Docker Documentation

# 1.what is docker ?

Docker 是一个[开源](https://baike.baidu.com/item/%E5%BC%80%E6%BA%90/246339" \t "https://baike.baidu.com/item/Docker/_blank)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](https://baike.baidu.com/item/Linux" \t "https://baike.baidu.com/item/Docker/_blank) 机器上，也可以实现[虚拟化](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%8C%96" \t "https://baike.baidu.com/item/Docker/_blank)。容器是完全使用[沙箱](https://baike.baidu.com/item/%E6%B2%99%E7%AE%B1/393318" \t "https://baike.baidu.com/item/Docker/_blank)机制，相互之间不会有任何接口。

# 2.get docker

### 2.1 卸载旧版本

老版本的Docker被称为docker或docker-engine。如果安装了它们，请卸载它们以及相关的依赖项。

sudo yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-selinux \

docker-engine-selinux \

docker-engine

### 2.2使用存储库进行安装

#### 2.2.1安装所需的包

yum-utils提供了yum-config-manager 效用，并device-mapper-persistent-data和lvm2由需要 devicemapper存储驱动程序。

sudo yum install -y yum-utils \

device-mapper-persistent-data \

lvm2

#### 2.2.2使用以下命令设置稳定的存储库

即使您想从**边缘**或**测试**存储库安装构建，也总是需要**稳定的**存储库。

sudo yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

### 2.3安装DOCKER CE

#### 2.3.1安装最新版本的Docker CE，或转到下一步安装特定版本

sudo yum install docker-ce

#### 2.3.2安装特定版本

在生产系统上，您应该安装特定版本的Docker CE，而不是始终使用最新版本。列出可用的版本。此示例使用该sort -r命令按版本号对结果进行排序，从最高到最低并被截断。

yum list docker-ce --showduplicates | sort -r

docker-ce.x86\_64 17.12.ce-1.el7.centos docker-ce-stable

列表的内容取决于启用了哪些存储库，并且特定于您的CentOS .el7版本（在此示例中，由版本的后缀指示）。选择一个特定的版本进行安装。第二列是版本字符串。您可以使用整个版本字符串，但****您至少需要包含第一个连字符****。第三列是存储库名称，它指示软件包来自哪个存储库并且通过扩展其稳定性级别。要安装特定版本，请将版本字符串附加到包名称并用连字符（-）分隔。

****注意****：版本字符串是软件包名称加上第一个连字符的版本。在上面的例子中，完全限定的包名是docker-ce-17.06.1.ce。

$ sudo yum install <FULLY-QUALIFIED-PACKAGE-NAME>

#### 2.3.3启动docker

sudo systemctl start docker

# 3.卸载Docker Ce

### 3.1卸载docker包

sudo yum remove docker-ce

### 3.2 删除docker包

不会自动删除主机上的图像，容器，卷或自定义配置文件。删除所有图像，容器和卷：

sudo rm -rf /var/lib/docker

# 4.测试Docker版本

确保你有一个受支持的Docker版本：

$ docker --version

Docker version 17.12.0-ce, build c97c6d6

运行docker version（不--）或docker info查看关于docker安装的更多细节：

$ docker info

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Server Version: 17.12.0-ce

Storage Driver: overlay2

...

# 5.构建你的docker

Docker 有三大概念： ****镜像（image），容器（container），仓库。****

* 镜像：可以用来创建 Docker 容器（container）。
* 容器：是镜像 创建的运行实例。可以被启动，开始，停止，删除。每个容器相互隔离，保证安全。容器可以看成是简易版的 Linux 环境，并且镜像 是 只读的，容器可以创建一层可写层作为最上层。
* 仓库：是存放镜像文件的场所。仓库注册服务器包括多个仓库，每个仓库又包括了 多个镜像。仓库又分为公开仓库 和 私有仓库，最大的公开仓库是 Docker Hub。（可以理解成 github 这样的）

### 5.1构建镜像

#### 5.1.1 拉取基础父镜像

**提示：**

命令行的格式为：docker search 镜像名字

docker pull centos

#### 5.1.2 编写Dockerfile文件

**结构**   
DockerFile分为四部分组成：基础镜像信、维护者信息、镜像操作指令和容器启动时执行指令。例如：

#第一行必须指令基于的基础镜像

From ubutu

#维护者信息

MAINTAINER docker\_user docker\_user@mail.com

#镜像的操作指令

apt/sourcelist.list

RUN apt-get update && apt-get install -y ngnix

RUN echo "\ndaemon off;">>/etc/ngnix/nignix.conf

#容器启动时执行指令

CMD /usr/sbin/ngnix

**指令**   
1、From指令   
From 或者From :

DockerFile第一条必须为From指令。如果同一个DockerFile创建多个镜像时，可使用多个From指令（每个镜像一次）

2、MAINTAINER   
格式为maintainer ，指定维护者的信息

3、RUN   
格式为Run 或者Run [“executable” ,”Param1”, “param2”]   
前者在shell终端上运行，即/bin/sh -C，后者使用exec运行。例如：RUN [“/bin/bash”, “-c”,”echo hello”]   
每条run指令在当前基础镜像执行，并且提交新镜像。当命令比较长时，可以使用“/”换行。

4、CMD指令   
支持三种格式：   
CMD [“executable” ,”Param1”, “param2”]使用exec执行，推荐   
CMD command param1 param2，在/bin/sh上执行   
CMD [“Param1”, “param2”] 提供给ENTRYPOINT做默认参数。

每个容器只能执行一条CMD命令，多个CMD命令时，只最后一条被执行。

5、EXPOSE

格式为 EXPOSE […] 。

告诉Docker服务端容器暴露的端口号，供互联系统使用。在启动Docker时，可以通过-P,主机会自动分配一个端口号转发到指定的端口。使用-P，则可以具体指定哪个本地端口映射过来

例如：   
EXPOSE 22 80 8443

6、ENV

格式为 ENV 。 指定一个环境变量，会被后续 RUN 指令使用，并在容器运行时保持。

例如

ENV PG\_MAJOR 9.3

ENV PG\_VERSION 9.3.4

RUN curl -SL http://example.com/postgres-$PG\_VERSION.tar.xz | \

tar -xJC /usr/src/postgress && …

ENV PATH /usr/local/postgres-$PG\_MAJOR/bin:$PATH

7、ADD   
格式为 ADD 。

该命令将复制指定的 到容器中的 。 其中 可以是Dockerfile所在目录的一个相对路径；也可以是一个URL；还可以是一个tar文件（自动解压为目录）。则。

8、COPY

格式为 COPY 。

复制本地主机的 （为Dockerfile所在目录的相对路径）到容器中的 。

当使用本地目录为源目录时，推荐使用 COPY 。

9、ENTRYPOINT

两种格式：

ENTRYPOINT [“executable”, “param1”, “param2”]   
ENTRYPOINT command param1 param2 （shell中执行）。   
配置容器启动后执行的命令，并且不可被 docker run 提供的参数覆盖。

每个Dockerfile中只能有一个 ENTRYPOINT ，当指定多个时，只有最后一个起效。

10、VOLUME

格式为 VOLUME [“/data”] 。

创建一个可以从本地主机或其他容器挂载的挂载点，一般用来存放数据库和需要保持的数据等。

11、USER

格式为 USER daemon 。

指定运行容器时的用户名或UID，后续的 RUN 也会使用指定用户。

当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户，例如： RUN groupadd -r postgres && useradd -r -g postgres postgres 。要临时获取管理员权限可以使用 gosu ，而不推荐 sudo 。

12、WORKDIR

格式为 WORKDIR /path/to/workdir 。

为后续的 RUN 、 CMD 、 ENTRYPOINT 指令配置工作目录。

可以使用多个 WORKDIR 指令，后续命令如果参数是相对路径，则会基于之前命令指定的路径。例如

WORKDIR /a   
WORKDIR b   
WORKDIR c   
RUN pwd   
则最终路径为 /a/b/c 。

13、ONBUILD

格式为 ONBUILD [INSTRUCTION] 。

配置当所创建的镜像作为其它新创建镜像的基础镜像时，所执行的操作指令。

例如，Dockerfile使用如下的内容创建了镜像 image-A 。

[…]   
ONBUILD ADD . /app/src   
ONBUILD RUN /usr/local/bin/python-build –dir /app/src   
[…]   
如果基于A创建新的镜像时，新的Dockerfile中使用 FROM image-A 指定基础镜像时，会自动执行 ONBUILD 指令内容，等价于在后面添加了两条指令。

FROM image-A

#Automatically run the following

ADD . /app/src

RUN /usr/local/bin/python-build --dir /app/src

使用 ONBUILD 指令的镜像，推荐在标签中注明，例如 ruby:1.9-onbuild 。

#### 5.1.3 构建你的镜像

执行口令

Docker build -t [name] .

### 5.2 运行你的容器

**语法**

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

OPTIONS说明：

-a stdin: 指定标准输入输出内容类型，可选 STDIN/STDOUT/STDERR 三项；

**-d: 后台运行容器，并返回容器ID；**

-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；

--name="nginx-lb": 为容器指定一个名称；

--dns 8.8.8.8: 指定容器使用的DNS服务器，默认和宿主一致；

--dns-search example.com: 指定容器DNS搜索域名，默认和宿主一致；

-h "mars": 指定容器的hostname；

-e username="ritchie": 设置环境变量；

--env-file=[]: 从指定文件读入环境变量；

--cpuset="0-2" or --cpuset="0,1,2": 绑定容器到指定CPU运行；

-m :设置容器使用内存最大值；

--net="bridge": 指定容器的网络连接类型，支持 bridge/host/none/container: 四种类型；

--link=[]: 添加链接到另一个容器；

--expose=[]: 开放一个端口或一组端口；

！！！注意一个比较重要的参数

1 数据卷 -v 容器目录 或 -v 本地目录:容器目录  
数据卷是一个可供一个或多个容器使用的特殊目录，它绕过 UFS，可以提供很多有用的特性：  
数据卷可以在容器之间共享和重用  
对数据卷的修改会立马生效  
对数据卷的更新，不会影响镜像  
卷会一直存在，直到没有容器使用  
\*数据卷的使用，类似于 Linux 下对目录或文件进行 mount。  
docker run 命令使用 -v 标记来创建一个数据卷并挂载到容器里。  
在一次 run 中多次使用可以挂载多个数据卷。  
--创建一个数据卷(到容器的 /webapp 目录)  
$ sudo docker run -d -P --name web -v /webapp training/webapp python app.py  
--指定挂载一个本地主机的目录到容器中去  
$ sudo docker run -d -P --name web -v /src/webapp:/opt/webapp training/webapp python app.py  
上面的命令加载主机的 /src/webapp 目录到容器的 /opt/webapp 目录  
(本地目录的路径必须是绝对路径)

1.2 数据卷容器 --volumes-from 容器名称  
数据卷容器，其实就是一个正常的容器，专门用来提供数据卷供其它容器挂载的。  
首先，创建一个命名的数据卷容器 dbdata：  
$ sudo docker run -d -v /dbdata --name dbdata training/postgres echo Data-only container for postgres  
然后，在其他容器中使用 --volumes-from 来挂载 dbdata 容器中的数据卷。  
$ sudo docker run -d --volumes-from dbdata --name db1 training/postgres  
$ sudo docker run -d --volumes-from dbdata --name db2 training/postgres  
--volumes-from 会继承被挂载容器中的数据卷  
$ sudo docker run -d --name db3 --volumes-from db1 training/postgres  
\*\*注意!!：使用 --volumes-from 参数所挂载数据卷的容器自己并不需要保持在运行状态。  
\*\*注意!!:如果删除了挂载的容器（包括 dbdata、db1 和 db2），数据卷并不会被自动删除。  
如果要删除一个数据卷，必须在删除最后一个还挂载着它的容器时使用 docker rm -v 命令来指定同时删除关联的容器

1.3 备份、恢复、迁移数据卷  
备份  
从本地主机挂载当前到容器的 /backup 目录,使用了 tar 命令来将 dbdata 卷备份为本地的 /backup/backup.tar  
$ sudo docker run --volumes-from dbdata -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata  
恢复  
如果要恢复数据到一个容器，首先创建一个带有数据卷的容器 dbdata2。  
然后创建另一个容器，挂载 dbdata2 的容器，并使用 untar 解压备份文件到挂载的容器卷中。  
$ sudo docker run -v /dbdata --name dbdata2 ubuntu /bin/bash  
$ sudo docker run --volumes-from dbdata2 -v $(pwd):/backup busybox tar xvf /backup/backup.tar

### 5.3 docker其他指令

进入容器内部指令 Docker exec -it [容器名字] /bin/bash

Docker images

Docker rmi [images name]

Docker ps -a

Docker rm [container name]

Docker stop $(docker ps -a -q)

Docker rm $(docker ps -a -q)

Docker start container

打包镜像 docker save [镜像名] > [名字].tar

容器生命周期管理 — docker [run|start|stop|restart|kill|rm|pause|unpause]

容器操作运维 — docker [ps|inspect|top|attach|events|logs|wait|export|port]

容器rootfs命令 — docker [commit|cp|diff]

镜像仓库 — docker [login|pull|push|search]

本地镜像管理 — docker [images|rmi|tag|build|history|save|import]

其他命令 — docker [info|version]

容器生命周期管理

[run](http://www.runoob.com/docker/docker-run-command.html" \t "http://www.runoob.com/docker/_blank)

[start/stop/restart](http://www.runoob.com/docker/docker-start-stop-restart-command.html" \t "http://www.runoob.com/docker/_blank)

[kill](http://www.runoob.com/docker/docker-kill-command.html" \t "http://www.runoob.com/docker/_blank)

[rm](http://www.runoob.com/docker/docker-rm-command.html" \t "http://www.runoob.com/docker/_blank)

[pause/unpause](http://www.runoob.com/docker/docker-pause-unpause-command.html" \t "http://www.runoob.com/docker/_blank)

[create](http://www.runoob.com/docker/docker-create-command.html" \t "http://www.runoob.com/docker/_blank)

[exec](http://www.runoob.com/docker/docker-exec-command.html" \t "http://www.runoob.com/docker/_blank)

容器操作

[ps](http://www.runoob.com/docker/docker-ps-command.html" \t "http://www.runoob.com/docker/_blank)

[inspect](http://www.runoob.com/docker/docker-inspect-command.html" \t "http://www.runoob.com/docker/_blank)

[top](http://www.runoob.com/docker/docker-top-command.html" \t "http://www.runoob.com/docker/_blank)

[attach](http://www.runoob.com/docker/docker-attach-command.html" \t "http://www.runoob.com/docker/_blank)

[events](http://www.runoob.com/docker/docker-events-command.html" \t "http://www.runoob.com/docker/_blank)

[logs](http://www.runoob.com/docker/docker-logs-command.html" \t "http://www.runoob.com/docker/_blank)

[wait](http://www.runoob.com/docker/docker-wait-command.html" \t "http://www.runoob.com/docker/_blank)

[export](http://www.runoob.com/docker/docker-export-command.html" \t "http://www.runoob.com/docker/_blank)

[port](http://www.runoob.com/docker/docker-port-command.html" \t "http://www.runoob.com/docker/_blank)

容器rootfs命令

[commit](http://www.runoob.com/docker/docker-commit-command.html" \t "http://www.runoob.com/docker/_blank)

[cp](http://www.runoob.com/docker/docker-cp-command.html" \t "http://www.runoob.com/docker/_blank)

[diff](http://www.runoob.com/docker/docker-diff-command.html" \t "http://www.runoob.com/docker/_blank)

镜像仓库

[login](http://www.runoob.com/docker/docker-login-command.html" \t "http://www.runoob.com/docker/_blank)

[pull](http://www.runoob.com/docker/docker-pull-command.html" \t "http://www.runoob.com/docker/_blank)

[push](http://www.runoob.com/docker/docker-push-command.html" \t "http://www.runoob.com/docker/_blank)

[search](http://www.runoob.com/docker/docker-search-command.html" \t "http://www.runoob.com/docker/_blank)

本地镜像管理

[images](http://www.runoob.com/docker/docker-images-command.html" \t "http://www.runoob.com/docker/_blank)

[rmi](http://www.runoob.com/docker/docker-rmi-command.html" \t "http://www.runoob.com/docker/_blank)

[tag](http://www.runoob.com/docker/docker-tag-command.html" \t "http://www.runoob.com/docker/_blank)

[build](http://www.runoob.com/docker/docker-build-command.html" \t "http://www.runoob.com/docker/_blank)

[history](http://www.runoob.com/docker/docker-history-command.html" \t "http://www.runoob.com/docker/_blank)

[save](http://www.runoob.com/docker/docker-save-command.html" \t "http://www.runoob.com/docker/_blank)

[import](http://www.runoob.com/docker/docker-import-command.html" \t "http://www.runoob.com/docker/_blank)

info|version

[info](http://www.runoob.com/docker/docker-info-command.html" \t "http://www.runoob.com/docker/_blank)

[version](http://www.runoob.com/docker/docker-version-command.html" \t "http://www.runoob.com/docker/_blank)

# 6 实例

### 6.1 Dcoker下springboot镜像Dockerfile Demo

#### 进入虚拟机并且安装

#### vi DockerFile

FROM centos:latest

MAINTAINER zhanghy

ADD jdk-8u151-linux-x64.tar.gz /usr/local/

ADD dp-exhibit-fontui-1.0.0.jar /opt/exhibit/

ENV JAVA\_HOME /usr/local/jdk1.8.0\_151

ENV CLASSPATH $JAVA\_HOME/lib.jar:$JAVA\_HOME/lib/tools.jar

ENV PATH $PATH:$JAVA\_HOME/bin

EXPOSE 8088

ENTRYPOINT ["java","-jar","/opt/exhibit/dp-exhibit-fontui-1.0.0.jar"]

#### 放入所需文件

将需要的包dp-exhibit-fontui-1.0.0.jar与jdk-8u151-linux-x64.tar.gz拷贝到当前目录

#### 4.执行build命令

Docker build -t springboot .

#### 执行run命令

Docker run -p 8088:8088 -v /home/logs:/opt/exhibit/logs --name exhibit springboot

### 6.2 Dcoker下Tomcat镜像Dockerfile Demo

Dockerfile文件如下

FROM centos:latest

MAINTAINER zhanghy

ADD jdk-8u151-linux-x64.tar.gz /usr/local/

ADD apache-tomcat-8.5.28.tar.gz /usr/local/

ENV JAVA\_HOME /usr/local/jdk1.8.0\_151

ENV CLASSPATH $JAVA\_HOME/lib.jar:$JAVA\_HOME/lib/tools.jar

ENV CATALINA\_HOME /usr/local/apache-tomcat-8.5.28

ENV CATALINA\_BASE /usr/local/apache-tomcat-8.5.28

ENV HOME /usr/local/apache-tomcat-8.5.28

ENV PATH $PATH:$JAVA\_HOME/bin:$CATALINA\_HOME/lib:$CATALINA\_BASE/bin

EXPOSE 8080

执行构建容器命令

Docker build -t tomcat .

执行运行容器命令

Docker run -p 8080:8080 -v /home/logs:/usr/local/apache-tomcat-8.5.28/logs --name tomcattest tomcat usr/local/apache-tomcat-8.5.28/bin/catalina.sh run