docker 相关命令汇总

操作容器的命令

镜像中的容器启动之后可以在 docker 中操作和查看容器的信息

- docker ps 查看运行的容器，如果想查看全部加上参数-a 即可
- docker create 完整的镜像名字# 创建容器但是不启动它
- docker run 完整的镜像名字# 创建并启动容器
- docker stop CONTAINER ID# 停止容器运行
- docker start CONTAINER ID# 启动停止状态的容器
- docker restart CONTAINER ID# 重启容器
- docker rm CONTAINER ID# 删除容器

获取容器相关信息

- docker ps # 显示状态为运行（Up）的容器
- docker ps -a # 显示所有容器,包括运行中（Up）的和退出的(Exited)
- docker inspect # 深入容器内部获取容器所有信息

- docker logs # 查看容器的日志(stdout/stderr)
- docker events # 得到 docker 服务器的实时的事件
- docker port # 显示容器的端口映射
- docker top # 显示容器的进程信息
- docker diff # 显示容器文件系统的前后变化

导出容器

- docker cp # 从容器里向外拷贝文件或目录
- docker export # 将容器整个文件系统导出为一个 tar 包，不带 layers、tag 等信息

镜像操作命令

- docker images # 显示本地所有的镜像列表
- docker import # 从一个 tar 包创建一个镜像，往往和 export 结合使用
- docker build # 使用 Dockerfile 创建镜像（推荐）
- docker commit # 从容器创建镜像
- docker rmi # 删除一个镜像
- docker load # 从一个 tar 包创建一个镜像，和 save 配合使用
- docker save # 将一个镜像保存为一个 tar 包，带 layers 和 tag 信息
- docker history # 显示生成一个镜像的历史命令
- docker tag # 为镜像起一个别名

镜像仓库命令

- docker login # 登录到一个 registry
- docker search # 从 registry 仓库搜索镜像
- docker pull # 从仓库下载镜像到本地
- docker push # 将一个镜像 push 到 registry 仓库中

Dockerfile 命令

（1）FROM（指定基础 image）

构建指令，必须指定且需要在 Dockerfile 其他指令的前面。后续的指令都依赖于该指令指定的 image。FROM 指令指定的基础 image 可以是官方远程仓库中的，也可以位于本地仓库。

该指令有两种格式：

FROM <image>

指定基础 image 为该 image 的最后修改的版本。或者：

FROM <image>:<tag>

指定基础 image 为该 image 的一个 tag 版本。

（2）MAINTAINER（用来指定镜像创建者信息）

构建指令，用于将 image 的制作者相关的信息写入到 image 中。当我们对该 image 执行 docker inspect 命令时，输出中有相应的字段记录该信息。

格式：

MAINTAINER <name>

（3）RUN（安装软件用）

构建指令，RUN 可以运行任何被基础 image 支持的命令。如基础 image 选择了 ubuntu，那么软件管理部分只能使用 ubuntu 的命令。

该指令有两种格式：

RUN <command> (the command is run in a shell - `/bin/sh -c`)

RUN ["executable", "param1", "param2" ...] (exec form)

（4）CMD（设置 container 启动时执行的操作）

设置指令，用于 container 启动时指定的操作。该操作可以是执行自定义脚本，也可以是执行系统命令。该指令只能在文件中存在一次，如果有多个，则只执行最后一条。

该指令有三种格式：

CMD ["executable","param1","param2"] (like an exec, this is the preferred form)

CMD command param1 param2 (as a shell)

当 Dockerfile 指定了 ENTRYPOINT，那么使用下面的格式：

CMD ["param1","param2"] (as default parameters to ENTRYPOINT)

ENTRYPOINT 指定的是一个可执行的脚本或者程序的路径，该指定的脚本或者程序将会以 param1 和 param2 作为参数执行。所以如果 CMD 指令使用上面的形式，那么 Dockerfile 中必须要有配套的 ENTRYPOINT。

（5）ENTRYPOINT（设置 container 启动时执行的操作）

设置指令，指定容器启动时执行的命令，可以多次设置，但是只有最后一个有效。

两种格式:

- ENTRYPOINT ["executable", "param1", "param2"] (like an exec, the preferred fo

rm)

- ENTRYPOINT command param1 param2 (as a shell)

该指令的使用分为两种情况，一种是独自使用，另一种和 CMD 指令配合使用。

当独自使用时 ,如果你还使用了 CMD 命令且 CMD 是一个完整的可执行的命令 ,那么 CMD 指令和 ENTRYPOINT 会互相覆盖只有最后一个 CMD 或者 ENTRYPOINT 有效。

CMD 指令将不会被执行，只有 ENTRYPOINT 指令被执行

CMD echo “Hello, World!”

ENTRYPOINT ls -l

另一种用法和 CMD 指令配合使用来指定 ENTRYPOINT 的默认参数，这时 CMD 指令不是一个完整的可执行命令，仅仅是参数部分；ENTRYPOINT 指令只能使用 JSON 方式指定执行命令，而不能指定参数。

FROM ubuntu

CMD ["-l"]

ENTRYPOINT ["/usr/bin/ls"]

（6）USER（设置 container 容器的用户）

设置指令，设置启动容器的用户，默认是 root 用户。

指定 memcached 的运行用户

ENTRYPOINT ["memcached"]

USER daemon

或

ENTRYPOINT ["memcached", "-u", "daemon"]

（7）EXPOSE（指定容器需要映射到宿主机器的端口）

设置指令，该指令会将容器中的端口映射成宿主机器中的某个端口。当你需要访问容器的时候，可以不是用容器的 IP 地址而是使用宿主机器的 IP 地址和映射后的端口。要完成整个操作需要两个步骤，首先在 Dockerfile 使用 EXPOSE 设置需要映射的容器端口，然后在运行容器的时候指定-p 选项加上 EXPOSE 设置的端口，这样 EXPOSE 设置的端口号会被随机映射成宿主机器中的一个端口号。也可以指定需要映射到宿主机器的那个端口，这时要确保宿主机器上的端口号没有被使用。EXPOSE 指令可以一次设置多个端口号，相应的运行容器的时候，可以配套的多次使用-p 选项。

格式:

EXPOSE <port> [<port>...]

映射一个端口

EXPOSE port1

相应的运行容器使用的命令

docker run -p port1 image

映射多个端口

EXPOSE port1 port2 port3

相应的运行容器使用的命令

docker run -p port1 -p port2 -p port3 image

还可以指定需要映射到宿主机器上的某个端口号

docker run -p host_port1:port1 -p host_port2:port2 -p host_port3:port3 image

端口映射是 docker 比较重要的一个功能，原因在于我们每次运行容器的时候容器的 IP 地址不能指定而是在桥接网卡的地址范围内随机生成的。宿主机器的 IP 地址是固定的，我们可以将容器的端口的映射到宿主机器上的一个端口 ,免去每次访问容器中的某个服务时都要查看容器的 IP 的地址。对于一个运行的容器，可以使用 docker port 加上容器中需要映射的端口和容器的 ID 来查看该端口号在宿主机器上的映射端口。

（8）ENV（用于设置环境变量）

构建指令，在 image 中设置一个环境变量。

格式:

ENV <key> <value>

设置了后，后续的 RUN 命令都可以使用，container 启动后，可以通过 docker inspect 查看这个环境变量，也可以通过在 docker run --env key=value 时设置或修改环境变量。

假如你安装了 JAVA 程序，需要设置 JAVA_HOME，那么可以在 Dockerfile 中这样写：

ENV JAVA_HOME /path/to/java/dirent

（9）ADD（从 src 复制文件到 container 的 dest 路径）

构建指令，所有拷贝到 container 中的文件和文件夹权限为 0755，uid 和 gid 为 0；如果是一个目录，那么会将该目录下的所有文件添加到 container 中，不包括目录；如果文件是可识别的压缩格式 ,则 docker 会帮忙解压缩(注意压缩格式) ;如果<src>是文件且<dest>中不使用斜杠结束，则会将<dest>视为文件，<src>的内容会写入<dest>；如果<src>是文件且<dest>中使用斜杠结束，则会<src>文件拷贝到<dest>目录下。

格式:

ADD <src> <dest>

<src> 是相对被构建的源目录的相对路径，可以是文件或目录的路径，也可以是一个远程的文件 url;

<dest> 是 container 中的绝对路径

（10）VOLUME（指定挂载点)）

设置指令，使容器中的一个目录具有持久化存储数据的功能，该目录可以被容器本身使用，也可以共享给其他容器使用。我们知道容器使用的是 AUFS ,这种文件系统不能持久化数据，当容器关闭后，所有的更改都会丢失。当容器中的应用有持久化数据的需求时可以在 Dockerfile 中使用该指令。

格式:

VOLUME ["<mountpoint>"]

FROM base

VOLUME ["/tmp/data"]

运行通过该 Dockerfile 生成 image 的容器，/tmp/data 目录中的数据在容器关闭后，里面的数据还存在。例如另一个容器也有持久化数据的需求，且想使用上面容器共享的 /tmp/data 目录，那么可以运行下面的命令启动一个容器：

docker run -t -i -rm -volumes-from container1 image2 bash

container1 为第一个容器的 ID，image2 为第二个容器运行 image 的名字。

（11）WORKDIR（切换目录）

设置指令，可以多次切换(相当于 cd 命令)，对 RUN,CMD,ENTRYPOINT 生效。

格式:

WORKDIR /path/to/workdir

在 /p1/p2 下执行 vim a.txt

WORKDIR /p1 WORKDIR p2 RUN vim a.txt

（12）ONBUILD（在子镜像中执行）

ONBUILD <Dockerfile 关键字>

ONBUILD 指定的命令在构建镜像时并不执行，而是在它的子镜像中执行。

百度搜索：小强测试品牌

官网：http://xqtesting.sxl.cn

博客：http://xqtesting.blog.51cto.com

挨踢脱口秀，技术与逗比的融合（荔枝 FM 或喜马拉雅手机客户端可搜索订阅）：

http://www.lizhi.fm/200893

扫码关注微信公众号