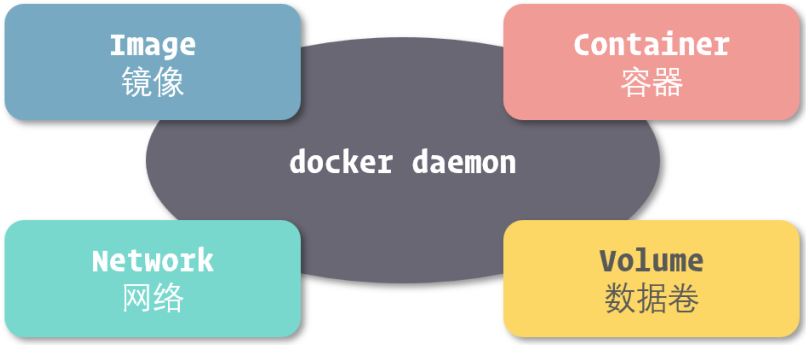
1.Docker笔记 - 基础入门

四个对象 (Object)

镜像 (Image)、容器 (Container)、网络 (Network)、数据卷 (Volume)

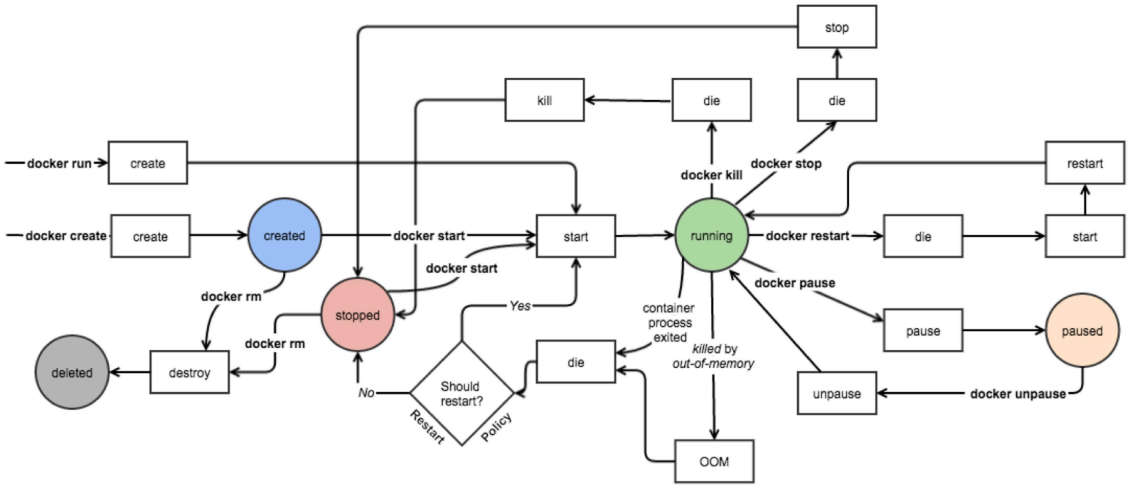
Docker Engine

docker daemon 和 docker CLI



Docker 镜像

docker images 查看全部镜像
docker pull 拉去镜像保存
docker search 搜索镜像
docker inspect 查看详细镜像
docker rmi 删除镜像

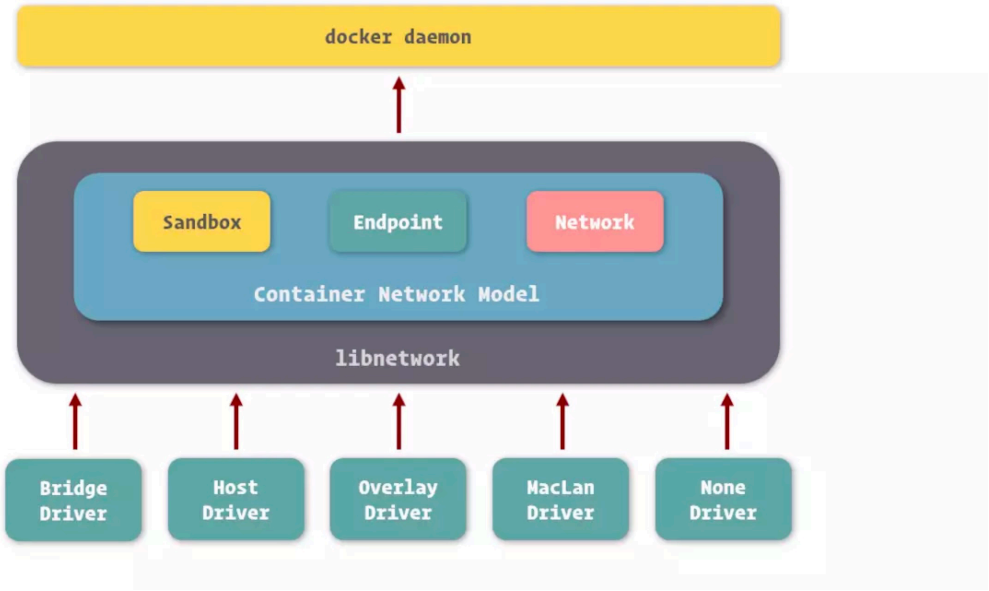


- **Created**：容器已经被创建，容器所需的相关资源已经准备就绪，但容器中的程序还未处于运行状态。
- **Running**：容器正在运行，也就是容器中的应用正在运行。
- **Paused**：容器已暂停，表示容器中的所有程序都处于暂停（不是停止）状态。
- **Stopped**：容器处于停止状态，占用的资源和沙盒环境都依然存在，只是容器中的应用程序均已停止。
- **Deleted**：容器已删除，相关占用的资源及存储在 Docker 中的管理信息也都已释放和移除。

Docker 容器

docker create 创建容器 --name <别名>
docker start 启动容器 -d --detach 后台运行 —rm 结束运行后 删除该容器
docker run 结合了create start合体
docker ps 查看运行中的容器 -a —all 查看所有状态的容器
docker stop 停止容器
docker rm 删除容器 -f —force 强制停止删除 -v 删除容器关联的数据卷
docker exec 进入容器 后面输入Linux命令就执行 比如 ls /
进入伪终端 就加上-it(--interactive输入流 --tty伪终端) 输入输出 执行 sh 或 bash
docker attach 把后台运行转为前台 Ctrl+C 退出

Docker 网络



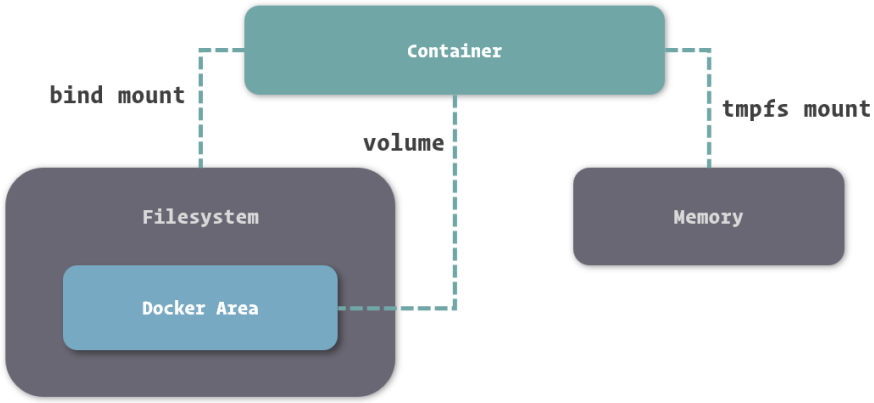
网络驱动 Bridge Driver、Host Driver、Overlay Driver、MacLan Driver、None Driver

- Bridge Driver
- Host Driver
- Overlay Driver
- MacLan Driver
- None Driver

--link 容器互联 别名会自动解析ip --link <name>:<alias>
--expose 暴露容器互联端口
--network 创建容器时 设置加入的网络
-p --publish 暴露外部端口 -p <ip>:<host-port>:<container-port>

docker network 网络相关
docker network create 创建网络 -d <网络驱动类型> <别名>
docker network ls /list 查看存在的网络

Docker 存储



挂载方式Bind Mount、Volume、Tmpfs Mount

- Bind Mount 能够直接将宿主操作系统中的目录和文件挂载到容器内的文件系统中，通过指定容器外的路径和容器内的路径，就可以形成挂载映射关系，在容器内外对文件的读写，都是相互可见的。
- Volume 也是从宿主操作系统中挂载目录到容器内，只不过这个挂载的目录由 Docker 进行管理，我们只需要指定容器内的目录，不需要关心具体挂载到了宿主操作系统中的哪里。
- Tmpfs Mount 支持挂载系统内存中的一部分到容器的文件系统里，不过由于内存和容器的特征，它的存储并不是持久的，其中的内容会随着容器的停止而消失。

-v --volume 挂载目录Bind Mount映射 -v <host-path>:<container-path> host-path 和 container-path 分别代表宿主操作系统中的目录和容器中的目录。仅使用绝对路径 :ro 只读挂载
--tmpfs 挂载临时文件目录 Tmpfs Mount 因为是存储在内存 可以挂载缓存文件夹
-v --volume 使用数据卷Volume -v <container-path> 别名 -v <name>:<container-path>
--mount 高级自定义挂载

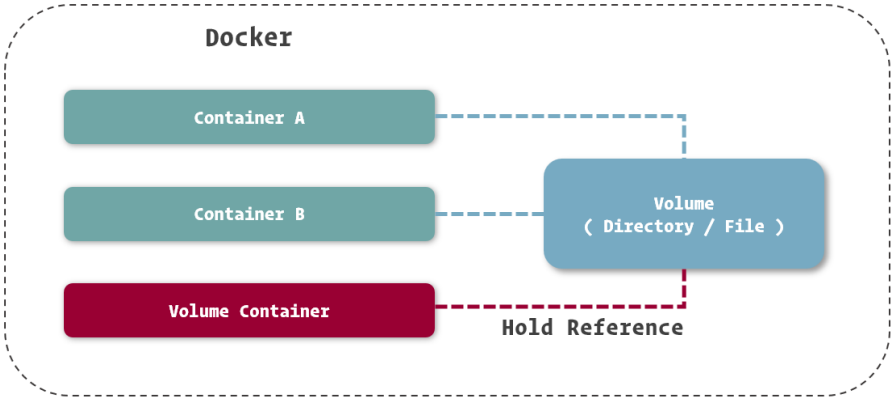
由于 -v 选项既承载了 Bind Mount 的定义，又参与了 Volume 的定义，所以其传参方式需要特别留意。-v 在定义绑定挂载时必须使用绝对路径，其目的主要是为了避免与数据卷挂载中命名这种形式的冲突。

由于数据卷的命名在 Docker 中是唯一的，所以我们很容易通过数据卷的名称确定数据卷，这就让我们很方便的让多个容器挂载同一个数据卷了。

docker volume 数据卷相关

docker volume create 创建数据卷
docker volume ls 查看已存在的数据卷
docker volume rm 删除数据卷
docker volume prune 删除没有被容器引用的数据卷
docker rm -v时 删除容器关联的数据卷

数据卷容器



比较抽象 先创建一个容器 使用数据卷 然后其他容器引用该数据卷 路径直接跟着数据卷容器走
创建
docker create --name appdata -v /webapp/storage ubuntu
在使用数据卷容器时，我们不建议再定义数据卷的名称，因为我们可以通过对数据卷容器的引用来完成数据卷的引用。而不设置数据卷的名称，也避免了在同一 Docker 中数据卷重名的尴尬。
Docker 的 Network 是容器间的网络桥梁，如果做类比，数据卷容器就可以算是容器间的文件系统桥梁。我们可以像加入网络一样引用数据卷容器，只需要在创建新容器时使用专门的 --volumes-from 选项即可。
docker run -d --name webapp --volumes-from appdata webapp:latest
引用数据卷容器时，不需要再定义数据卷挂载到容器中的位置，Docker 会以数据卷容器中的挂载定义将数据卷挂载到引用的容器中。

备份/迁移数据卷

备份：挂载 打包
docker run --rm --volumes-from appdata -v /backup:/backup ubuntu tar cvf /backup/backup.tar /webapp/storage

迁移：解压 复制
docker run --rm --volumes-from appdata -v /backup:/backup ubuntu tar xvf /backup/backup.tar -C /webapp/storage --strip

docker cp 容器与宿主机的文件系统间拷贝文件和目录