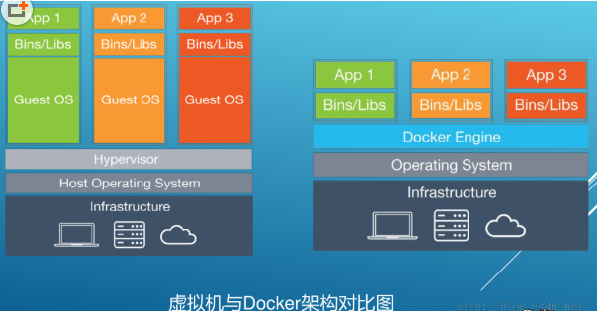
# 一、原理与安装

**1、目的：解决运行环境和配置问题**

**2、容器和虚拟机的区别：**

（1）传统虚拟机：虚拟硬件环境，带环境安装操作系统，资源占用多，冗余步骤多，启动慢

（2）容器：内核级虚拟化在操作系统层面虚拟化，直接复用本地主机操作系统，将应用容器化部署实现隔离，实现轻量化，降低资源占用，提高运行速度

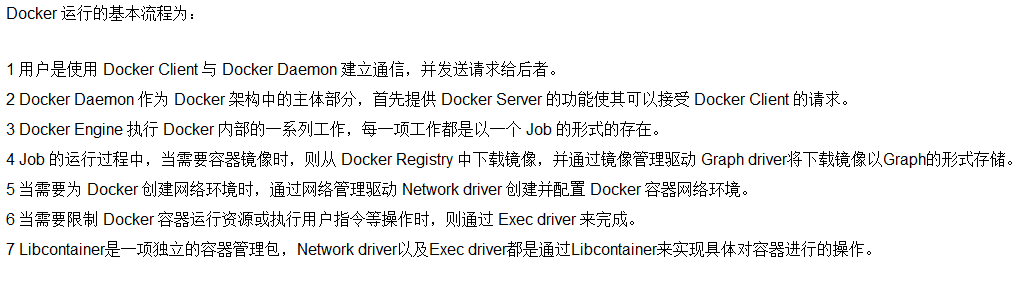


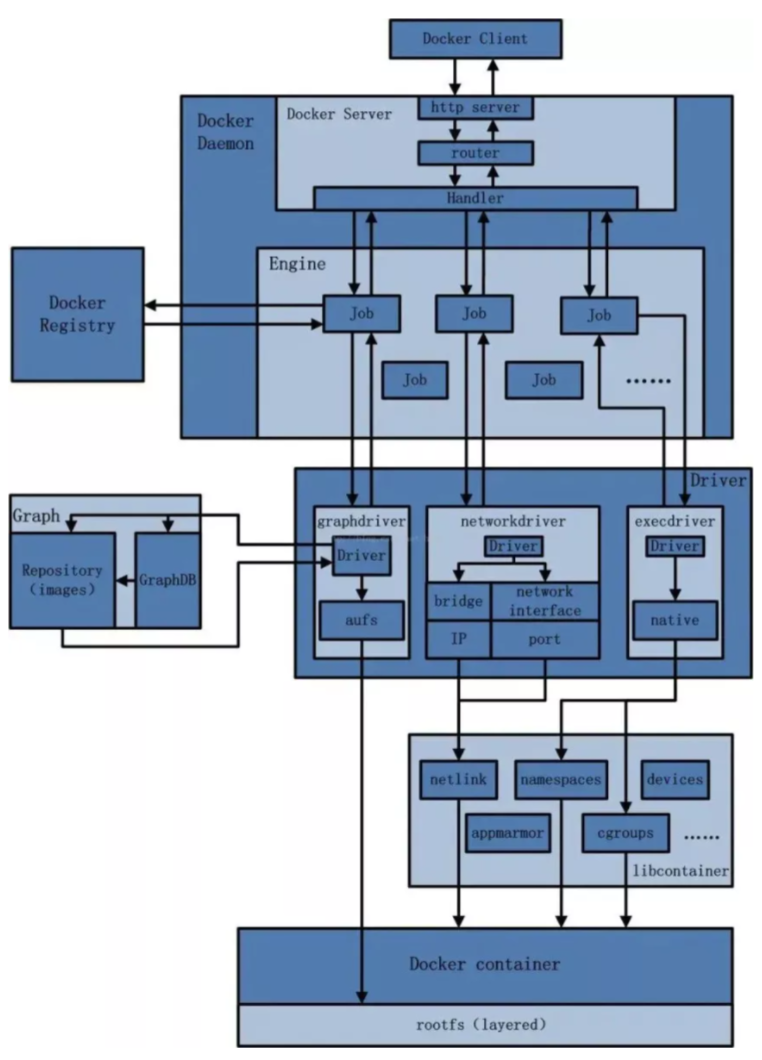


**3、docker必须运行在linux环境中，windows安装是增加了一层linux虚拟化**

**4、docker三要素：镜像、容器（实例）、仓库**

**5、架构：**Docker 是一个 C/S 模式的架构，后端是一个松耦合架构，众多模块各司其职。





**6、安装: https://docs.docker.com/engine/install/ubuntu/**

**7、问题：**

若执行sudo docker run hello-world时出现docker: Error response from daemon: Get "https://registry-1.docker.io/v2/": net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers).异常，则添加/etc/docker/daemon.json文件并写入：

{

"registry-mirrors": [

"https://pee6w651.mirror.aliyuncs.com",

"https://registry.docker-cn.com",

"http://hub-mirror.c.163.com",

"https://docker.mirrors.ustc.edu.cn",

"https://docker.m.daocloud.io",

"https://dockerpull.com"

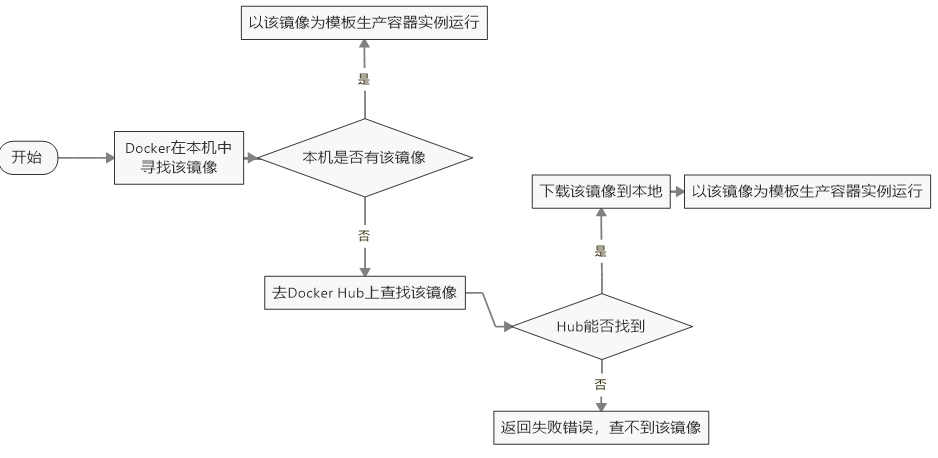
]

}

然后执行sudo systemctl daemon-reload

sudo systemctl restart docker

**8、docker run命令执行流程：**



# 二、常用命令

（1）管理：

启动docker： systemctl start docker

停止docker： systemctl stop docker

重启docker： systemctl restart docker

查看docker状态： systemctl status docker

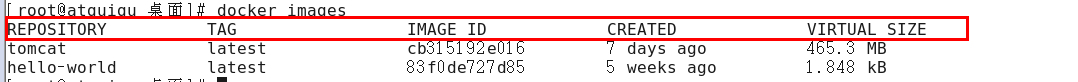
开机启动： systemctl enable docker

查看docker概要信息： docker info

监控执行状态：docker stats

（2）镜像操作：

* **docker images：列出本地主机上的镜像(images是复数)**



REPOSITORY：表示镜像的仓库源

TAG：镜像的标签版本号

IMAGE ID：镜像ID

CREATED：镜像创建时间

SIZE：镜像大小

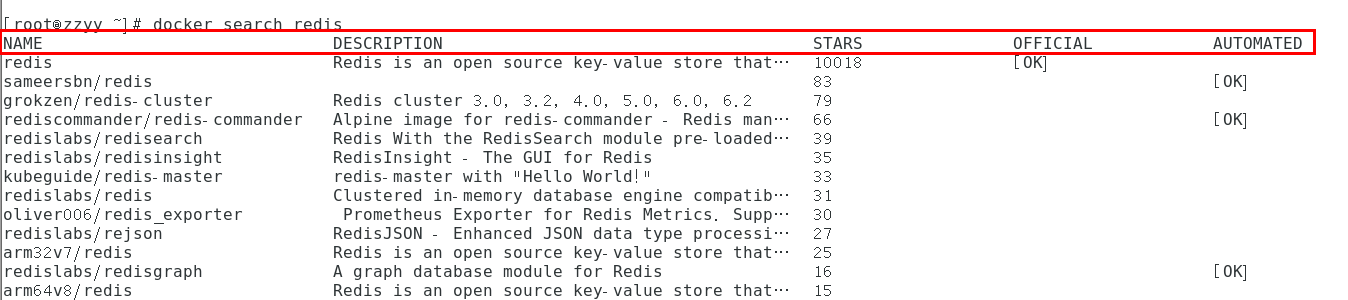
同一仓库源可以有多个 TAG版本，代表这个仓库源的不同个版本，我们使用 REPOSITORY:TAG 来定义不同的镜像。如果不指定一个镜像的版本标签，例如你只使用 ubuntu，docker 将默认使用 ubuntu:latest 镜像

* **docker search**  [OPTIONS] 镜像名字

OPTIONS说明：--limit : 只列出N个镜像，默认25个

docker search --limit 5 redis

**在远程仓库中搜索镜像。国内可能不能用**





* **Docker pull 镜像名字[:TAG]**

下载镜像，tag为版本号，没有tag默认最新版本，等价于docker pull 镜像:latest

* **Docker system df 查看镜像和容器所占用的空间**
* **Docker rmi -f 镜像id 删除单个镜像id**
* **Docker rmi -f 镜像名1:TAG 镜像名2:TAG 删除多个镜像**
* **docker rmi -f $(docker images -qa) 删除全部镜像**

（3）容器操作：

* docker run [OPTIONS] **IMAGE** [COMMAND] [ARG...]

OPTIONS说明（常用）：有些是一个减号，有些是两个减号

--name="容器新名字" 为容器指定一个名称；

-d: 后台模式运行容器并返回容器ID，也即启动守护式容器(后台运行)；

-i：以交互模式运行容器，通常与 -t 同时使用；

-t：为容器重新分配一个伪输入终端，通常与 -i 同时使用；也即启动交互式容器(前台有伪终端，等待交互)；

-P: 随机端口映射，大写P

-p: 指定端口映射，小写p



Docker run -it ubuntu /bin/bash

参数说明：

-i: 交互式操作。

-t: 终端。

centos : centos 镜像。

/bin/bash：放在镜像名后的是命令，这里我们希望有个交互式 Shell，因此用的是 /bin/bash。

要退出终端，直接输入 exit:

* **docker ps [OPTIONS]: 列出当前所有正在运行的容器**

OPTIONS说明（常用）：

-a :列出当前所有正在运行的容器+历史上运行过的

-l :显示最近创建的容器。

-n：显示最近n个创建的容器。

-q :静默模式，只显示容器编号。

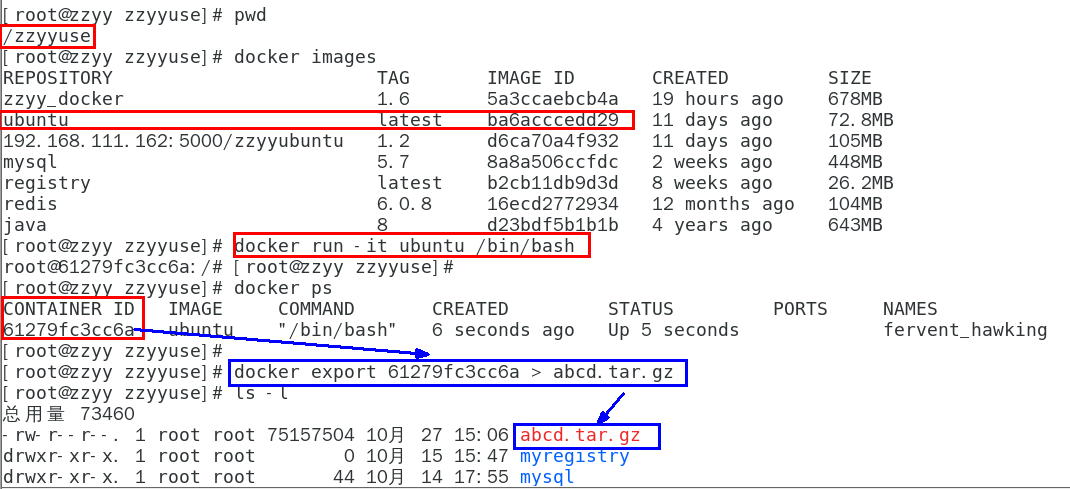
* **退出容器**: 交互模式进入容器直接输入exit彻底关闭，或者ctrl+p+q退出但不关闭容器
* **启动已关闭的容器：**docker start 容器ID或者容器名
* **重启容器：**docker restart 容器ID或者容器名
* **停止容器：**docker stop 容器ID或者容器名
* **强制停止容器：**docker kill 容器ID或者容器名
* **删除已停止的容器:** docker rm -f 容器ID或者容器名 ; docker rm -f $(docker ps -a -q)删除多个。 -f强制删除，没关闭也会被删除
* **查看容器日志**: docker logs 容器ID
* **查看容器内运行的进程:** docker top 容器ID
* **查看容器内部细节:** docker inspect 容器ID
* **进入正在运行的容器并以命令行交互:**

1. 命令1：docker exec -it 容器ID bashShell
2. 命令2：docker attach 容器ID（很少使用）
3. 区别：

attach直接进入启动时时的终端界面（启动容器时必须使用了-it交互模式并ctrl+p+q退出），如果在界面中输入exiit会导致容器直接关闭。

而exec模式则新建交互终端，即使容器使用后台启动-d模式也可以进入。且退出时使用exit也不会导致容器关闭，尽为关闭当前终端

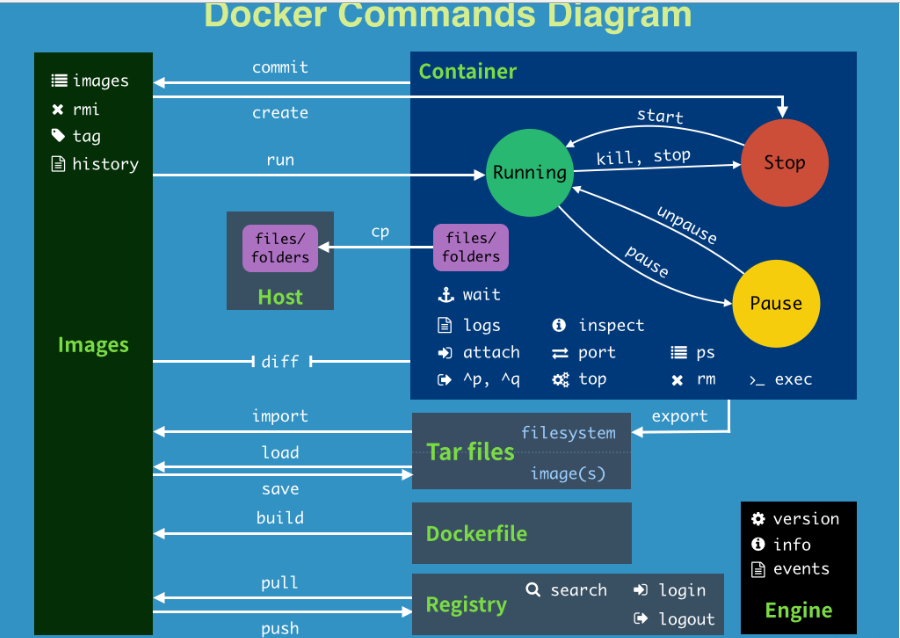
* **从容器内拷贝文件到主机上（容器→主机）：**docker cp 容器ID:容器内路径 目的主机路径
* **容器导出：docker export 容器ID > 文件名.tar （export 导出容器的内容留作为一个tar归档文件[对应import命令]）**



* **容器导入：cat 文件名.tar | docker import - 镜像用户/镜像名:镜像版本号 （import 从tar包中的内容创建一个新的文件系统再导入为镜像[对应export]）**



# 三、命令小结



attach Attach to a running container # 当前 shell 下 attach 连接指定运行镜像

build Build an image from a Dockerfile # 通过 Dockerfile 定制镜像

commit Create a new image from a container changes # 提交当前容器为新的镜像

cp Copy files/folders from the containers filesystem to the host path #从容器中拷贝指定文件或者目录到宿主机中

create Create a new container # 创建一个新的容器，同 run，但不启动容器

diff Inspect changes on a container's filesystem # 查看 docker 容器变化

events Get real time events from the server # 从 docker 服务获取容器实时事件

exec Run a command in an existing container # 在已存在的容器上运行命令

export Stream the contents of a container as a tar archive # 导出容器的内容流作为一个 tar 归档文件[对应 import ]

history Show the history of an image # 展示一个镜像形成历史

images List images # 列出系统当前镜像

import Create a new filesystem image from the contents of a tarball # 从tar包中的内容创建一个新的文件系统映像[对应export]

info Display system-wide information # 显示系统相关信息

inspect Return low-level information on a container # 查看容器详细信息

kill Kill a running container # kill 指定 docker 容器

load Load an image from a tar archive # 从一个 tar 包中加载一个镜像[对应 save]

login Register or Login to the docker registry server # 注册或者登陆一个 docker 源服务器

logout Log out from a Docker registry server # 从当前 Docker registry 退出

logs Fetch the logs of a container # 输出当前容器日志信息

port Lookup the public-facing port which is NAT-ed to PRIVATE\_PORT # 查看映射端口对应的容器内部源端口

pause Pause all processes within a container # 暂停容器

ps List containers # 列出容器列表

pull Pull an image or a repository from the docker registry server # 从docker镜像源服务器拉取指定镜像或者库镜像

push Push an image or a repository to the docker registry server # 推送指定镜像或者库镜像至docker源服务器

restart Restart a running container # 重启运行的容器

rm Remove one or more containers # 移除一个或者多个容器

rmi Remove one or more images # 移除一个或多个镜像[无容器使用该镜像才可删除，否则需删除相关容器才可继续或 -f 强制删除]

run Run a command in a new container # 创建一个新的容器并运行一个命令

save Save an image to a tar archive # 保存一个镜像为一个 tar 包[对应 load]

search Search for an image on the Docker Hub # 在 docker hub 中搜索镜像

start Start a stopped containers # 启动容器

stop Stop a running containers # 停止容器

tag Tag an image into a repository # 给源中镜像打标签

top Lookup the running processes of a container # 查看容器中运行的进程信息

unpause Unpause a paused container # 取消暂停容器

version Show the docker version information # 查看 docker 版本号

wait Block until a container stops, then print its exit code # 截取容器停止时的退出状态值

# 四、镜像原理

## 1、镜像是什么

是一种轻量级、可执行的独立软件包，它包含运行某个软件所需的所有内容，我们把应用程序和配置依赖打包好形成一个可交付的运行环境(包括代码、运行时需要的库、环境变量和配置文件等)，这个打包好的运行环境就是image镜像文件。

只有通过这个镜像文件才能生成Docker容器实例(类似Java中new出来一个对象)。

**镜像是分层的**，以我们的pull为例，在下载的过程中我们可以看到docker的镜像好像是在一层一层的在下载

## 2、镜像文件系统UnionFS（联合文件系统）

Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。**Union 文件系统是 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（没有父镜像，类似于java中的基础父类）**，可以制作各种具体的应用镜像。

**特性：**一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

## 3、docker镜像加载原理

docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS。

bootfs(boot file system)主要包含bootloader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，此时内存的使用权已由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。

**为什么docker容器占用空间那么小？**

对于一个精简的OS，rootfs可以很小，只需要包括最基本的命令、工具和程序库就可以了，**因为底层直接用Host的kernel，自己只需要提供 rootfs 就行了**。由此可见对于不同的linux发行版, bootfs基本是一致的, rootfs会有差别, 因此不同的发行版可以公用bootfs。

**为什么 Docker 镜像要采用这种分层结构呢**

镜像分层最大的一个好处就是共享资源，方便复制迁移，就是为了复用。

比如说有多个镜像都从相同的 base 镜像构建而来，那么 Docker Host 只需在磁盘上保存一份 base 镜像；

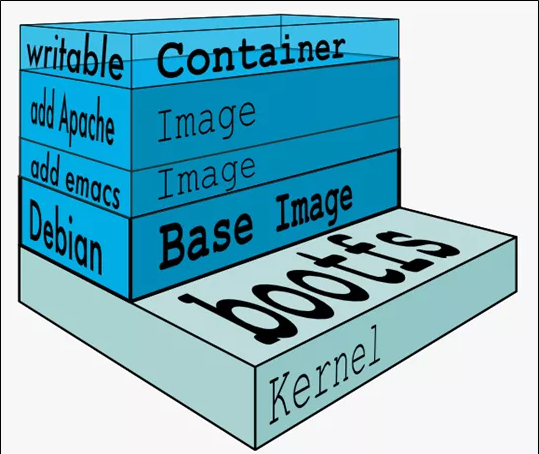
同时内存中也只需加载一份 base 镜像，就可以为所有容器服务了。而且镜像的每一层都可以被共享。

## 4、读写

**Docker镜像层都是只读的，容器层是可写的**

**当容器启动时，一个新的可写层被加载到镜像的顶部**。这一层通常被称作“容器层”，“容器层”之下的都叫“镜像层”。

**所有对容器的改动 - 无论添加、删除、还是修改文件都只会发生在容器层中**。只有容器层是可写的，容器层下面的所有镜像层都是只读的。



## 5、容器修改demo

**docker commit提交容器副本使之成为一个新的镜像。**

**使用方法：**

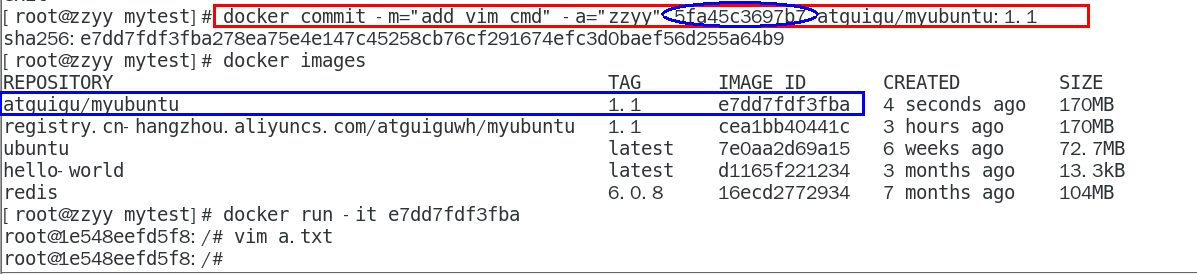
**docker commit -m="提交的描述信息" -a="作者" 修改后容器ID 要创建的目标镜像名:[标签名]**

（1）容器ubuntu安装vim并生成新镜像

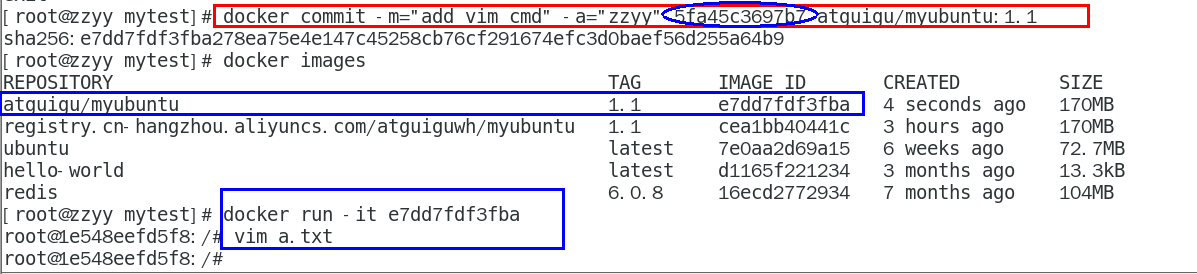
1.1. 从Hub上下载ubuntu镜像到本地并成功运行。原始的默认Ubuntu镜像是不带着vim命令的

1.2. 外网连通的情况下，安装vim

1.3. 安装完成后，commit我们自己的新镜像

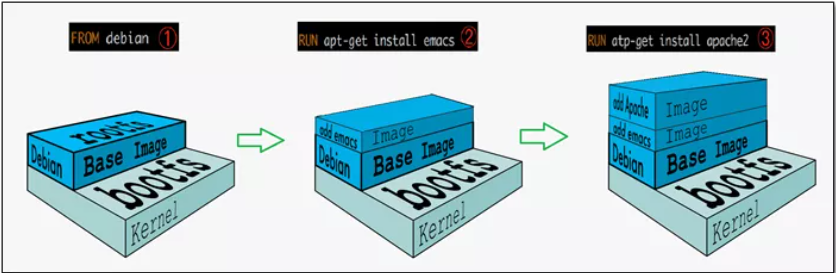


1.4. 启动我们的新镜像并和原来的对比



Docker中的镜像分层，支持通过扩展现有镜像，创建新的镜像。类似Java继承于一个Base基础类，自己再按需扩展。

新镜像是从 base 镜像一层一层叠加生成的。每安装一个软件，就在现有镜像的基础上增加一层



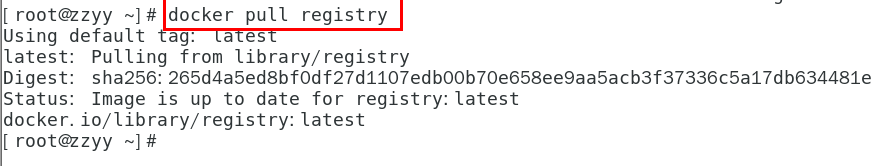
## 6、发布容器

容器修改好后可以发布到私有库或者如阿里云的公网库。

1. 私有库搭建

Docker Registry是官方提供的工具，可以用于构建私有镜像仓库

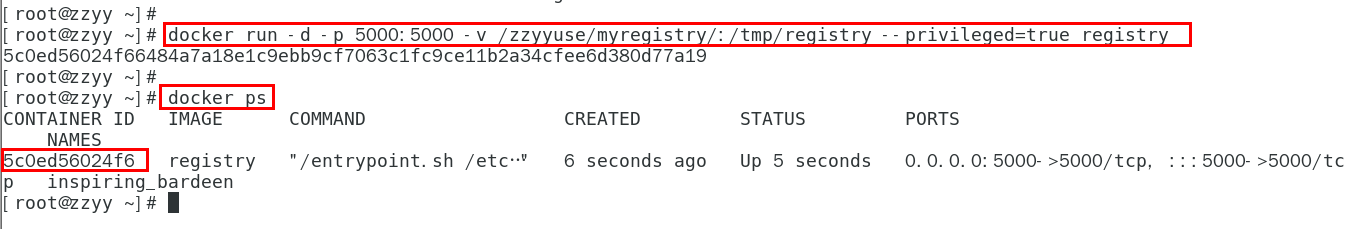
1. 下载镜像Docker Registry



1. 运行私有库Registry，相当于本地有个私有Docker hub。

docker run -d -p 5000:5000 -v /zzyyuse/myregistry/: /var/lib/registry --privileged=true registry

其中，-v是指仓库路径映射到宿主机。



1. 默认私有库不支持http，需要添加配置增加http支持。

修改/etc/docker/daemon.json配置，根节点增加"insecure-registries": ["192.168.111.162:5000"]。

{

"registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"],

"insecure-registries": ["192.168.111.162:5000"]

}

如果不生效需要重启docker

1. 创建生成新镜像
2. 查询私服库上有什么镜像

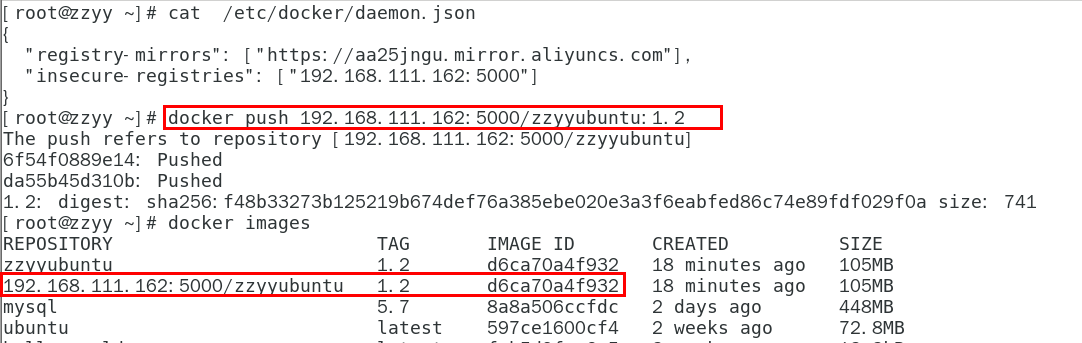
curl -XGET http://私有库ip:5000/v2/\_catalog

1. 修改修改符合私服规范的Tag

命令：docker tag 镜像:Tag Host:Port/Repository:Tag

docker tag zzyyubuntu:1.2 192.168.111.162:5000/zzyyubuntu:1.2

1. 推送到私服库：docker push 192.168.111.162:5000/zzyyubuntu:1.2



1. 使用curl验证私服库
2. Pull拉取执行：docker pull 192.168.111.162:5000/zzyyubuntu:1.2

# 五、Docker容器数据卷

## 1、卷是什么

**卷就是目录或文件**，存在于一个或多个容器中，由docker挂载到容器，但**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：

卷的设计目的就是数据的持久化，完全独立于容器的生存周期，因此**Docker不会在容器删除时删除其挂载的数据卷**

## 2、卷的作用

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. 数据持久化，数据卷的生命周期一直持续到没有容器使用它为止

## 3、数据卷操作

（1）添加数据卷映射：

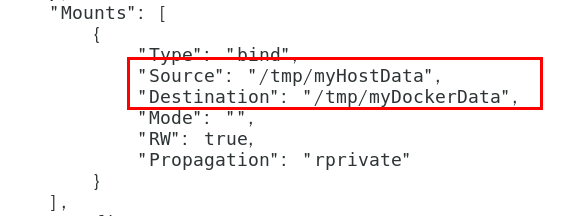
docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录 镜像名

可以挂多个-v

**注意：目录挂载的情况被centos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，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。**

**也即使用该参数，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。**

（2）查看是否挂载成功：docker inspect 容器ID



（3）数据同步：数据卷内数据的变化对容器和宿主机来说都是实时的

（4）挂载容器内只读的数据卷：

docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录:ro 镜像名

在-v的容器目录后面添加:ro，表示read only。在容器内该目录只读

（5）复用其他容器挂载的数据卷：

docker run -it --privileged=true --volumes-from 其他容器名 --name u2 ubuntu

## 六、简易安装mysql

1、拉取MySQL镜像

2、启动实例：

docker run -d -p 3306:3306 --privileged=true \

-v /zzyyuse/mysql/log:/var/log/mysql \

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

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

-e MYSQL\_ROOT\_PASSWORD=123456 --name mysql mysql:5.7

通过挂载MySQL配置目录将mysql的配置路径挂载到宿主机

3、在宿主机目录/zzyyuse/mysql/conf:下新建my.conf并写入以下内容：

[client]

default\_character\_set=utf8

[mysqld]

collation\_server = utf8\_general\_ci

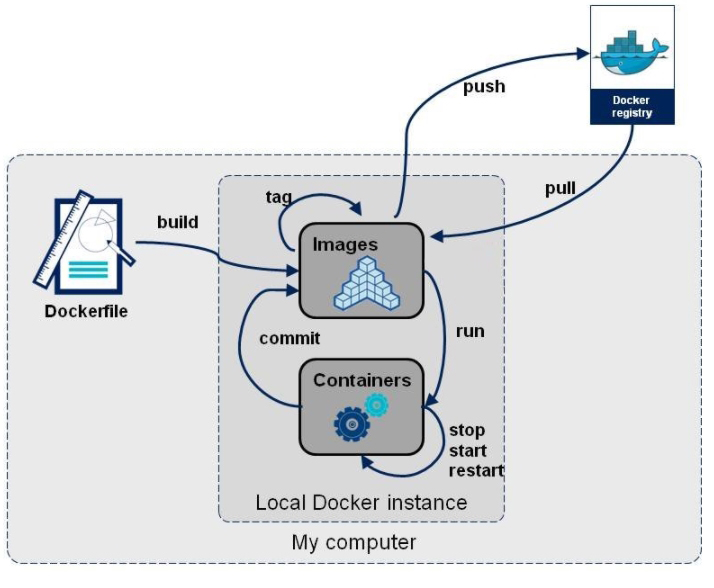
character\_set\_server = utf8

4、重启mysql实例

# 六、dockerFile

## 1、介绍

Dockerfile是用来构建Docker镜像的文本文件，是由一条条构建镜像所需的指令和参数构成的脚本。



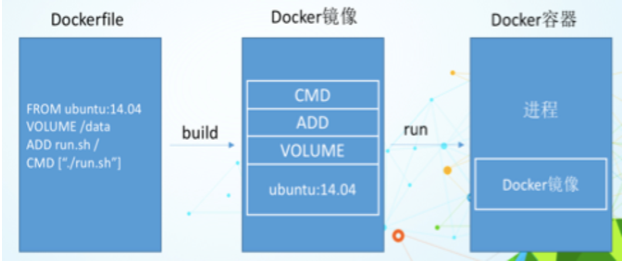
从应用软件的角度来看，Dockerfile、Docker镜像与Docker容器分别代表软件的三个不同阶段，

\* Dockerfile是软件的原材料

\* Docker镜像是软件的交付品

\* Docker容器则可以认为是软件镜像的运行态，也即依照镜像运行的容器实例

Dockerfile面向开发，Docker镜像成为交付标准，Docker容器则涉及部署与运维，三者缺一不可，合力充当Docker体系的基石。



1. Dockerfile，需要定义一个Dockerfile，Dockerfile定义了进程需要的一切东西。Dockerfile涉及的内容包括执行代码或者是文件、环境变量、依赖包、运行时环境、动态链接库、操作系统的发行版、服务进程和内核进程(当应用进程需要和系统服务和内核进程打交道，这时需要考虑如何设计namespace的权限控制)等等;

1. 2 Docker镜像，在用Dockerfile定义一个文件之后，docker build时会产生一个Docker镜像，当运行 Docker镜像时会真正开始提供服务;

1. Docker容器，容器是直接提供服务的。

## 2、DockerFile构建流程

（1）编写Dockerfile文件

（2）docker build命令构建镜像

（3）docker run依镜像运行容器实例

## 3、dockerFile编写要求

（1）每条保留字（docker命令词）指令**都必须为大写字母**且**后面要跟随至少一个参数**

（2）指令按照从上到下，顺序执行

（3）#号表示注释

（4）每条指令都会创建一个新的镜像层并对镜像进行提交

## 4、dockerFile构建镜像流程

（1）docker从基础镜像运行一个容器

（2）执行一条指令并对容器作出修改

（3）执行类似docker commit的操作提交一个新的镜像层

（4）docker再基于刚提交的镜像运行一个新容器

（5）执行dockerfile中的下一条指令直到所有指令都执行完成

## 5、dockerFile常用构建命令

### **（1）FROM**

基础镜像，当前新镜像是基于哪个镜像的，指定一个已经存在的镜像作为模板，第一条必须是FROM。例如mysql的dockerFIle首句：

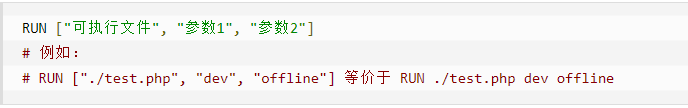


### （2）MAINTAINER

镜像维护者的姓名和邮箱地址

### （3）RUN

容器构建时需要运行的命令

有两种格式：shell格式。例如RUN yum -y install vim；另一种为exec格式，例如

RUN是在 docker build时运行

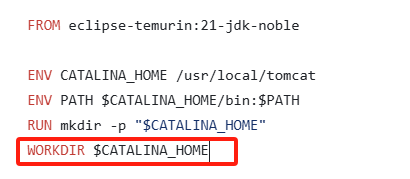
### （4）EXPOSE

当前容器对外暴露出的端口。



### （5）WORKDIR

指定在创建容器后，终端默认登陆的进来工作目录，一个落脚点。即-it模式启动后的目录



### （6）USER

指定该镜像以什么样的用户去执行，如果都不指定，默认是root

### （7）ENV

用来在构建镜像过程中设置环境变量。例如ENV MY\_PATH /usr/mytest

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缀一样；

也可以在其它指令中直接使用这些环境变量，比如：WORKDIR $MY\_PATH。

如WORKDIR实例图。

### （8）ADD

将宿主机目录下的文件拷贝进镜像且会自动处理URL和解压tar压缩包

### （9）COPY

类似ADD，拷贝文件和目录到镜像中。

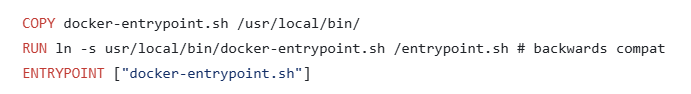
将从构建上下文目录中 <源路径> 的文件/目录复制到新的一层的镜像内的 <目标路径> 位置

COPY src dest

COPY ["src", "dest"]

<src源路径>：源文件或者源目录

<dest目标路径>：容器内的指定路径，该路径不用事先建好，路径不存在的话，会自动创建。



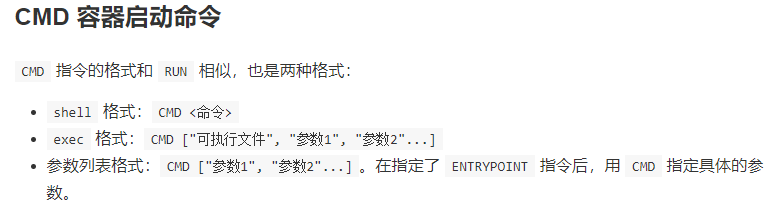
### （10）VOLUME

容器数据卷，用于数据保存和持久化工作



### （11）CMD

指定容器启动后的要干的事情。



注意：Dockerfile 中可以有多个 CMD 指令，但**只有最后一个生效，CMD 会被 docker run 之后的参数替换。**即cmd是默认后台启动-d模式下自动执行，但如果使用-it模式执行且在最后输入了/bin/bash进入容器，则会被/bin/bash命令替换掉cmd中设定的执行命令。

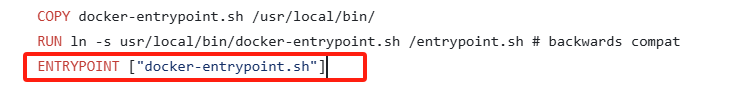


### （12）ENTRYPOINT

也是用来指定一个容器启动时要运行的命令。同样的，如果 Dockerfile 中如果存在多个 ENTRYPOINT 指令，仅最后一个生效。

类似于 CMD 指令，但是**ENTRYPOINT不会被docker run后面的命令覆盖**，

而且这些命令行参数会被当作参数送给 ENTRYPOINT 指令指定的程序



注意：ENTRYPOINT可以和CMD一起用，一般是**变参才会使用 CMD** ，这里的 CMD 等于是在给 ENTRYPOINT 传参。

当指定了ENTRYPOINT后，CMD的含义就发生了变化，不再是直接运行其命令而是将CMD的内容作为参数传递给ENTRYPOINT指令，他两个组合会变成graphic



|  |  |  |
| --- | --- | --- |
| 是否传参 | 按照dockerfile编写执行 | 传参运行 |
| Docker命令 | docker run nginx:test | docker run nginx:test -c /etc/nginx/new.conf |
| 衍生出的实际命令 | nginx -c /etc/nginx/nginx.conf | nginx -c /etc/nginx/new.conf |

也就是，CMD负责默认执行的参数，不指定就按照CMD中的默认来执行，ENTRYPOINT则设定为固定的命令行，CMD中的命令或者参数会传递给ENTRYPOINT的命令行之后组合一起执行。指定CMD参数时，变更CMD默认但不变更ENTRYPOINT的命令。

## 6、简单demo

### （1）编写DockerFile文件

|  |
| --- |
| FROM centos  MAINTAINER zzyy<zzyybs@126.com>    ENV MYPATH /usr/local  WORKDIR $MYPATH    #安装vim编辑器  RUN yum -y install vim  #安装ifconfig命令查看网络IP  RUN yum -y install net-tools  #安装java8及lib库  RUN yum -y install glibc.i686  RUN mkdir /usr/local/java  #ADD 是相对路径jar,把jdk-8u171-linux-x64.tar.gz添加到容器中,安装包必须要和Dockerfile文件在同一位置  ADD jdk-8u171-linux-x64.tar.gz /usr/local/java/  #配置java环境变量  ENV JAVA\_HOME /usr/local/java/jdk1.8.0\_171  ENV JRE\_HOME $JAVA\_HOME/jre  ENV CLASSPATH $JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar:$JRE\_HOME/lib:$CLASSPATH  ENV PATH $JAVA\_HOME/bin:$PATH    EXPOSE 80    CMD echo $MYPATH  CMD echo "success--------------ok"  CMD /bin/bash |

### （2）在DockerFile同目录下执行docker build -t 新镜像名字:TAG ./

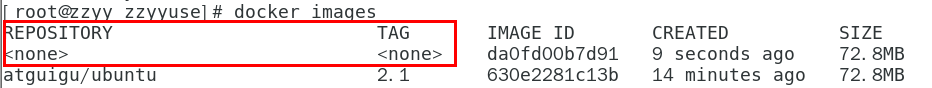
docker build -f /path/to/Dockerfile -t myimage:latest ./

(末尾的./路径是构建上下文，不能省略)

### （3）启动新镜像

## 7、虚悬镜像

指仓库名、标签都是<none>的镜像，俗称dangling image



在创建修改或者删除镜像时出错，没指定仓库名或者标签时可能出现的镜像，属于错乱镜像。

* 查看：docker image ls -f dangling=true
* 删除：docker image prune 删除全部虚悬镜像

## 8、spring微服务打包dockerFIle形成容器demo

### 1、编写jar包和dockerFile文件

|  |
| --- |
| # 基础镜像使用java  FROM java:8  # 作者  MAINTAINER zzyy  # VOLUME 指定临时文件目录为/tmp，在主机/var/lib/docker目录下创建了一个临时文件并链接到容器的/tmp  VOLUME /tmp  # 将jar包添加到容器中并更名为zzyy\_docker.jar  ADD docker\_SNAPSHOTboot-0.0.1-.jar zzyy\_docker.jar  # 运行jar包  RUN bash -c 'touch /zzyy\_docker.jar'  ENTRYPOINT ["java","-jar","/zzyy\_docker.jar"]  #暴露6001端口作为微服务  EXPOSE 6001 |

### 2、将jar包和dockerFile文件传到同一个目录执行docker build

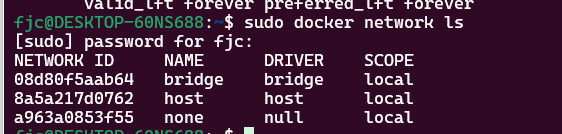
# 七、docker网络

## 1、作用

Docker容器内部的ip是会变动的，重启的先后顺序、删除、新建等都可能导致容器ip发生变更。

（1）容器间的互联和通信以及端口映射

（2）容器IP变动时候可以通过服务名直接网络通信而不受到影响



## 2、常用基本命令

### （1）docker network ls

查看网络

### （2）docker network inspect 网络名或id

查看网络源数据

### （3）docker network rm 网络名或id

删除网络

## 3、网络模式



bridge模式：使用--network bridge指定，默认使用docker0

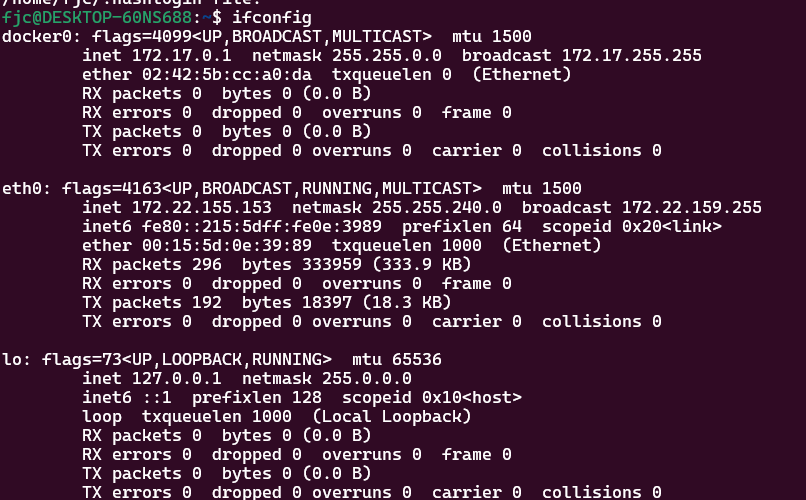
host模式：使用--network host指定

none模式：使用--network none指定。很少使用。

container模式：使用--network container:NAME或者容器ID指定

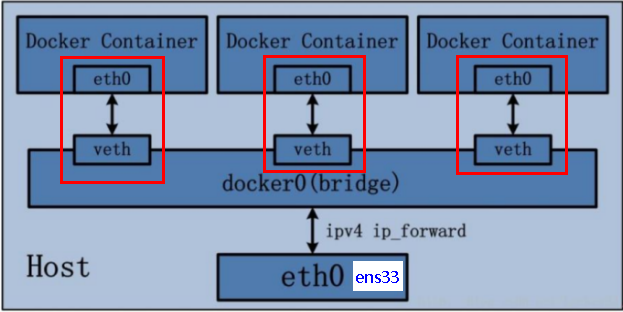
## 4、bridge模式

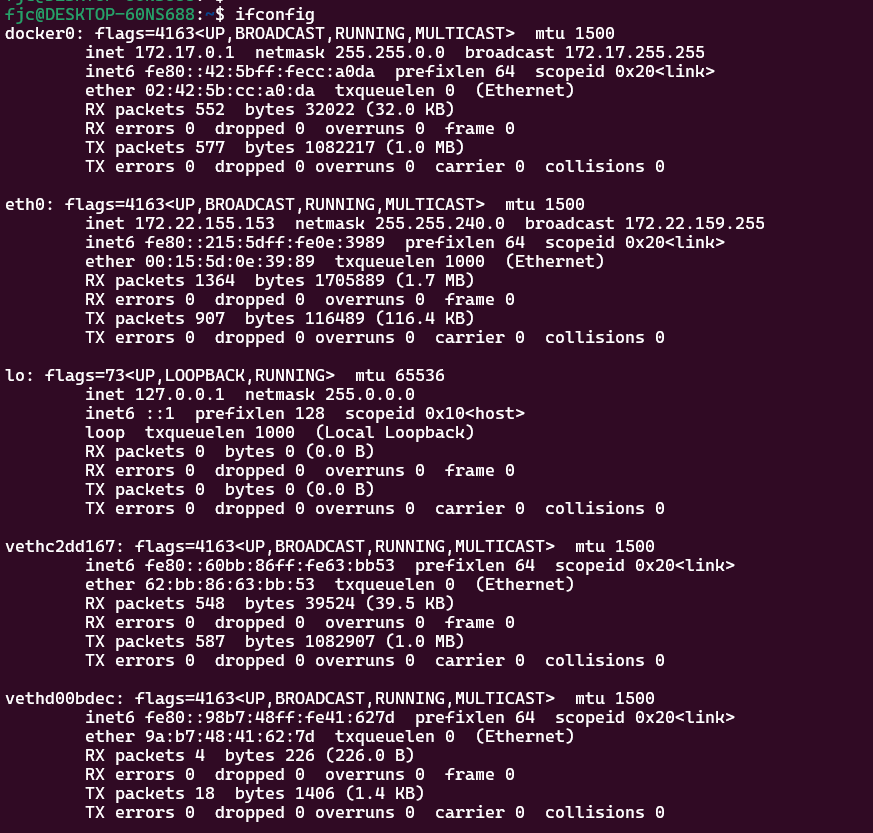
Docker 服务启动时默认会创建一个 docker0 网桥（使用ifconfig命令可以查看到，其上有一个 docker0 内部接口），该桥接网络的名称为docker0，它在内核层连通了其他的物理或虚拟网卡，这就将所有容器和本地主机都放到同一个物理网络。Docker 默认指定了 docker0 接口 的 IP 地址和子网掩码，让主机和容器之间可以通过网桥相互通信。



（1）Docker使用Linux桥接，在宿主机虚拟一个Docker容器网桥(docker0)，Docker启动一个容器时会根据Docker网桥的网段分配给容器一个IP地址，称为Container-IP，同时**Docker网桥是每个容器的默认网关**。因为在同一宿主机内的容器都接入同一个网桥，这样容器之间就能够通过容器的Container-IP直接通信。

（2）docker run 的时候，没有指定network的话默认使用的网桥模式就是bridge，使用的就是docker0。在宿主机ifconfig,就可以看到docker0和自己create的network(后面讲)eth0，eth1，eth2……代表网卡一，网卡二，网卡三……，lo代表127.0.0.1，即localhost，inet addr用来表示网卡的IP地址。





（3）网桥docker0创建一对对等虚拟设备接口一个叫veth，另一个叫eth0，成对匹配。

* 整个docker服务的网桥都是docker0，类似一个交换机有一堆接口，每个接口叫veth，在本地主机和容器内分别创建一个虚拟接口，并让他们彼此联通（这样一对接口叫veth pair）；
* 每个容器实例内部也有一块网卡，每个接口叫eth0；
* docker0上面的每个veth匹配某个容器实例内部的eth0，两两配对，一一匹配。

通过上述，将所有容器都通过docker0连接到这个内部网络上，两个容器在同一个网络下,会从这个网关下各自拿到分配的ip，此时**两个容器的网络是互通的**。

## 5、host模式

（1）说明

host模式直接使用宿主机的 IP 地址与外界进行通信，不再需要额外进行NAT 转换。容器将不会获得一个独立的Network Namespace， 而是和宿主机共用一个Network Namespace。容器将不会虚拟出自己的网卡而是使用宿主机的IP和端口。

（2）指定host模式

在容器启动时添加--network host。

注意：使用host模式时每个容器都直接共用宿主机ip，因此不需要在指定-p端口映射

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



**访问服务时直接使用http://宿主机IP:端口/**

## 6、none模式（极少使用）

禁用网络功能，容器内只有lo标识(就是127.0.0.1表示本地回环)

docker run -d -p 8084:8080 --network none --name tomcat84 billygoo/tomcat8-jdk8

## 7、container模式

新建的容器和已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IP，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。

docker run -d -p 8086:8080 **--network container:tomcat85** --name tomcat86 billygoo/tomcat8-jdk8

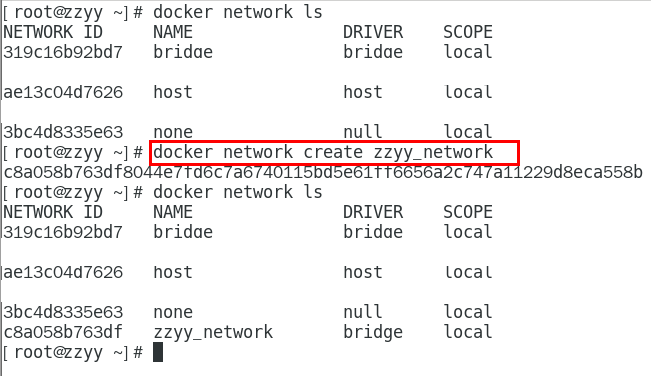
**--network container：其他容器id或名字**

表示复用另外一个容器的网络，两个容器的端口不能出现冲突

## 8、自定义网络模式

自定义桥接网络,自定义网络默认使用的是桥接网络bridge

（1）创建新网络



（2）新建容器加入上一步新建的自定义网络

docker run -d -p 8081:8080 --network zzyy\_network --name tomcat81 billygoo/tomcat8-jdk8

docker run -d -p 8082:8080 --network zzyy\_network --name tomcat82 billygoo/tomcat8-jdk8

（3）好处

使用自定义网络，容器将容器名和容器id进行了绑定，相同网络内可以直接根据容器名连接。而使用默认bridge网络无法使用容器名绑定

# 八、Docker-compose容器编排

Docker-Compose是Docker官方的开源项目，负责实现对Docker容器集群的快速编排。

Compose 是 Docker 公司推出的一个工具软件，可以管理多个 Docker 容器组成一个应用。你需要定义一个 YAML 格式的配置文件docker-compose.yml，**写好多个容器之间的调用关系**。然后，只要一个命令，就能同时启动/关闭这些容器。

## （1）作用

docker建议我们每一个容器中只运行一个服务,因为docker容器本身占用资源极少,所以最好是将每个服务单独的分割开来。

如果我需要同时部署好多个服务，docker官方给我们提供了docker-compose多服务部署的工具。**Compose允许用户通过一个单独的docker-compose.yml模板文件（YAML 格式）来定义一组相关联的应用容器为一个项目（project）**。可以很容易地用一个配置文件定义一个多容器的应用，然后使用一条指令安装这个应用的所有依赖，完成构建。Docker-Compose 解决了容器与容器之间如何管理编排的问题。

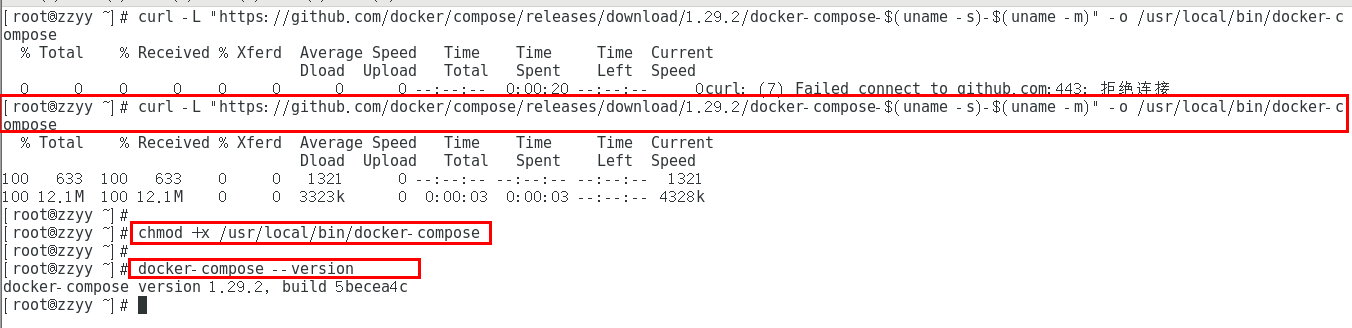
## （2）下载安装

官网：https://docs.docker.com/compose/compose-file/compose-file-v3/

下载：https://docs.docker.com/compose/install/

安装步骤：

|  |
| --- |
| curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose |
| chmod +x /usr/local/bin/docker-compose |
| docker-compose --version |



## （3）卸载



## （4）compose构成

（1）一文件：docker-compose.yml

（2）两要素：

* 服务（service）：一个个应用容器实例，比如订单微服务、库存微服务、mysql容器、nginx容器或者redis容器
* 工程（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。

## （5）编排流程

* 编写Dockerfile定义各个微服务应用并构建出对应的镜像文件
* 使用 docker-compose.yml 定义一个完整业务单元，安排好整体应用中的各个容器服务。
* 最后，执行docker-compose up命令 来启动并运行整个应用程序，完成一键部署上线

## （7）Compose常用命令

docker-compose -h # 查看帮助

docker-compose up # 启动所有docker-compose服务

docker-compose up -d # 启动所有docker-compose服务并后台运行

docker-compose down # 停止并删除容器、网络、卷、镜像。

docker-compose exec yml里面的服务id # 进入容器实例内部 docker-compose exec docker-compose.yml文件中写的服务id /bin/bash

docker-compose ps # 展示当前docker-compose编排过的运行的所有容器

docker-compose top # 展示当前docker-compose编排过的容器进程

docker-compose logs yml里面的服务id # 查看容器输出日志

docker-compose config # 检查配置

docker-compose config -q # 检查配置，有问题才有输出

docker-compose restart # 重启服务

docker-compose start # 启动服务

docker-compose stop # 停止服务

## （8）compose编排demo

1、编写业务程序，编写dockerFile，构建和拉取相应的docker镜像

2、编写docker-compose.yml文件：

|  |
| --- |
| version: "3"    services:  microService:  image: zzyy\_docker:1.6  container\_name: ms01  ports:  - "6001:6001"  volumes:  - /app/microService:/data  networks:  - atguigu\_net  depends\_on:  - redis  - mysql    redis:  image: redis:6.0.8  ports:  - "6379:6379"  volumes:  - /app/redis/redis.conf:/etc/redis/redis.conf  - /app/redis/data:/data  networks:  - atguigu\_net  command: redis-server /etc/redis/redis.conf    mysql:  image: mysql:5.7  environment:  MYSQL\_ROOT\_PASSWORD: '123456'  MYSQL\_ALLOW\_EMPTY\_PASSWORD: 'no'  MYSQL\_DATABASE: 'db2021'  MYSQL\_USER: 'zzyy'  MYSQL\_PASSWORD: 'zzyy123'  ports:  - "3306:3306"  volumes:  - /app/mysql/db:/var/lib/mysql  - /app/mysql/conf/my.cnf:/etc/my.cnf  - /app/mysql/init:/docker-entrypoint-initdb.d  networks:  - atguigu\_net  command: --default-authentication-plugin=mysql\_native\_password #解决外部无法访问    networks:  atguigu\_net: |

3、执行 docker-compose up 或者 执行 docker-compose up -d

4、测试

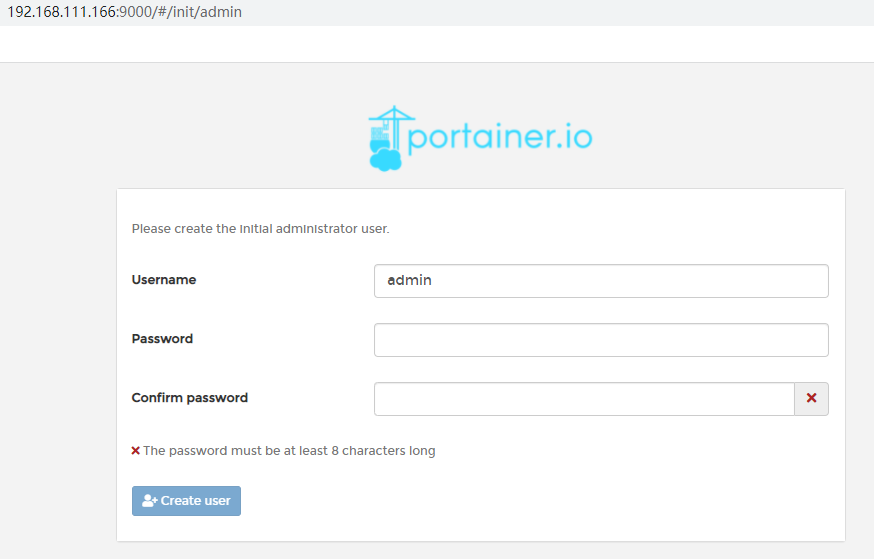
# 九、轻量级可视化工具

Portainer 是一款轻量级的应用，它提供了图形化界面，用于方便地管理Docker环境，包括单机环境和集群环境。官网https://www.portainer.io/

（1）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、

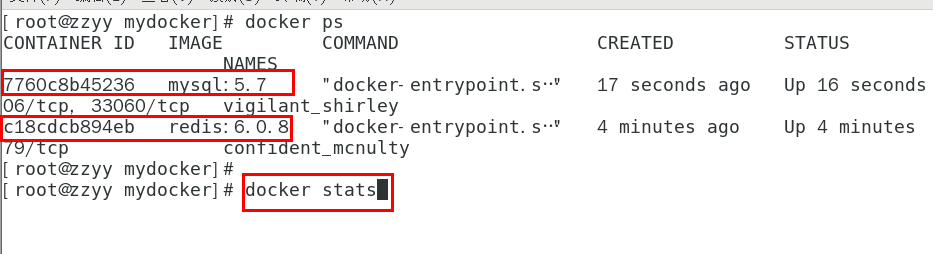
第一次登录需创建admin，访问地址：xxx.xxx.xxx.xxx:9000

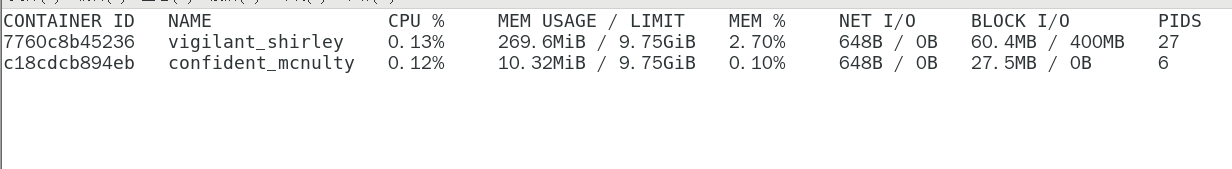


# 十、Docker容器监控

## 1、命令行监控

使用docker stats监控

**docker stats命令的结果**

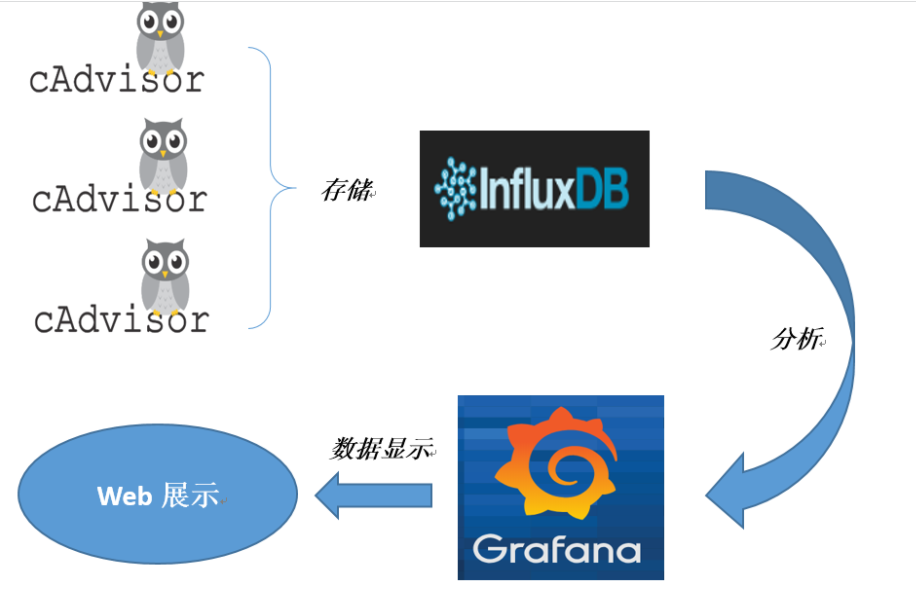


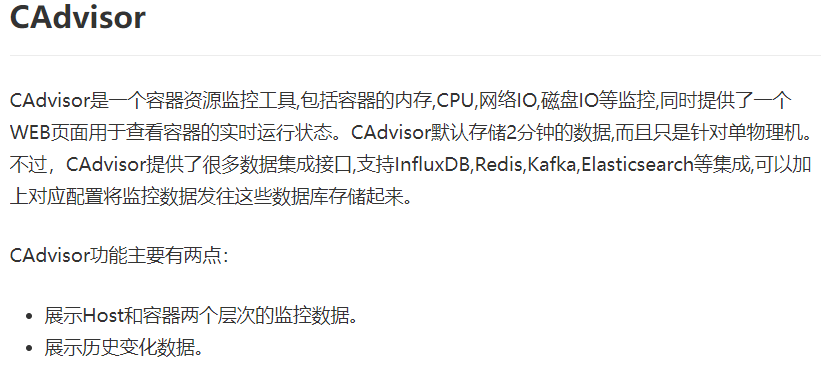
问题：通过docker stats命令可以很方便的看到当前宿主机上所有容器的CPU,内存以及网络流量等数据，一般小公司够用了。

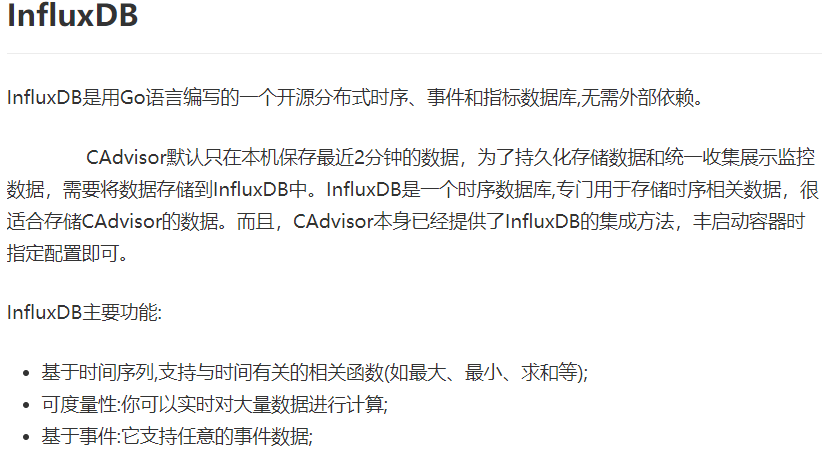
但是，docker stats统计结果只能是当前宿主机的全部容器，数据资料是实时的，没有地方存储、没有健康指标过线预警等功能

## 2、监控工具

CAdvisor监控收集+InfluxDB存储数据+Granfana展示图表









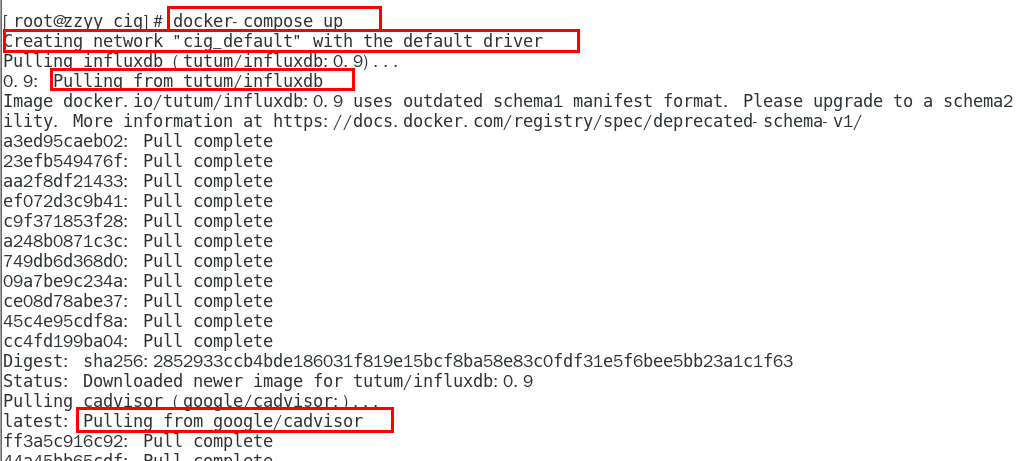
## 3、部署

### （1）新建目录

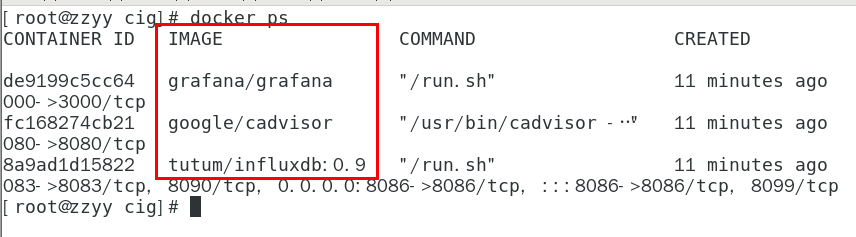
### （2）编写docker-compose.yml

|  |
| --- |
| 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 -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  user: "104"  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 |

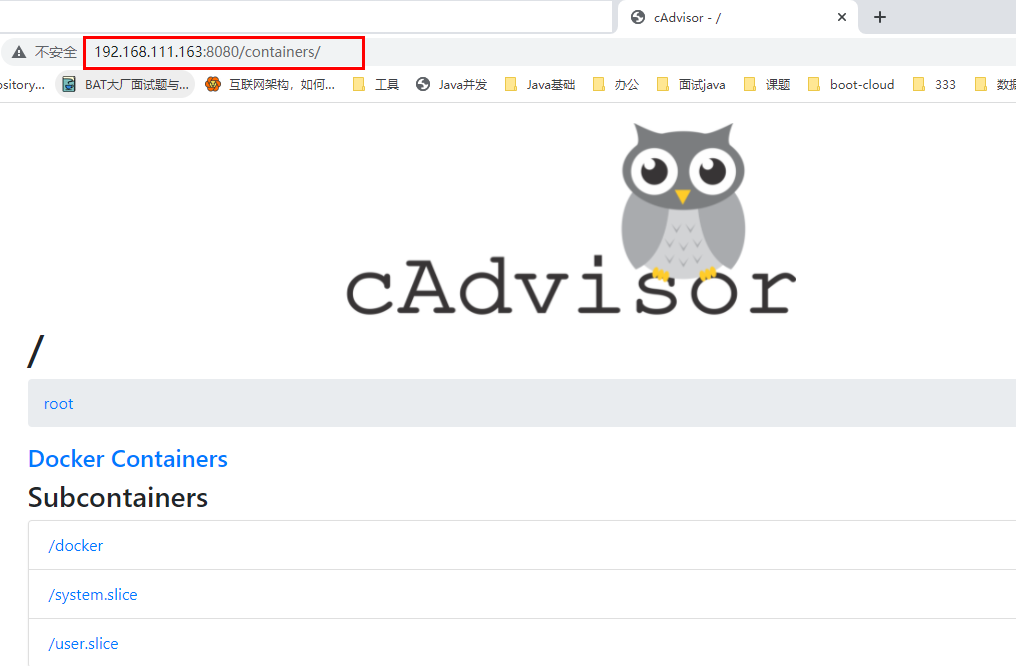
### （3）启动docker-compose文件



### （4）检查是否启动



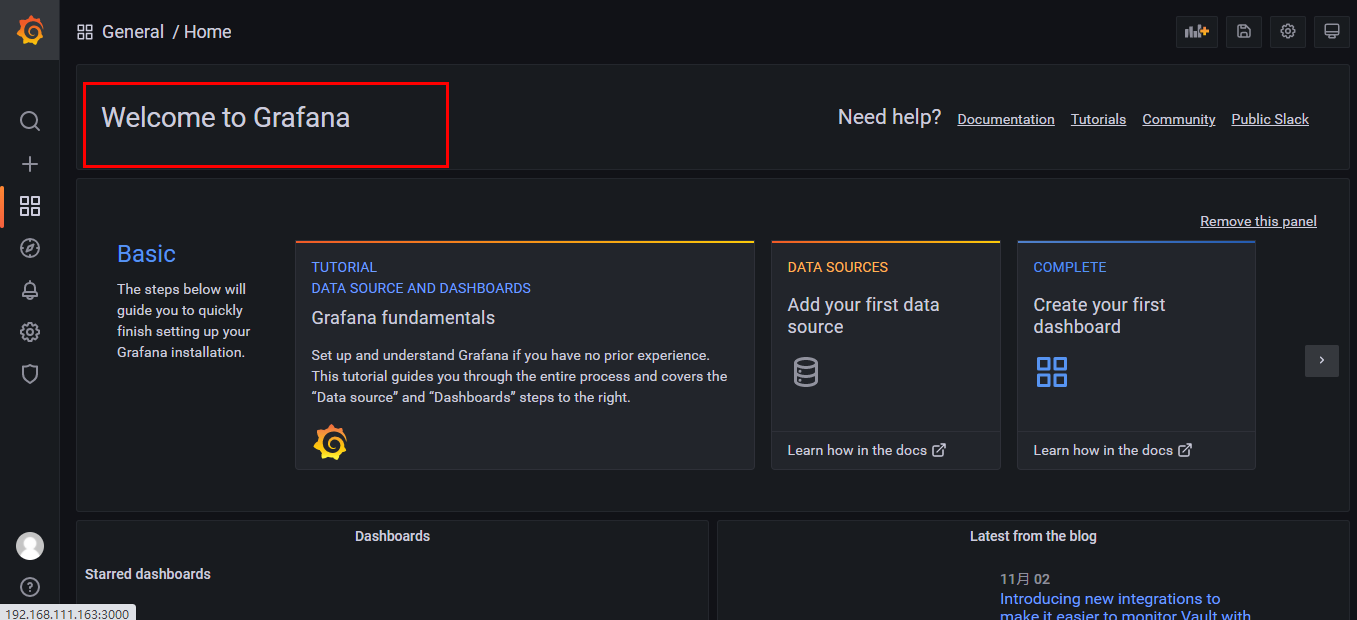
### （5）浏览cAdvisor收集服务，<http://ip:8080/>



### （6）浏览influxdb存储服务，http://ip:8083/

### （7）浏览grafana展现服务，http://ip:3000

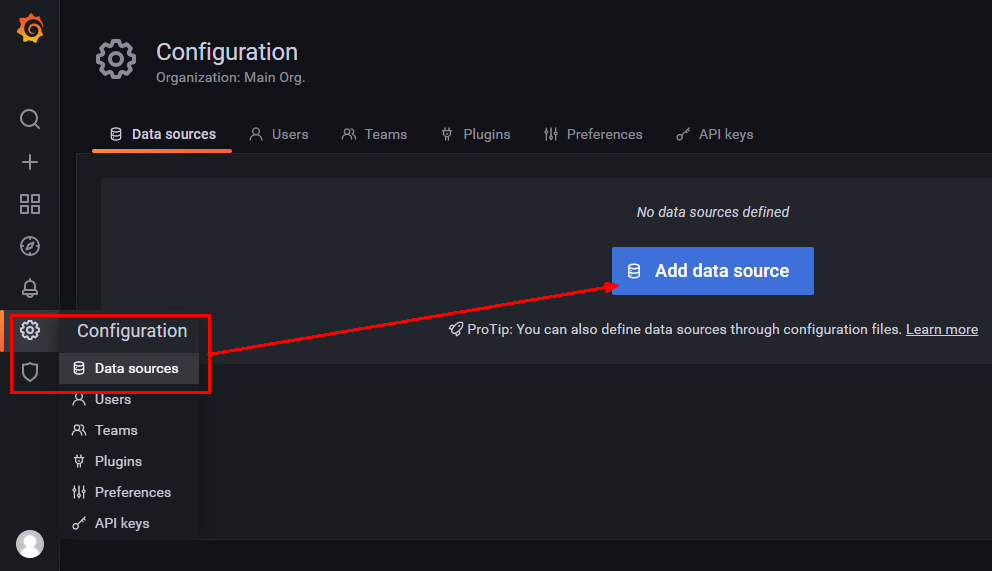
ip+3000端口的方式访问,默认帐户密码（admin/admin）



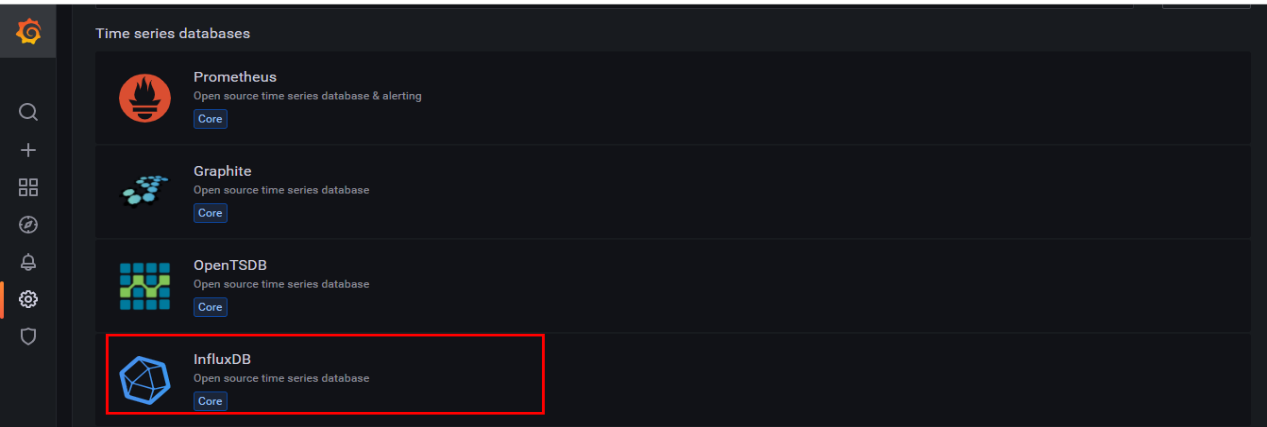
## 4、配置数据源

在grafana服务中

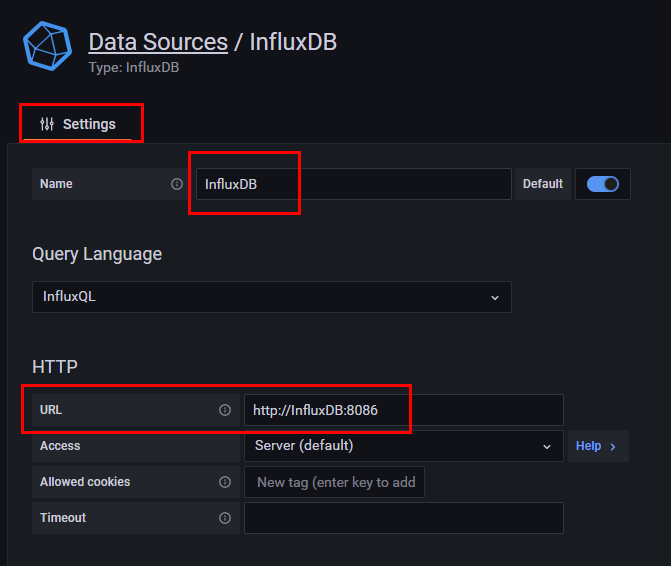
### （1）配置数据源

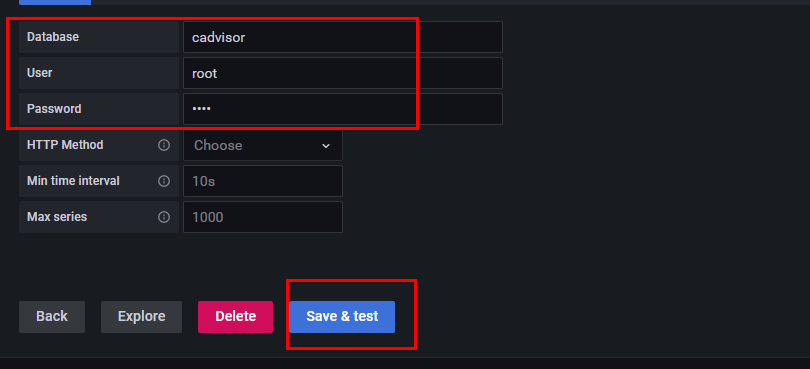


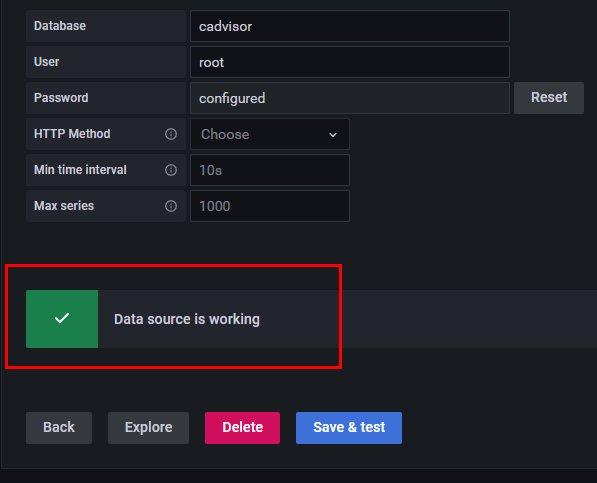
### （2）选择influxdb数据源



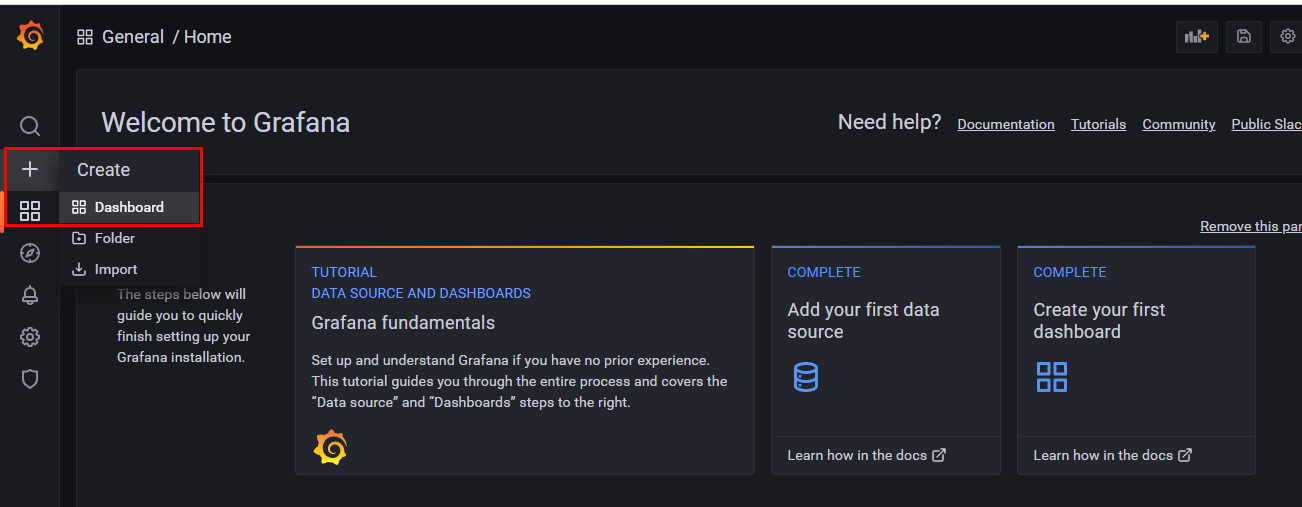
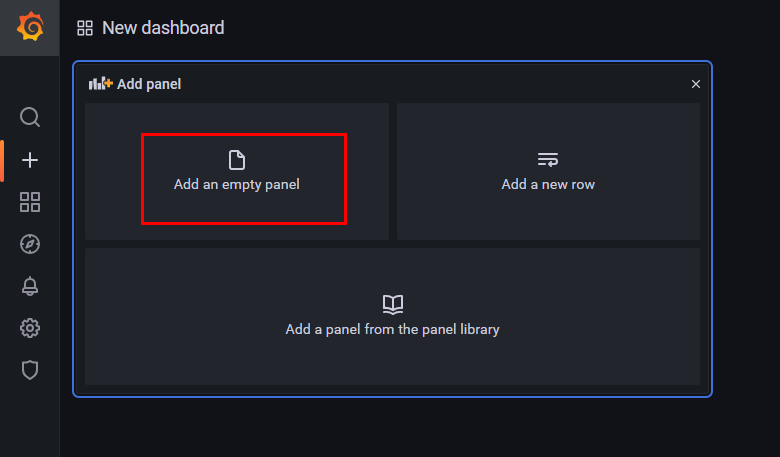
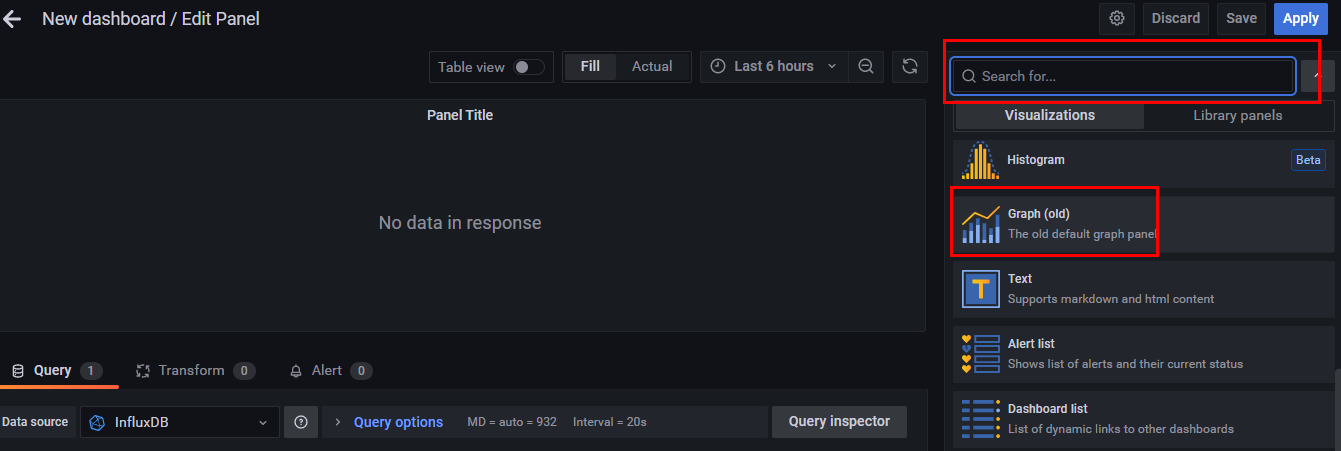
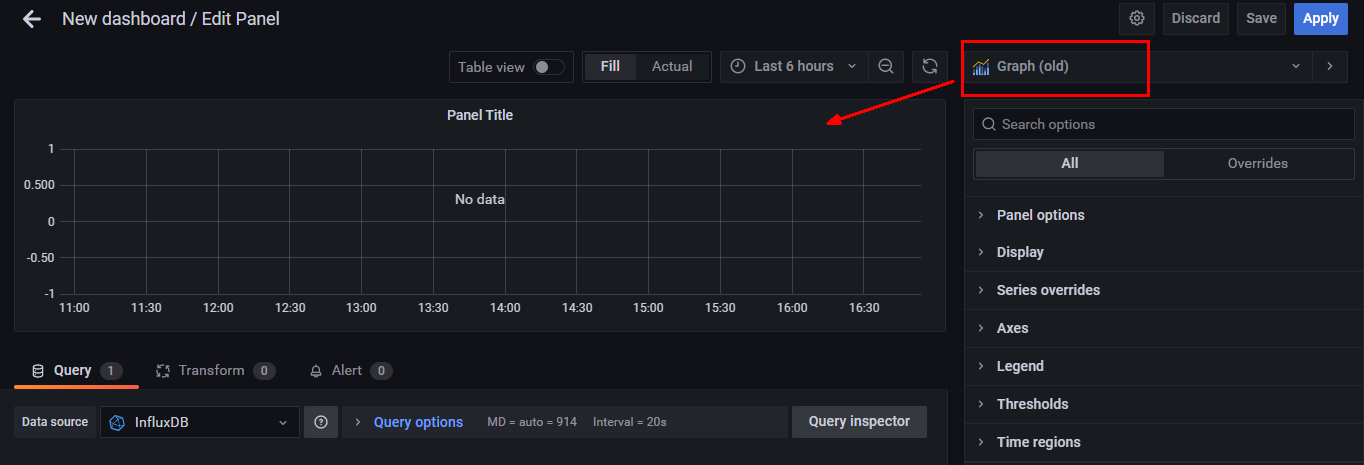
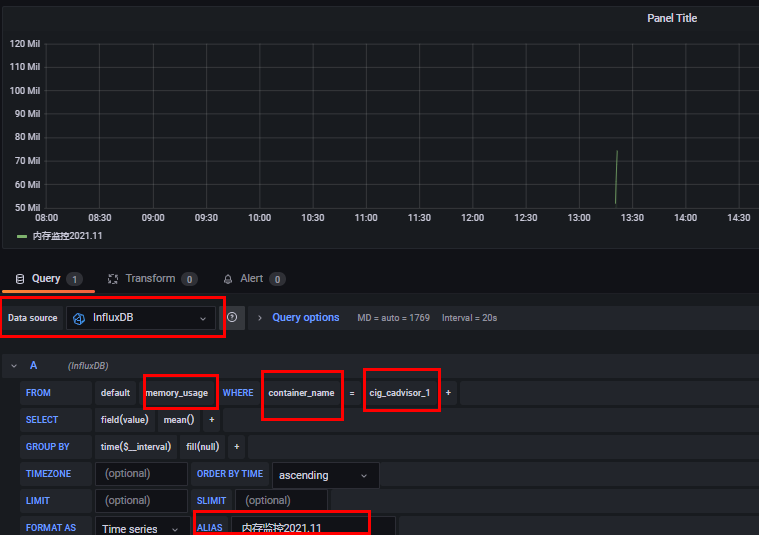
### （4）配置细节







### （5）配置面板panel

* 
* 
* 
* 
* 
* 