Docker容器

容器的优点

- 容器能够和主机的操作系统共享资源，因此它的效率高出很多数量级。启动和停止只是一瞬间。相比在主机上直接运行，容器的性能损耗低，几乎可以说是0损耗。
- 容器具有可移植性。
- 容器是轻量级的。开发者能够运行数十个容器，能够模拟分布式系统运行环境的真实情况。

迈出第一步

```
docker run debian echo "Hello world"
```

运行debian容器，如果没有会进行下载，将会输出hello world

```
docker run -i -t debian /bin/bash

exit
```

表示请求Docker提供一个容器中的Shell。

-i、-t表示我们想要一个附有tty(终端)的交互会话

/bin/bash： 表示我们想要获取一个 bash shell

```
docker run -h "终端名字" -i -t debian /bin/bash // 为终端取了名字而已
// docker run -h --rm "终端名字" -i -t debian /bin/bash --rm 表示容器退出的时候，和容器相关的文件系统会被一起删除。
```

```
docker inspect +容器名称  // 获取该容器的各种参数信息
// 信息太多过滤一下 ip地址
docker inspect 容器名称 | grep IPAddress

docker inspect --format {{.NetworkSettings.IPAddress}} 容器名称

// 查看容器的那些文件被改过
docker diff 容器名称

// 删除容器
docker rm 容器名称

// 清理已经停止的容器。这个命令很常用，可以写成脚本，或者定义为alias。或者
docker rm -v $(docker ps -aq -f status=exited)

```

cowsay(奶牛说话)

```
 docker run -it --name cowsay --hostname cowsay debian bash

 apt-get update

 apt-get install -y cowsay fortune
 // 以上是安装

 // 执行这个容器
 /usr/games/fortune | /usr/games/cowsay


docker commit cowsay test/cowsayimage 将容器装换成镜像，cowsay 是容器的名字，cowsayimage是新容器的名字，/test是存放的位置

docker run test/rowsayimage /usr/games/cowsay
```

> 如果我们想把这个创建的镜像和别人分享的话，那么必须要从开始做起，重复步骤不是一个好的事情，如果步骤多了后就容易出错。解决方法：Dockerfile

Dockerfile(创建镜像自动化)

```
mkdir cowsay

cd cowsay

touch Dockerfile

// 将下面的FROM...RUN... 放在Dockefile中
// FROM指令指定初始镜像 使用debian 并且指定他的版本是wheezy
FROM debian:wheezy
RUN apt-get update && apt-get install -y cowsay fortune
```

> 所有的Dockerfile一定要有From指令作为第一个非注释命令。 RUN命令指定的shell命令，是将要在镜像里执行的，这个命令和之前一样，安装 cowsay和fortune

> FROM指定基础镜像，镜像不存在会在docker hub上拉取
>
> COPY把宿主机中的文件复制到镜像中去
>
> ADD类似COPY命令，支持URL路径----如果可以访问网络的话，会访问网络下载。如果是从本地路径添加一个归档文件，那么它会被自动解压。
>
> RUN在容器内执行命令，常用于提前安装编译软件。
>
> ENV用于为镜像定义所需的环境变量
>
> EXPOSE暴露端口
>
> CMD容器启动时用到的命令
>
> WORKDIR为后续RUN、CMD、或着ENTRYPOINT命令指定工作目录

> VOLUME创建一个可以从本地宿主机或其他容器挂载的挂载点:持久化存储
>
> ENTRYPOINT：设置一个于容器启动时运行的可执行文件/脚本（以及默认参数）。任何CMD指令或docker run命令中镜像名称之后的参数，将作为参数传给这个可执行文件。ENTRYPOINT指令通常用于提供“启动”脚本，目的是在解析参数之前，对变量和服务进行初始化。
>
> MAINTAINER：作者标识：可以使用 docker inspect -f {{.Author}} image 来查看作者信息
>
> ONBUILD： 指定当镜像被用作另一个镜像的基础镜像时将会执行的指令。对于处理一些将要添加到子镜像的数据，这个指令将会非常有用（例如，把代码从一个已选定的目录中复制出来，并在执行构建脚本时使用它）
>
> USER: 设置任何后续的RUN、CMD或ENTRYPOINT指令执行时所用的用户（用户名或UID）。 请注意，UID在主机和容器中是相同的，但用户名则可能被分配到不同的UID，导致设置权限时变得复杂。

```
docker build -t test/cowsay-dockerfile .
// 创建成功后 就可以像以前一样运行镜像    . 表示打包的上下文，目前表示的是当前路径
docker run test/cowsay-dockerfile /usr/games/cowsay "moo"
```

> 不要使用 “/”作为构建上下文
>
> 因为构建环境上下文都被放进去一个tar文件，然后传给Docker守护进程，因此你绝对不会希望使用一个含有大量文件的目录。因为客户端需要把所有的文件归档，然后传给守护进程，所以像/home/user 、downloads/ 都会花销非常多的时间来处理。

镜像、容器和联合文件系统

Docker使用的核心技术：联合文件系统(联合挂载)。联合文件系统允许多个文件系统叠加，表现为一个单一的文件系统。文件夹中的文件可以来自多个文件系统，但是如果两个文件的路径完全一致，那么最后挂载的文件会覆盖较早的那个。Docker支持的文件系统：AUFS、Overlay、devicemapper、BTRFS及ZFS。

Docker的镜像是由多个不同的 “层”layer组成，每一个层都是一个只读文件系统。Dockerfile里的每个指令都会创建一个新的层，每个新的层位于前一个层之上，当一个镜像被转化成一个容器的时候，docker引擎会在镜像之上添加一个处于最上层的可读写文件系统。为了不必要的层数使得镜像比较臃肿，尽可能的减少层数。比如，RUN把多个命令拼在一起。

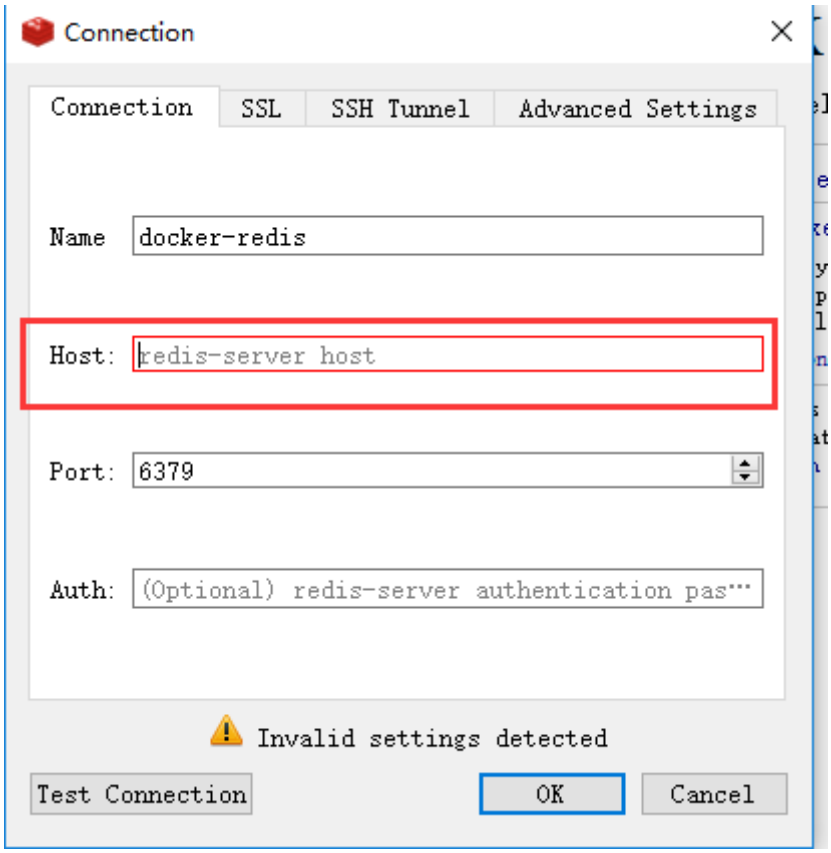
容器状态：已创建、重启中、运行中、已暂停、已退出。

```
docker info 可以查看docker的基本信息。
```

使用Redis官方镜像

```
docker search redis

docker pull redis 下载镜像

启动Redis容器（第一次要用-d）
docker run --name myredis -d redis -d参数表示容器在后台运行。Docker照常启动容器。不一样的是，这次不会把容器的输出打印出来，只返回容器ID，然后就会退出。容器仍在后台运行，可以通过 docker logs 查看容器的输出

---得到的数据 c601e351688f0c350580bfe280cc130f25552cc06ac47f7b2165fe8dcd7af6a2

```

启动Redis后，我们可以在本地主机连接这个虚拟机的redis；



> ip地址 使用 docker inspect myredis | grep IPAddress
>
> 或者 docker inspect --format {{.NatworkSettings.IPAddress}} myredis
>
> jiueky

```
docker run --rm -it --link myredis:redis redis /bin/bash
--link把两个东西连接在一起
--link myredis:redis 告诉docker新的容器与现存的myredis容器连接起来，并且在新容器中以“redis”作为“myredis”容器的主机名。为了实现这一点。Docker会在新容器的/etc/hosts里面添加一个新条目，把redis指向“myredis”的IP地址。

redis-cli -h redis -p 6379 打开redis客户端

```

关于Redis数据持久化的

- 要么在dockerFile中使用Volume VOLUME /data 生成一个数据卷
- 要么在启动的时候使用 -v命令 docker run -v /data test/webserver

```
Redis容器备份

docker run --rm --volumes-from myredis -v $(pwd)/backup:/backup \debian cp /data/dump.rdb /backup/

```

docker基本概念

.dockerignore文件： 为了构建环境的上下文中排除不必要的文件.

.git :上下文的根目录下的 .git 文件或者目录会被排除，但允许出现在子目录中(即 .git 会被排除，但 dir1/.git不会)

*/.git： 只排除第一层目录下的.git 文件或者目录(即 dir1/.git 会被排除，但.git 和 dir1/dir2/.git 不会)。

*/ * /.git： 只排除第二层目录下的.git文件

*.sw?： 当前层的 匹配的会被排除，其他的不会。

镜像层

```
docker history mongo:latest 查看镜像所有层

```

缓存

使用缓存能够增加docker的效率，但是满足以下条件：

- 上一个命令能够在缓存中能够找到，并且
- 缓存中存在一个镜像层，而他的指令和你的指令一模一样，父层也完全相同(即使指令中出现一些无关重要的空格也会使缓存失效)

> 关于COPY ADD ，如果他们引用的文件的校验和 或者 元数据发生变化，那么缓存也失效。意味着：尽管每次调用的结果可能都不一样的RUN指令，也仍然会被缓存。如果 下载文件、apt-get update、复制源码库要十分注意

```
docker build --no-cache 使得缓存失效。
也可以：

在dockerfile中(不推荐)
ENV UPDATED_ON "14:12 17 February 2015"
RUN git clone


```

使容器和世界相连

如果正在容器运行一个web容器，如何使外界能访问它

```
docker run -d -p 8000:80 nginx  80是当前端口，8000是转发出去的端口

curl localhost:8000 访问一下

或者：
ID=$(docker run -d -p nginx)
docker port $ID 80    将会自动分配一个端口
curl localhost:33771
```

> -p 选项主要优点就是你不需要负责跟踪那些端口被费配过，如果几个容器同时发布了不同端口，这就变得比较重要。

容器互联

docker的连接(link) 是允许同一主机上的容器互相通信的最简单方法，Dokcer默认的联网模型时，容器之间的通信通过Docker的内部网络，这意味着主机网络无法看见这些通信。

连接的初始化是通过：docker run 命令时传入 --link CONTAINER(容器名称):ALIAS(别名，主容器用来称呼目标容器的一个本地名称)。

```
docker run -d --name yangxinredis redis

docker run --link yangxinredis:redis debian env
```

使用数据卷和数据容器管理数据

在执行Docker时，通过 -v选项来宣告一个数据卷：

```
docker -run -it --name container-test -h CONTAINER -v /data debian
/bin/bash

docker inspect -f {{.Mounts}} container-test
# [{volume f27b3ec457d91cd1df19b041c468ea7598b57c0b8bf75aadd30d4bf38baeb177
/var/lib/docker/volumes/f27b3ec457d91cd1df19b041c468ea7598b57c0b8bf75aadd30
d4bf38baeb177/_data /data local  true }]
向该数据卷中添加数据
touch /var/../../../_data/test-file


指定数据卷的第二种方式
在dockerfile中：
COLUME /data
```

在dockerfile中设置数据卷权限

```
FROM debian
RUN useradd foo
```

```
VOLUME /data
RUN touch /data/x
RUN chown -R foo:foo /data
事实上上面的dockerfile并不能实现权限的处理。因为RUN是在创建的临时层运行，运行完了后，就会被删除，那么此举毫无意义。

下面就可以避免这个问题
FROM debian
RUN useradd foo
RUN mkdir /data && touch /data/x
RUN chown -R foo:foo /data
VOLUME /data

docker run -v /home/adrian/data:/data debian ls /data
表示把主机的 /home/adrian/data目录挂载到容器的/data目录。容器能够使用/home、adrian/data目录中的任何文件.
特点：不会被真正的删除


```

共享数据

```
docker run -it -h NEWCONTAINER --volumes-from container-test debian /bin/bash
将container-test的数据卷关联到新的容器中共享
```

数据容器

创建数据容器，唯一的目的就是与其他容器分享数据。这个方法的优点在于，他提供了一个方便的命名空间，使数据卷很容易通过--volumes-from 命令加载

```
docker run --name dbdata mysql echo "data-only container for mysql "
创建一个mysql数据库容器后，最后输出

让其他容器可以使用这个数据卷
docker run -d --volumes-from dbdata --name db1 mysql

删除数据卷
docker rm -v
或者
docker run 命令执行时带有 --rm选项
```

docker常用命令

布尔型选项

```
docker logs --help
docker logs --help = false
```

run命令

选项允许用户配置镜像运行的方式、覆盖Dockerfile设置、配置联网，以及设置容器的权限和资源。

- -a ,--attach 把指定的数据流(如STDOUT)连接倒终端。如果没有指定，则默认连接倒stdout和stderr。若数据流未指定，而容器以交互模式(-i)启动，则stdin也会被连接倒终端

  此选项与-d选项不兼容
- -d, --detach 使容器在"分离"模式下运行。容器会在后台运行，而命令的返回值是容器的ID。
- -i , --interactive 保持stdin打开(即使它没有被被连接至终端)，一般和-t同时使用，做启动交互式会话容器
- -restart。配置Docker在什么情况下尝试重启已经退出的容器。参数为 no意味着永远不会尝试重新启动容器。always 指不管退出状态是什么，总会尝试重新启动。on-failure仅当退出状态不为0的时候才会尝试重启，并且可以追加一个可选参数，指定重启次数

```
docker run --restart on-failure:10 postgres
```

- --rm :退出的时候自动删除容器。不能与-d同时使用
- -t --tty 分配一个伪终端。通常和-i使用，启动交互式容器
- -e --env 设置容器内的环境变量

```
docker run -e var1=val -e var2="val 2" debian env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=b15f833d65d8 var1=val var2=val 2  HOME=/root
```

- -h --hostname NAME 设置容器的unix主机名为NAME

```
docker run -h "myhost" debian hostname
```

- --name NAME 把NAME设置为容器的名称，以后docker的其他命令便可以使用这个名称来称呼这个容器。
- -v --volumes 设置数据卷
- --volumes-from 挂载指定容器拥有的数据卷
- --expose 指定容器将会使用的端口或端口范围，但并不会把端口打开。只有与-P参数同时使用，以及在连接容器时，才有真正意义。
- --link 建立一个与指定容器连接的内部网络接口。
- -p --publish “发布”容器的端口，使主机能访问它
- --enrtrypoint 把参数指定为容器的入口。将会覆盖Dockerfile中的ENTRYPOINT指令
- -u 设置运行时候所使用的用户
- -w 将参数路径设置为容器的工作命令

容器管理

docker attach [OPTIONS] CONTAINER

attach 命令允许用户查看容器内的主进程，或者与他们交互

```
ID=$()
```