### 启动

* 1. 阿里云镜像加速

容器镜像服务->镜像加速器

* 1. 运行交互式容器

docker run -d -p 80:80 httpd

* 1. 启动容器

创建一个以进程方式运行的容器

docker run -d ubuntu:15.10 /bin/sh -c "while true; do echo hello world; sleep 1; done"

-d:让容器在后台运行

返回容器id

docker ps 查看正在运行的容器（会自动分配容器名称NAMES）

参数： -a 查看所有容器

-q : 静默模式，只显示容器编号

停用所有运行中的容器：docker stop $(docker ps -q)

删除所有容器：docker rm $(docker ps -aq)

一条命令停用并删除所有容器：docker stop $(docker ps -q) & docker rm $(docker ps -aq)

docker logs -f 容器id/容器名称NAMES 查看容器内的标准输出(类似tail -f 输出)

docker stop 容器id/容器名称NAMES 停止容器

docker start 容器id/容器名称NAMES 启动容器

docker restart 容器id/容器名称NAMES 重启容器

docker port 容器id/容器名称NAMES 查看端口映射情况

docker top 容器id/容器名称NAMES 查看容器内部运行的进程

docker inspect 容器id/容器名称NAMES 查看容器的配置和状态信息

docker rm 容器id/容器名称NAMES 移除web应用容器(移除前必须先stop)

* 1. 镜像

docker pull ubuntu:13.10 下载一个镜像

docker images 列出本机上的镜像

docker search 镜像名 搜索镜像

* 1. 容器连接

docker run -d -P training/webapp python app.py

-d :容器在后台运行

-P: 容器内部端口随机映射到主机的高端口。

docker run -d -p 5000:5000 training/webapp python app.py

-p: 容器内部端口绑定到指定的主机端口

docker run -d -p 127.0.0.1:5001:5000 training/webapp python app.py

指定容器绑定的网络地址127.0.0.1

容器命名：

创建重启的时候，docker会自动对它进行命名，也可使用--name来指定名称

### 常用命令

在运行的容器内执行命令：

docker exec [OPTIONS] [容器ID或容器名称] [具体命令]

-d :分离模式: 在后台运行

-i :即使没有附加也保持STDIN 打开

-t :分配一个伪终端

查看所有容器或某个容器资源使用实时统计：

docker stats [容器ID或容器名称]，容器参数可选

### 镜像

#### base镜像

base镜像底层直接用 Host 的 kernel，自己只需要提供 rootfs 就行了

注意：1、base 镜像只是在用户空间与发行版一致，kernel 版本与发型版是不同的。

例如 CentOS 7 使用 3.x.x 的 kernel，如果 Docker Host 是 Ubuntu 16.04（比如我们的实验环境），那么在 CentOS 容器中使用的实际是是 Host 4.x.x 的 kernel。

ps : uname -r 查看操作系统发行版本

2、容器只能使用 Host 的 kernel，并且不能修改

#### 镜像分层结构

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。

镜像层数量可能会很多，所有镜像层会联合在一起组成一个统一的文件系统。如果不同层中有一个相同路径的文件，比如 /a，上层的 /a 会覆盖下层的 /a，也就是说用户只能访问到上层中的文件 /a。在容器层中，用户看到的是一个叠加之后的文件系统。

添加文件

在容器中创建文件时，新文件被添加到容器层中。

读取文件

在容器中读取某个文件时，Docker 会从上往下依次在各镜像层中查找此文件。一旦找到，打开并读入内存。

修改文件

在容器中修改已存在的文件时，Docker 会从上往下依次在各镜像层中查找此文件。一旦找到，立即将其复制到容器层，然后修改之。

删除文件

在容器中删除文件时，Docker 也是从上往下依次在镜像层中查找此文件。找到后，会在容器层中记录下此删除操作。

**容器层记录对镜像的修改，所有镜像层都是只读的，不会被容器修改，所以镜像可以被多个容器共享**

#### 构建镜像

##### docker commit

运行容器->修改容器->将容器保存为新**镜像**

docker run -it ubuntu

安装软件

docker ps 查看当前运行的容器

docker commit oldName newName

##### Dockerfile构建镜像

1. docker build -t 'continer name' .

-t: 容器命名

. : 指明build context（build 环境） 为当前目录，Docker会从build context中查找Dockerfile文件；或用-f 指明Dockerfile文件位置

1. docker将bulid context中的所有文件发送给docker deamon(docker服务器)
2. 执行Dockerfile中的命令：FROM 为base镜像，RUN会进行一系列操作（会在临时创建的镜像上进行），然后将容器保存为镜像，并删除临时容器。

#### 镜像缓存特性

Docker 会缓存已有镜像的镜像层，构建新镜像时，如果某镜像层已经存在，就直接使用，无需重新创建

默认使用缓存，若不想使用缓存，使用 docker build –no-cache

注意：Dockerfile 中每一个指令都会创建一个镜像层，上层是依赖于下层的。无论什么时候，只要某一层发生变化，其上面所有层的缓存都会失效。这时不会使用缓存，会重新创建。

#### debug Dockerfile

通过 Dockerfile 构建镜像的过程:

从 base 镜像运行一个容器。

执行一条指令，对容器做修改。

执行类似 docker commit 的操作，生成一个新的镜像层。

Docker 再基于刚刚提交的镜像运行一个新容器。

重复 2-4 步，直到 Dockerfile 中的所有指令执行完毕。

如果 Dockerfile 由于某种原因执行到某个指令失败了，我们也将能够得到前一个指令成功执行构建出的镜像，此时可以执行:

docker run -it '最后一个成功的镜像id' ; 然后可以手动执行后续操作来调试

#### RUN vs CMD vs ENTRYPOINT

Shell 和 Exec 格式：

两种方式来指定RUN CMD ENTRYPOINT要运行的命令。

Shell格式：<instruction> <command>

shell 格式底层会调用 /bin/sh -c <command>

ENV name Cloud Man

ENTRYPOINT ["/bin/echo", "Hello, $name"]

输出：Hello, Cloud Man

Exec格式：<instruction> ["executable", "param1", "param2", ...]

当指令执行时，会直接调用 <command>，不会被 shell 解析

ENV name Cloud Man

ENTRYPOINT ["/bin/echo", "Hello, $name"]

输出：Hello, $name 此时$name并没被替换

**CMD 和 ENTRYPOINT 推荐使用 Exec 格式，因为指令可读性更强，更容易理解。RUN 则两种格式都可以。**

* RUN 指令通常用于安装应用和软件包；在当前镜像顶部执行命令，创建新的镜像层
* CMD 指令允许用户指定容器的默认执行的命令。

**此命令会在容器启动且 docker run 没有指定其他命令时运行。**

**注意：**

*如果 docker run 指定了其他命令，CMD 指定的默认命令将被忽略*。

如果 Dockerfile 中有多个 CMD 指令，只有最后一个 CMD 有效。

CMD 有三种格式：

Exec 格式：CMD ["executable","param1","param2"]

这是 CMD 的推荐格式。

CMD ["param1","param2"] 与Exec 格式 的 ENTRYPOINT 指令配合使用，是为 ENTRYPOINT 设置默认的参数，此时 ENTRYPOINT 必须使用 Exec 格式。

例如：

Dockerfile: CMD echo "Hello world"

运行容器 docker run -it [image] 将输出：Hello world

若这样运行：docker run -it [image] /bin/bash，CMD 会被忽略掉

Shell 格式：CMD command param1 param2

* ENTRYPOINT

让容器以应用程序或者服务的形式运行。

Exec 格式：

ENTRYPOINT *不会被忽略，一定会被执行*，即使运行 docker run 时指定了其他命令。

ENTRYPOINT 中的参数始终会被使用，而 CMD 的额外参数可以在容器启动时动态替换掉

例如： ENTRYPOINT ["/bin/echo", "Hello"]

CMD ["world"]

当docker run -it [image] 启动时，输出Hello world；

当docker run -it [image] CloudMan启动时，输出Hello CloudMan；

Shell格式：ENTRYPOINT 的 Shell 格式会忽略任何 CMD 或 docker run 提供的参数。

***最佳实践***

使用 RUN 指令安装应用和软件包，构建镜像。

如果 Docker 镜像的用途是运行应用程序或服务，比如运行一个 MySQL，应该优先使用 Exec 格式的 ENTRYPOINT 指令。CMD 可为 ENTRYPOINT 提供额外的默认参数，同时可利用 docker run 命令行替换默认参数。

如果想为容器设置默认的启动命令，可使用 CMD 指令。用户可在 docker run 命令行中替换此默认命令。

#### 镜像命名

实际上一个特定镜像的名字由两部分组成：repository 和 tag



若执行docker build没有指定tag，会默认使用latest，相当于：

docker build -t ubuntu-with-vi:latest

tag一般用于描述版本信息：

每个 repository 可以有多个 tag，而多个 tag 可能对应的是同一个镜像。

假设我们现在发布了一个镜像 myimage，版本为 v1.9.1。那么我们可以给镜像打上四个 tag：1.9.1、1.9、1 和 latest。



docker tag myimage-v1.9.1 myimage:1

docker tag myimage-v1.9.1 myimage:1.9

docker tag myimage-v1.9.1 myimage:1.9.1

docker tag myimage-v1.9.1 myimage:latest

过了一段时间，我们发布了 v1.9.2。这时可以打上 1.9.2 的 tag，并将 1.9、1 和 latest 从 v1.9.1 移到 v1.9.2。



之后，v2.0.0 发布了。这时可以打上 2.0.0、2.0 和 2 的 tag，并将 latest 移到 v2.0.0。



最终：

myimage:1 始终指向 1 这个分支中最新的镜像。

myimage:1.9 始终指向 1.9.x 中最新的镜像。

myimage:latest 始终指向所有版本中最新的镜像。

如果想使用特定版本，可以选择 myimage:1.9.1、myimage:1.9.2 或 myimage:2.0.0。