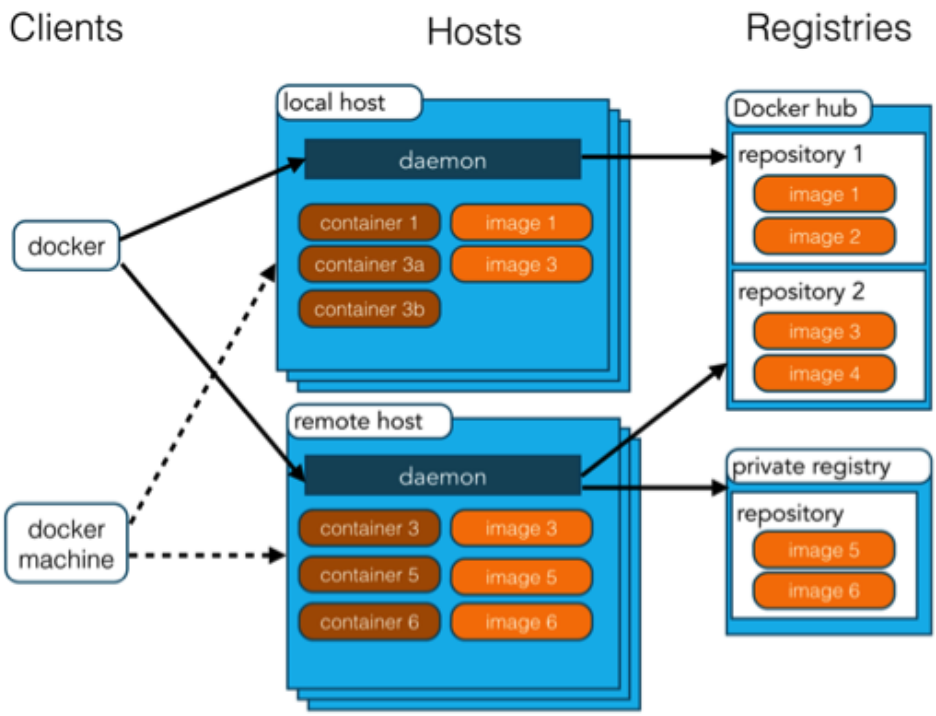
架构



- **Docker**三个组件：
  - Docker Client：以cli的形式让用户可以与docker damon通信
  - Docker Daemon: 运行于主机上，处理client的请求，从本地镜像中拉起并运行容器
  - Docker Registery: 镜像仓库
- **Docker**三个基本要素：
  - Docker Images: 镜像，用来创建docker容器的模板
  - Docker Containers: 容器，独立运行的一个或一组应用，是镜像运行的实体
  - DockerFile： 文件指令集，用来说明如何创建镜像

命令

- 镜像：
  - 下载镜像到本地：

    ```
    docker pull [镜像名]:[tag=latest]
    ```
  - 列举本地已经下载的镜像：

    ```
    docker images
    ```
  - 在**docker registery**中查找镜像：

    ```
    docker search [镜像名]
    ```
  - 删除本地已经下载的镜像：

    ```
    docker rmi [镜像名]
    ```

- 更新镜像：

  `docker run -it [镜像名]:[tag=latest] /bin/bash -> apr-get update`
- 根据**Dockerfile**构建镜像：

  `docker build -t [镜像名]:[tag] [Dockerfile所在的绝对路径]`

- 容器
  - 创建容器：
    - *创建容器并启动：*

      `docker run [options] [镜像名] [command]`
      - options:

        -d: 后台运行容器

        -p [宿主端口]:[容器端口]: 指定端口映射

        -it: 给容器分配一个输入终端

        -v [宿主目录]:[容器目录]: 指定目录映射
      - command:

        创建容器后要运行的命令
    - *创建容器但不启动：*

      `docker create [options] [镜像名] [command]`

      options用法同docker run
  - 启停容器：
    - *启动容器：*

      `docker start [容器名/容器id]`
    - *直接停止容器：*

      `docker kill [容器名/容器id] (直接kill容器)`
    - *优雅停止容器：*

      `docker stop [容器名/容器id] (先发送SIGTERM信号，让容器自己停止，如果规定时间内不停止，再kill容器)`
    - *重启容器：*

      `docker restart [容器名/容器id]`
    - *暂停容器：*

      `docker pause [容器名/容器id]`
    - *恢复已暂停容器：*

      `docker unpause [容器名/容器id]`
  - 删除容器：

    `docker rm [options] [容器名/容器id]`
    - options：

      -f: 即使容器正在运行，也强制删除容器

      -v：移除容器时顺带删除跟容器管理的宿主机目录
  - 操作容器：
    - *查看容器列表*：

      `docker ps [options]`
      - options:

        -a: 显示所有容器，包括未运行的

- *获取容器日志*：

  `docker logs [options] [容器名/容器id]`

  - options:

    -f: 跟踪日志输出

    -t：显示时间戳

    --tail=[N]: 列出最新的N条日志
- *进入容器执行命令*：

  1. 退出后不停止容器：

     `docker exec [options] [容器名/容器id] [command]`

     - options：

       -it: 给容器分配一个伪终端

       -d：在后台运行命令
  2. 退出后停止容器：

     `docker attach [options] [容器名/容器id] [command]`

     - options：

       -it: 给容器分配一个伪终端

       -d：在后台运行命令
- *宿主机与容器之间的数据拷贝：*
  - 宿主机拷贝到容器： `docker cp [src_path] [容器名/容器id]:[destination_path]`
  - 容器拷贝到宿主机： `docker cp [容器名/容器id]:[destination_path] [src_path]`
- *检查容器里文件结构的变更：*

  `docker diff [容器名/容器id]`

Dockerfile

根据Dockerfile构建镜像的指令：

`docker build [options] [context_path | url]`

- options:

  -f [Dockerfile路径]：指定Dockerfile路径

  -t [镜像名:tag] 设置镜像名
- context_path: 宿主机中的某个目录路径

  url：指定git仓库，docker会自动clone，解压该仓库，并以该仓库根目录作为上下文

Dockerfile指令：

- `FROM [镜像名]:`

  基于某个基础镜像构建一个新镜像

  例：FROM nginx：基于Nginx的镜像构建一个新镜像；FROM scratch：基于空镜像构建一个新镜像
- `RUN [shell命令]:`

  在构建镜像时通过执行命令，在容器中添加一层, RUN的命令在docker build的时候被执行

  例：RUN mkdir temp (等同于RUN ["mkdir", "temp"]): 在容器中创建一个名为temp的目录, 每添加一个RUN指令，都是在容器中添加一层，因此不要滥用RUN
- `CMD [shell命令]:`

  在启动容器时默认要执行的命令，CMD的命令在docker run的时候被执行，CMD 指令指定的程序可被 docker run 命令行参数中指定要运行的程序所覆盖

- `ENTRYPOINT [shell命令]:`
  在容器启动时指定要执行的命令，ENTRYPOINT的命令在docker run的时候被执行，ENTRYPOINT指令指定的程序可被 docker run 命令行参数中指定要运行的程序所追加

> Example:
>
> ```
> // 构建启动的Nginx镜像
> FROM nginx
> ENTRYPOINT ["nginx", "-c"] # 定参
> CMD ["/etc/nginx/nginx.conf"] # 变参
> ```

- `COPY [--chown=<user>:<group>] [上下文路径1] [上下文路径2]... [容器内目标路径]：`
  将上下文路径下的文件或目录复制到容器内的目标路径下
- `ADD [--chown=<user>:<group>] [上下文路径1] [上下文路径2]... [容器内目标路径]`
  将上下文路径下的文件或目录复制到容器内的目标路径下（如果要复制的文件是tar压缩包，会自动解压）
- `ENV <key>=<value>`
  设置环境变量，供容器中运行的进程使用
- `ARG <key>=<value>`
  设置构建参数，供Dockerfile中使用
- `VOLUME ["<容器内路径1>", "<容器内路径2>" ...]`
  将容器内目录挂载到匿名卷
  EXPOSE <端口1> <端口2> ...
  声明要暴露的端口， docker run -P时会随机映射EXPOSE的端口
- `WORKDIR <工作目录路径>`
  指定后续指令的工作目录，相当于Dockerfile中的cd指令
- `USER <用户名>:<用户组>`
  指定后续命令的用户和用户组，相当于Dockerfile中的su命令

docker-compose

docker-compose.yml

```yaml
version: '3' # docker compose 版本
services:
  app:
    container_name: app # 指定容器名
    build: ./app1 # 指定dockerfile的context
    command: node ./start.js # 用于覆盖dockerfile中默认的CMD
    ports: # 配置端口映射，相当于docker run -p
      - "80:80"
    volumns: # 配置目录映射，相当于docker run -v
      - "~/.ssh:/home/app/.ssh"
    depends_on: # 声明依赖的容器，docker-compose up时会按照依赖顺序依次启动容器
      - redis
      - db
    link: # 声明容器之间的连接，声明后容器app可以以db为hostName连接容器db
      - "db:db"
  redis:
    container_name: redis
    image: redis
    expose:
      - "6379"
  db:
    ...
```