql  
-v 主机目录/mysql/data:/var/lib/mysql  
-v 主机目录/mysql/conf:/etc/mysql/conf.d  
-e MYSQL\_ROOT\_PASSWORD=密码  
-- name 名字  
mysql:tag

#### 进入从服务器目录新建配置文件

* 目录路径：/mydata/mysql-slave/conf下新建一个my.cnf
* 复制粘贴以下的内容为my.cnf文件的内容

[mysqld]  
##设置server\_id，同一局城网中需要唯一  
server\_id =102 ##修改为102  
##指定不需要同步的数据库名称  
binlog-ignore-db=mysql  
##开启二进制日志功能  
log-bin=mall-mysql-bin |  
##设置二进制日志使用内存大小（事务）  
binlog\_cache\_size=1M  
##设置使用的二进制日志格式 (mixed,statement,row)  
binlog\_format=mixed  
##二进制日志过期清理时间。默认值为0，表示不自动清理。  
expire\_logs\_days=7  
##跳过主从复制中遇到的所有错误或指定类型的错误，避免slave端复制中断。  
##如：1062错误是指一些主键重复，1032错误是因为主从数据库数据不一致  
slave\_skip\_errors=1062  
##relay\_log配置中继日志  
relay\_log=mall-mysql-relay-bin  
##log\_slave\_updates表示slave将复制事件写进自己的二进制日志  
log\_slave\_updates=1  
##slave设置为只读（具有super权限的用户除外）  
read\_only=1

#### 重启从机实例

### 回到主服务器查看主从服务器的同步状态

* 进入主服务器的sql界面得到mysql>命令，输入命令show master status [[Pasted image 20230407152642.png|600]]

### 进入从服务器，完成下列配置

#### 绑定主服务器

change master to master\_host='宿主机ip', master\_user='slave',master\_password='123456', master\_port=3307,  
master\_log\_file='mall-mysql-bin.000001', master\_log\_pos=617, master\_connect\_retry=30;

**说明：** master\_host：主数据库的!P地址； 如何找ip地址？ 打开主服务器终端，用ifconfig查看ens33 的ip地址 master port：主数据库的运行端口； master user： 在主数据库创建的用于同步数据的用户账号； master\_password：在主数据库创建的用于同步数据的用户密码； master\_log\_file：指定从数据库要复制数据的日志文件，通过查看主数据的状态，获取File 参数； master\_log\_pgs：指定从数据库从哪个位置升始复制数据，通过查看主数据的状态，款取Position参数： master\_connect retry：连按失败重试的时间问隔，单位为秒。 #### 在从服务器查看主从同步状态 - 命令 - show slave status - 在show slave status后面加个是纵向排列参数。 - 查看slave io running和slave sql running的状态，应该是nono的状态

#### 在从服务器中接受主从同步

在mysql命令中，运行 start slave;

#### 再次在从服务器运行show slave status

检查slave io running 和slave sql running的状态是不是变成了yes。

# redis主从服务器

[[redis集群（3主3从-docker配置案例）|redis主从服务器]]

# Dockerfile

^5de74a 用来构建Docker镜像的本地文件，是由一条条构建镜像所需的指令和参数构成的脚本； - Dockerfile官网-https://docs.docker.com/engine/reference/builder/

编写完Dockerfile文件后用build命令可以根据Dockerfile文件进行配置构建镜像

## 与docker-compose的区别

[[Docker大笔记#与dockerfile文件的区别|与docker-compose的区别]]

## 基础

注意： - 每条保留的字指令都必须为大写字母且后面都要跟随至少一个参数 - 指令按照从上到下的顺序执行 - #表示注释 - 每条指令都会创建一个新的镜像层并对镜像进行提交

执行流程： - 从基础镜像运行一个容器 - 执行一条命令并对容器做出修改 - 执行类似Docker commit的操作提交一个新的镜像层 - Docker再基于刚提交的镜像运行一个新的容器 - 执行dockerfile中的下一条指令直到所有指令都执行完成； [[Pasted image 20230407202417.png]]

## 保留字指令

* FROM：基础镜像，基于哪个镜像构建
* MAINTAINER:镜像维护者的姓名和邮箱地址
* RUN： 构建镜像时执行的命令： shell格式和exec格式； shell：命令终端模式 exec：[“可执行文件”,“参数1”,“参数2”]
* EXPOSE:当前容器对外暴露出的端口（端口映射）
* WORKDIR:创建容器后，终端登录后的默认工作目录（bash后默认打开哪个工作文件夹）
* USER:指定镜像用哪个用户角色执行，不写的话就是默认root用户
* ENV：构建镜像中的环境变量 ENV MY RATH（变量值纯大写） /usr/mytest（目录地址） 这个环境变量可以在后续的任何RUN指令中使用，这就如同在命令前面指定了环境变量前级一样： 也可以在其它指令中直接使用这些环境变量， 比如：WORKDIR $MY\_PATH
* ADD:拷贝宿主机的文件进镜像，自动处理和解压tar包
* COPY: 拷贝文件和目录到镜像中，

[[Pasted image 20230407204802.png]] - VOLUME：指定构建时使用的容器卷 - CMD: [[Pasted image 20230407204946.png]] 注意：Dockerfile中是可以有多个CMD指令的，但是只有最后一个生效，**CMD会被Docker run之后的参数替换**，在Docker run命令后面再加参数就相当于在Dockerfile中的cmd字指令后又加入一行cmd，就覆盖了前面的cmd命令； - ENTRYPOINT: 也是指定容器运行时执行的命令 跟cmd类似，但是用这个写的不会被docker run后面的命令覆盖；命令行的参数会被当做参数送给entrypoint指令指定的程序；

[[Pasted image 20230407210133.png]] 案例如下：假设已通过 Dockerfile 构建了nginx.test 镜像： FROM nginx ENTRYPOINT [“nginx”, “-c”] # 定参；固定的命令 CMD [“/etc/nginx/nginx.conf”]＃ 变参,把这个作为一个内容传递到entrypoint的后面； 组成一个完整的命令： nginx -c /etc/nginx/nginx.conf

## 熟悉操作

# 虚悬镜像

查看虚悬镜像命令：

docker image ls -f dangling=true

删除虚悬镜像命令：

docker image prune

# docker微服务

（此部分跳过）

# ip地址

命令：ifconfig - 主机的地址看ens33； - 容器的ip地址看docker即可；

# docker network

安装完docker后会默认创建三个网络模式 列出网络列表：

docker network ls

查看更多network命令帮助：

docker network COMMAND

## 常用基本命令

* All命令：docker network help
* 查看网络：docker network ls
* 查看网络源数据：docker network inspect xxx网络名字
* 删除网络:docker network rm xxx网络名字
* 案例

## 能干什么？

* 实现容器间的互联和通信、以及端口映射；
* 容器ip发生变动时，可以通过服务名直接网络通信而不受影响。

## 网络模式

[[Pasted image 20230410125311.png]] - bridge：每个容器独立的ip地址；使用命令： –network bridge指定 - host：共享使用宿主机的ip和端口；使用命令: –network host指定 - none:不进行任何网络配置 - container：共享使用其他容器的ip和端口范围；使用命令: –network container:NAME或容器id

## docker容器的ip地址会随着其他容器的变化变更

已经自动分配好的容器的ip地址，如果其中一个停止了，此容器的ip地址会自动腾给其他新容器；无法再次通过这个ip地址访问这个容器。相当于动态的ip地址；

## docker的桥接

docker使用的是linux的桥接，会在虚拟机中虚拟出一个docker的容器网桥（docker0）； 该网桥（docker0）有地址段网段（ip地址的范围） 该虚拟出的网桥（docker0）是每个容器的默认网关； 同一宿主机下的容器接入的都是同一个网桥; ## bridge [[Pasted image 20230410132230.png]] ### 接口 网桥上的接口叫做veth，容器上的接口为eth0，每个容器的eth0接口和网桥上的veth互接。两两匹配，一一匹配； [[Pasted image 20230410133852.png]] #### 如何查看一对匹配？ 在宿主机上使用命令：

ip addr

可以查看到docker网桥（docker0）的接口信息； [[Pasted image 20230410134535.png|600]]

在docker容器内使用命令：

ip addr

可以查看docker容器上跟网桥docker0相接的接口信息； [[Pasted image 20230410134714.png|600]] 查看容器内部信息：

docker inspect 容器id

## host

直接用宿主机的ip地址加端口，相当于在机器上装服务 命令：

docker run -d -p 8083:8080 --network host --name tomcat83 billygoo/tomcat8-jdk8

* 错误使用：会报错，会提示使用主机网络模式时将丢弃已发布的端口 [[Pasted image 20230410155925.png]]
* 正确使用： 直接不写不指定端口，就不会提示

docker run -d --network host --name tomcat83 billygoo/tomcat8-jdk8

* 查看共用结果：ip addr [[Pasted image 20230410160526.png]] 查看容器内部信息：

docker inspect 容器id

## none

禁用网络功能，只有lo标识 - 如何启动？

docker run -d -p 8084:8080 --network none --name tomcat84 billygoo/tomcats-jdk8

查看容器内部信息：

docker inspect 容器id

## container

新建的容器和己经存在的一个容器共享一个网络ip配置而不是和宿主机共享。 新创建的容器不会创建自己的网卡，配置自己的1P，而是和一个指定的容器共享IP、端口范围等。同样，两个容器除了网络方面．其他的如文件系统、进程列表等还是隔离的。 [[Pasted image 20230410192951.png|600]]

* 命令使用：

docker run - it --network container: 要用哪个容器的端口和ip --name 本容器名称 镜像名称 /bin/sh

* 注意：寄生容器被停掉，则该容器将无法访问；

## 自定义网络

* 创建网络的命令：

docker network create 自定义的网络名称

* 让新创建的容器加入到上面创建好的自定义网络

docker run -d -p 8081: 8080 --network 自定义的网络名称 --name 自己要命名的容器名称 容器镜像实际名称

* 进入容器内部 ### 没有自定义网络的时候
* 问题： 两个网段按照服务名是ping不通的；

### 加入自定义网络后

假设：两个容器创建后默认都是网桥模式。 因为：两个容器的在同一网络中 所以：两个容器是能互相ping 服务名。

### 加入自定义网络后为什么能ping通服务名？

“自定义网络”由dns解析功能，能够解析出容器ip地址对应的容器名称，因此能够ping通。 ### 实际工作中 因为：容器的ip地址可能会因为停止、重启、删除而变更 所以：需要将服务名写死，ping服务名

### 为什么不把ip地址写死？

在Docker中，使用静态IP地址有一些缺点： - IP地址可能会冲突：如果您手动指定IP地址，则必须确保该地址未被其他容器或设备使用。否则，将导致IP地址冲突，可能会导致网络故障。

* 管理困难：如果您的应用程序需要在不同的环境中运行，例如开发、测试和生产环境，那么静态IP地址将非常难以管理和维护。每个环境都可能需要不同的IP地址。
* 动态分配更灵活：使用动态分配的IP地址，Docker可以更灵活地管理IP地址，并且可以确保IP地址的唯一性。这些地址是从Docker网络的IP地址池中动态分配的，这使得在不同的环境中部署和管理应用程序更加容易。

因此，虽然在某些情况下静态IP地址可能有用，但通常不推荐在Docker中使用静态IP地址。

### 什么时候用自定义网络？

建议：相同服务的多个容器使用同一个网络

# docker compose

## 是什么？能干什么？

* 一个工具软件，官方开源项目；
* 实现对docker容器集群的快速编排；
* 同时部署多个服务，不用为每个服务单独写dockerfile文件；
* 将docker-compose 文件里的东西作为一个大项目；
* 用一条指令完成所有依赖的构建；

## 用什么使用compose？

用docker-compose的yml文件进行编写。使用yaml文件写好多个容器之间的调用关系。

## 官网提供有文档

地址：docker官网的compose

## 安装

### 正常可访问安装

* 下载包

curl -L "https://github.com/docker/compose/releases/download/1.29.2(版本号)/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

* 赋予这个包可执行权限

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

* 检查是否安装成功

docker-compose --version

### 代理安装

修改包地址的github地址为其他可用的地址；

## compose文件

基本思路 [[Pasted image 20230411122025.png|600]]

### 微服务改造升级

[p81-p86](https://www.bilibili.com/video/BV1gr4y1U7CY?p=81&vd_source=e742beba73941ecc64dc838b29db628a) #### 不使用docker-compose编排 不使用docker-compose编排时，用docker运行服务 - 需要注意服务的先后顺序（以redis和MySQL和微服务为例） - 需要进行多个run命令 - 容器间如果启停或宕机，有可能导致ip地址对应的容器实例变化，映射出错； - 此条可以通过服务调用解决，不推荐写死ip

#### 使用docker-compose编排

* 编写docker-compose文件

样例参考： [[Pasted image 20230411125828.png]] [[Pasted image 20230411130206.png]]

修改原配置的ip地址为服务名称即可： [[Pasted image 20230411130356.png]] [[Pasted image 20230411130455.png]]

## 启动docker-compose

启动前检查配置文件是否有问题：

docker-compose config -q

没有问题就不会输出任何消息；

## 与dockerfile文件的区别

^14b553 - [[Docker大笔记#^5de74a|dockerfile]]是构建镜像的； ^2809ae - compose文件是已经有镜像，用来运行的；

# docker 的监控

## docker portainer

轻量级的可视化docker前端服务插件，但是集群更推荐k8s ### 安装

docker run -d -p 8000:8000 -p 9000:9000 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer\_data:/data  
portainer/portainer

* restart always：跟随docker服务的重启而重启，保证不掉线；

### 访问

用本机地址或服务器的公网地址加端口号访问页面

### 设置用户名和密码

需要设置管理员用户名和密码；本机存储；

## CIG

* CAdvisor 监控收集
  + 能得到host和容器两个层级的监控数据
  + 能显示历史变化数据
* +InfluxDB 存储数据（go语言 开源）
  + 支持时间序列、时间相关的函数（比如显示最大、最小、求和）
  + 能够对大量数据进行计算
  + 支持任意事件的数据
* +Granfana；展示图表
  + 图形化选项，
  + 支持导入influxDB的数据
  + 能混合多种风格
* 更专业重量级的；

### 使用CIG之前

使用CIG之前可查询所有容器cpu、内存、网络流量数据的命令：

docker stats

* 不足：
  + 数据资料只能实时显示；
  + 没有存储这些信息的功能；
  + 没有监控指标是否过线的预警功能；

### compose 一键部署 CIG

#### CIG yaml文件

#cig\_yaml文件

version: '3.1'  
  
volumes:  
 grafana\_data: {}  
   
services:  
 influxdb:  
 image: tutum/influxdb:0.9  
 restart: always  
 environment:  
 - PRE\_CREATE\_DB=cadvisor  
 ports:  
 - "8083:8083"  
 - "8086:8086"  
 volumes:  
 - ./data/influxdb:/data  
  
 cadvisor:  
 image: google/cadvisor  
 links:  
 - influxdb:influxsrv  
 command:  
 - storage\_driver=influxdb -storage\_driver\_db=cadvisor -d storage\_driver\_host=influxsrv:8086  
 restart: always  
 ports:  
 - "8080:8080"  
 volumes:  
 - /:/rootfs:ro  
 - /var/run:/var/run:rw  
 - /sys:/sys:ro  
 - /var/lib/docker/:/var/lib/docker:ro  
  
 grafana:  
 user: "104"  
 image: grafana/grafana  
 restart: always  
 links:  
 - influxdb:influxsrv  
 ports:  
 - "3000:3000"  
 volumes:  
 - grafana\_data:/var/lib/grafana  
 environment:  
 - HTTP\_USER=admin  
 - HTTP\_PASS=admin  
 - INFLUXDB\_HOST=influxsrv  
 - INFLUXDB\_PORT=8086  
 - INFLUXDB\_NAME=cadvisor  
 - INFLUXDB\_USER=root  
 - INFLUXDB\_PASS=root

检查配置文件：

docker-compose config -q

没有问题不会输出任何消息

##### 启动

基于docker-compose.yml文件所在的目录执行命令：

docker-compose up

### 各服务访问端口

#cig端口 - cAdvisor：8080(第一次较慢)，只有最近两分钟，实时的； - influxDB: 8083（web界面）或8086（后台界面）； - Grafana: 3000，可以再yaml文件中修改账号和密码；（现在还不能展现，需要添加数据源）

### Grafana中添加panel

* 添加数据源 [[Pasted image 20230411141657.png]]
* 添加influxDB的细节信息 [[Pasted image 20230411141807.png]]
* 成功提示 [[Pasted image 20230411141839.png|300]]
* 接下来 [[Pasted image 20230411141950.png]] [[Pasted image 20230411142012.png]] [[Pasted image 20230411142101.png]] [[Pasted image 20230411142120.png]] [[Pasted image 20230411142154.png]] [[Pasted image 20230411142250.png]]
* 部署完成；