## 深入浅出Docker(命令有些老,原理很清晰)

DCA (Docker Certified Associate)Docker认证工程师

### docker由来

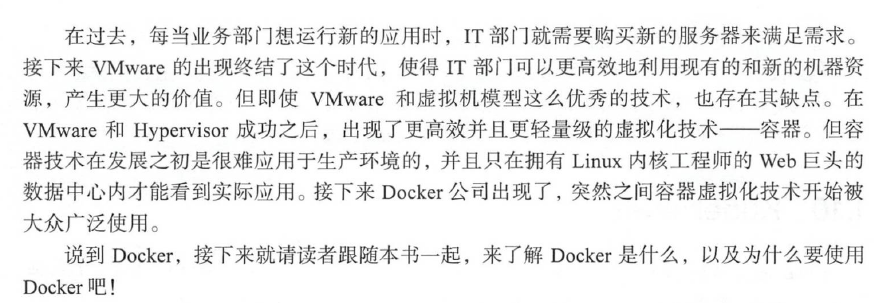
Linux容器: 推动其发展的是

内核命名空间(Kernel Namespace) <https://blog.csdn.net/qq_38410730/article/details/81105132>

内核也是程序,也需要分配内存给他,这块内存就是内核空间

控制组( Control Group)

联合文件系统( Union File System )



运行中的容器共享宿主机的内核,这一点很重要

Docker 引擎是用于运行和编排容器的基础设施工具,类似Vmware的ESXi

Docker开源项目是Moby,该项目包含Docker引擎和其他的子项目和工具,多数项目和工具给予golang编写

### Docker生态

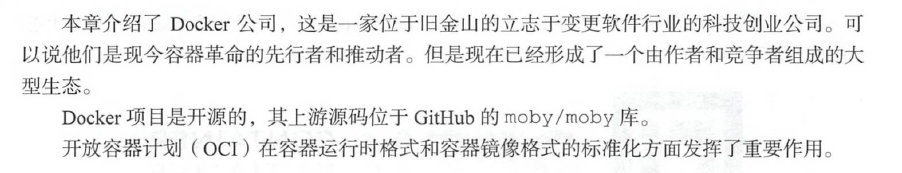
理念是: “含电池,但可拆卸”

Docker内置的很多组件都可以替换成第三方的组件,比如: 网络技术栈

### 开放容器计划 OCI

The Open Container Initiactive(倡议; 新方案; 主动性; 积极性; 自发性; 掌握有利条件的能力(或机会); 主动权;)

对容器基础架构中的基础组件进行标准化的委员会



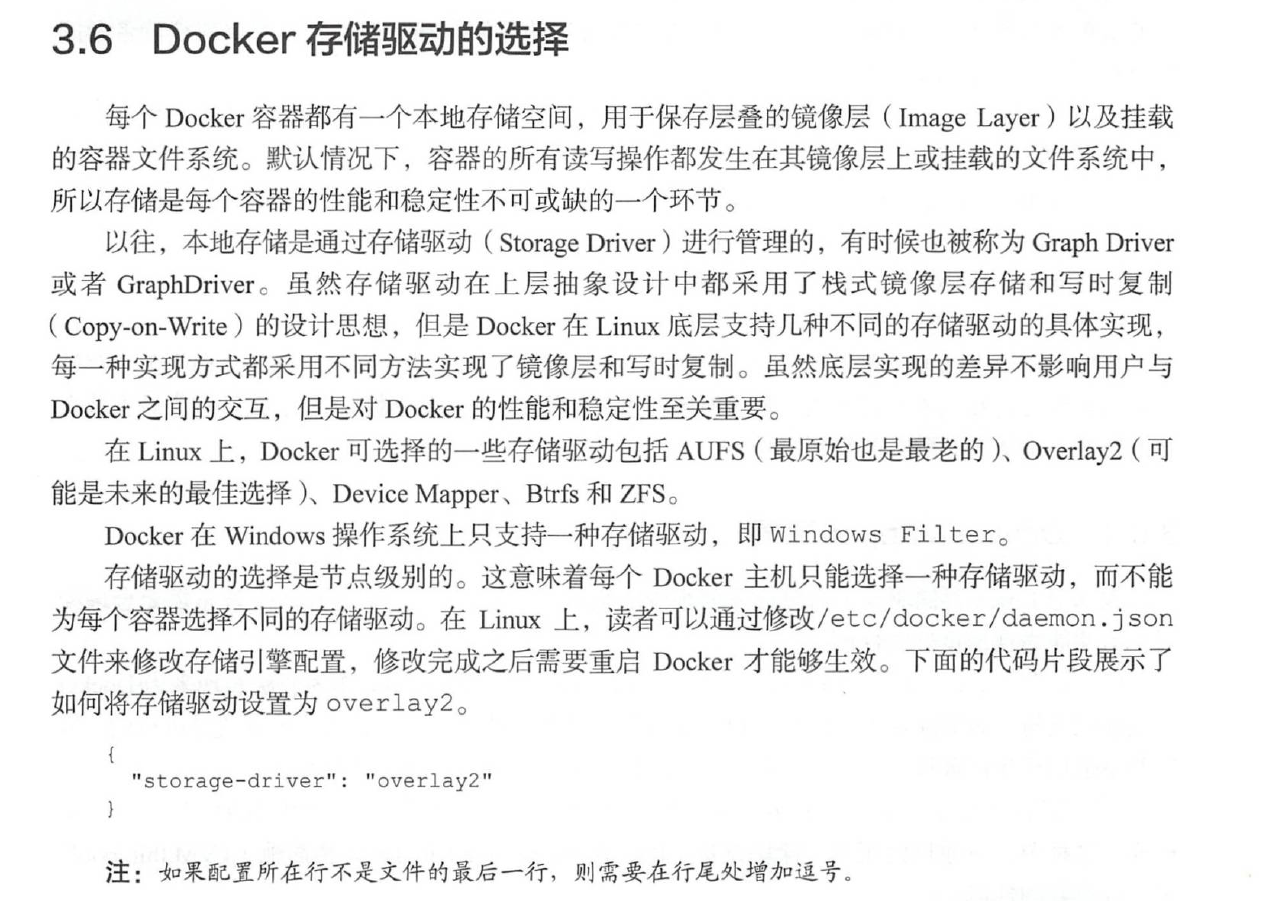
### Docker 安装

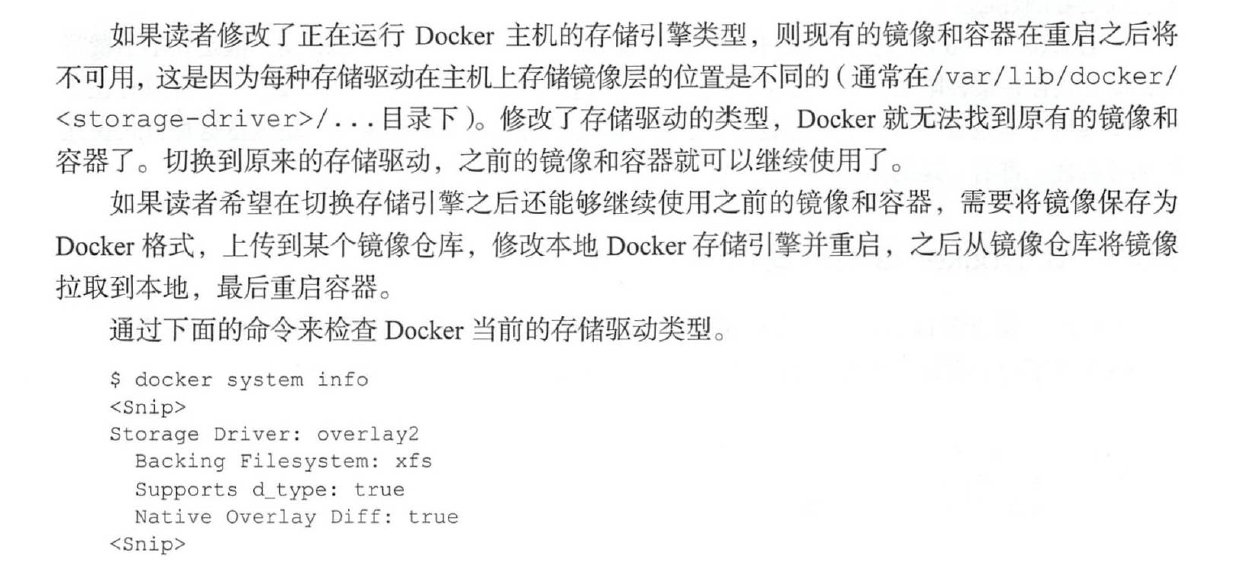
Windows上面一种是将docker守护进程运行在Hyper-V虚拟机的一个轻量级Linux中,这种情况docker –version 输出的OS/Arch 任然是linux/amd64.此时只能跑Linux容器!!

另一种是切换到Windows容器,此时docker守护进程运行在Windows内核上,就只能运行Windows的容器

Docker引擎的升级: 先卸载旧版本; 重新安装

Docker存储驱动

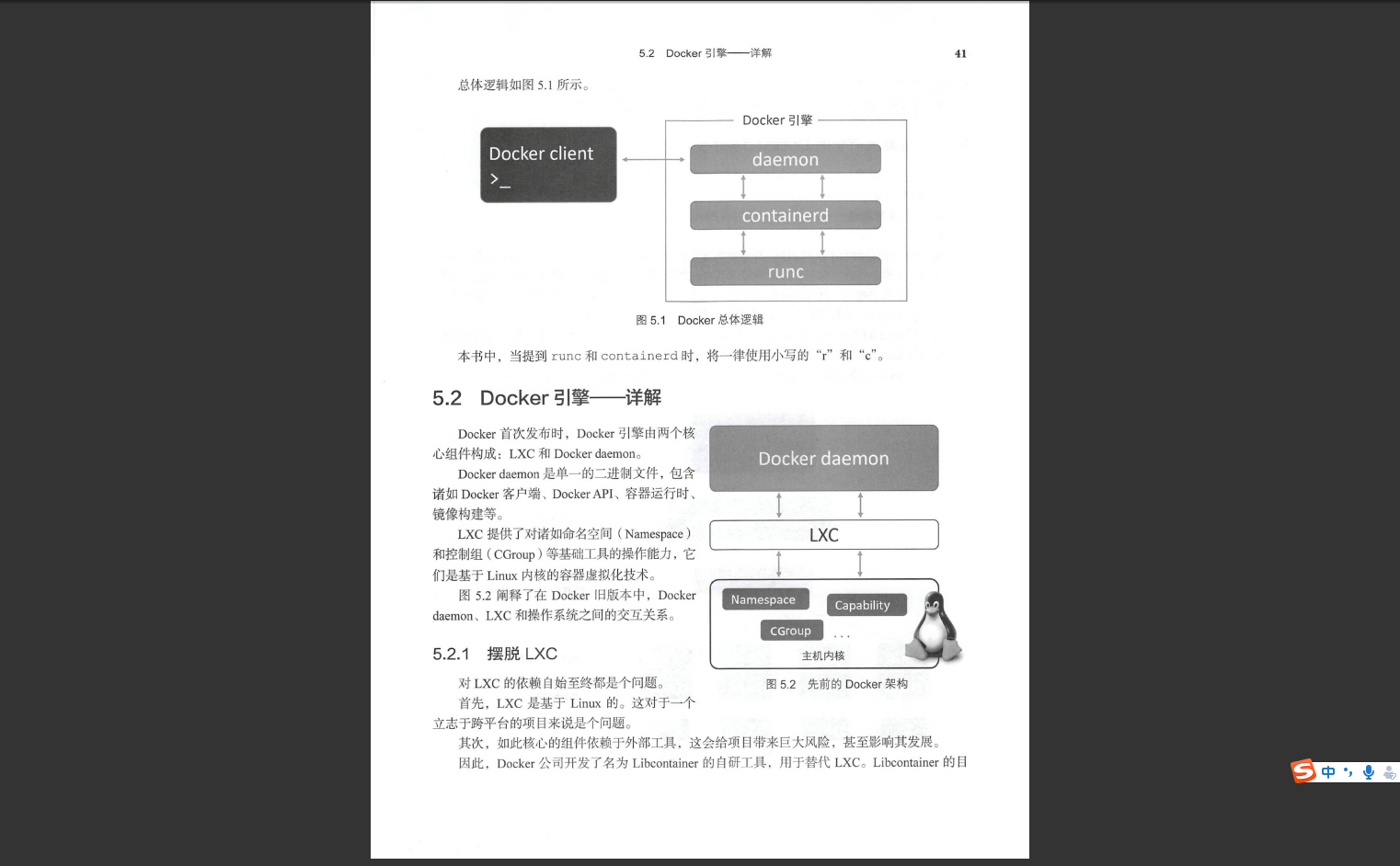




可以从图片中看到, 我们的镜像层在主机上面的存储目录结构就是/var/lib/docker/<storage-driver>/…目录

### Docker引擎

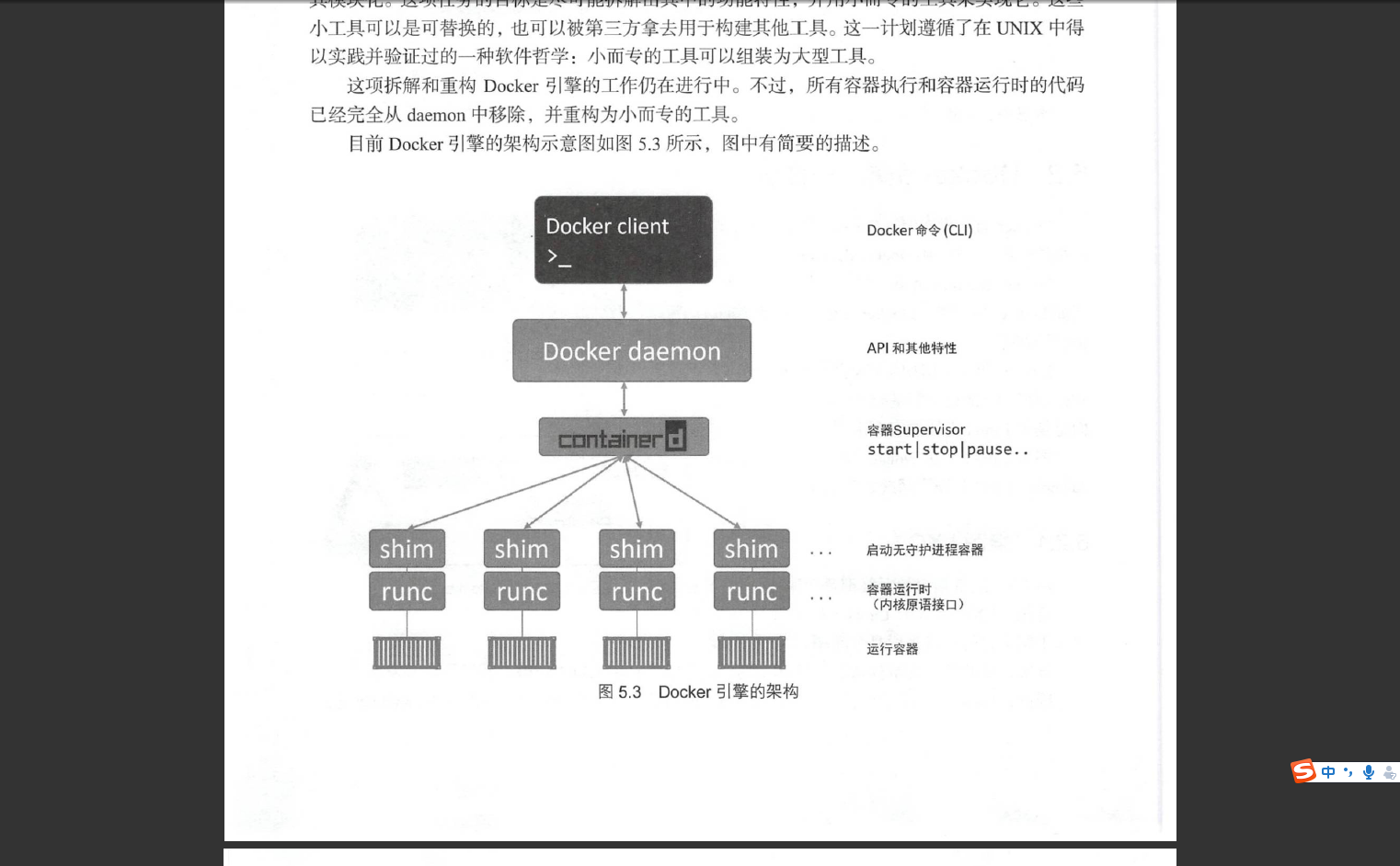
用来运行和管理容器的核心软件



如此严重的依赖LXC(linux container) 对于软件跨平台性是严重的问题!!

Docker公司开发开发了libcontainer,替代了LXC 基于不同内核为docker上层提供必要的容器交互能力,后面docker的默认执行驱动也就是libcontainer

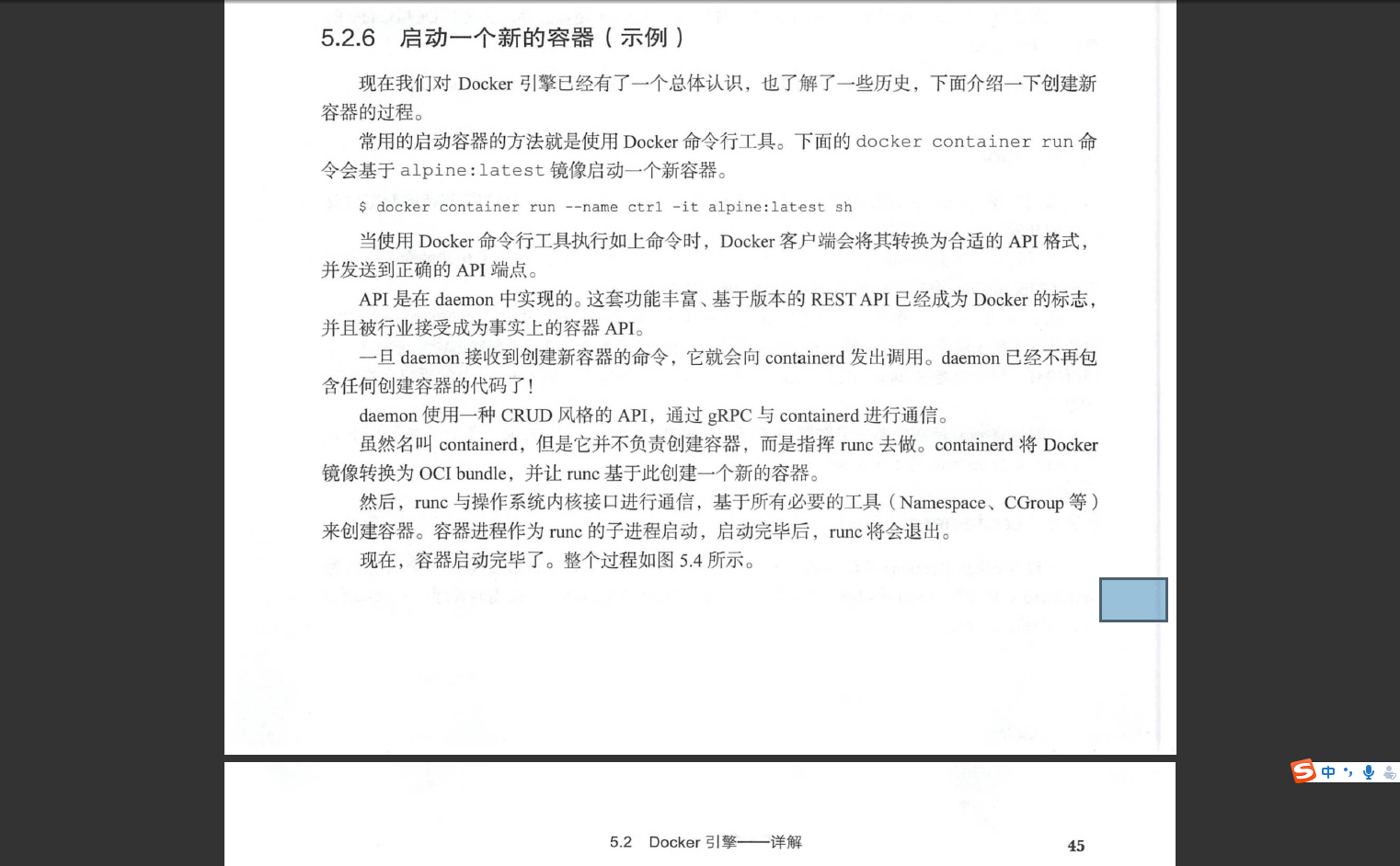
后面docker daemon 也向模块化靠拢 linux软件哲学(小而专的工具可以组装成大公工具)

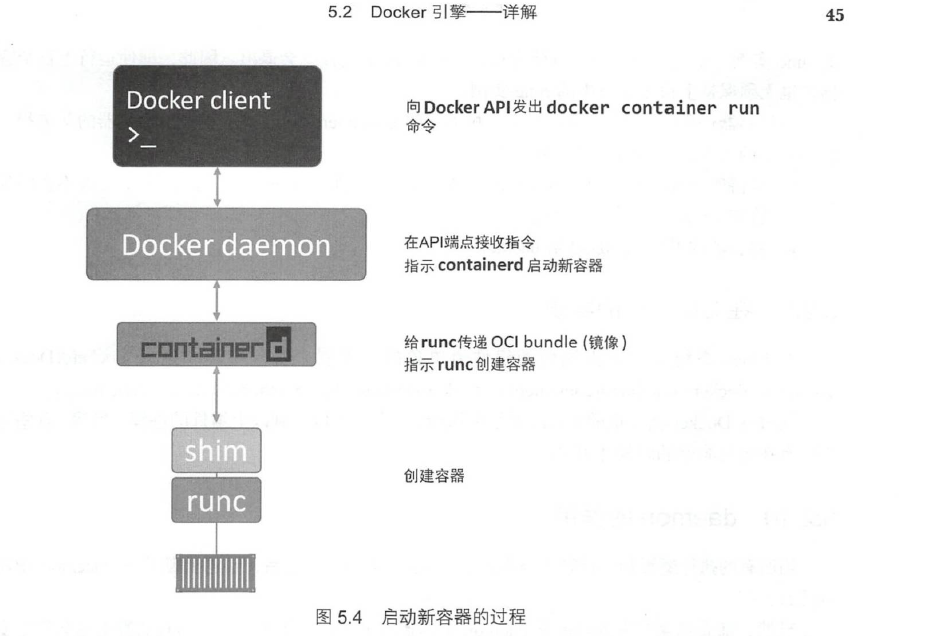


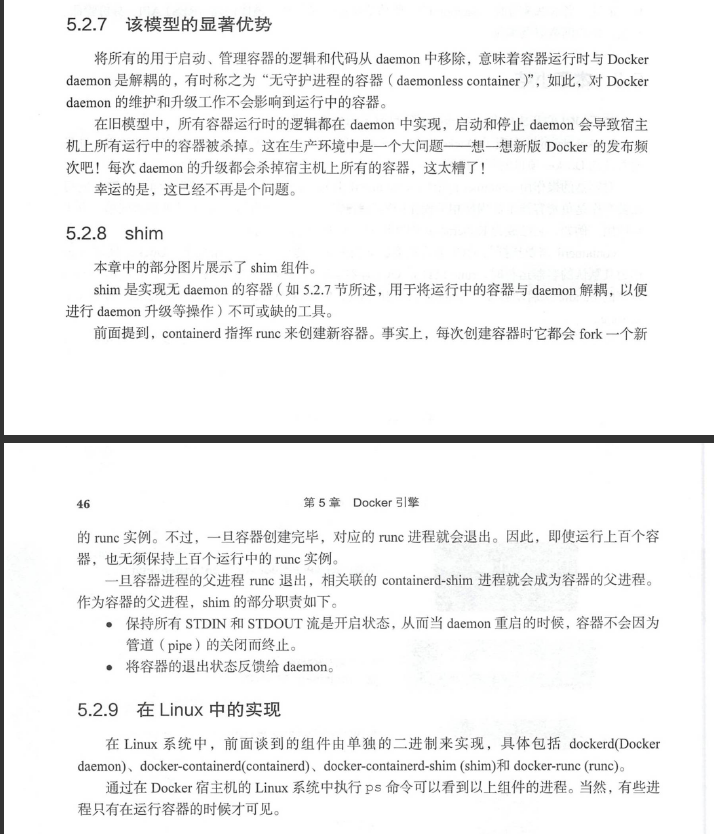
Runc是容器运行时规范的参考实现,实质是轻量级,针对libcontainer进行了包装的命令行交互工具(二进制文件),只有一个作用,创建容器

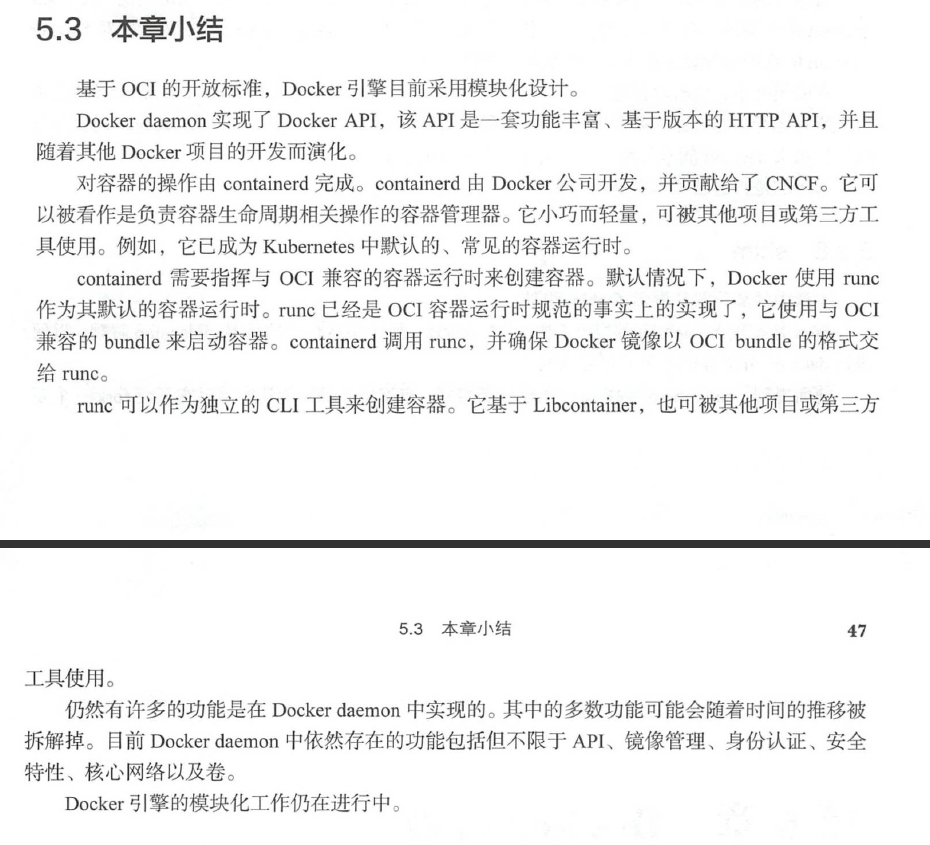
ContainerD (container-dee)容器的执行逻辑放在这里面处理,任务就是容器的生命周期管理,后面有添加了镜像管理,但是请记住都是模块化的,可选的

API实在daemon中实现的,daemon接收到创建容器的命令,就调用containerd,daemon不包含任何的创建容器的带代码; containerd并不自己创建容器,而是传给runc

调用过程:







### Docker镜像

为什么那么小??

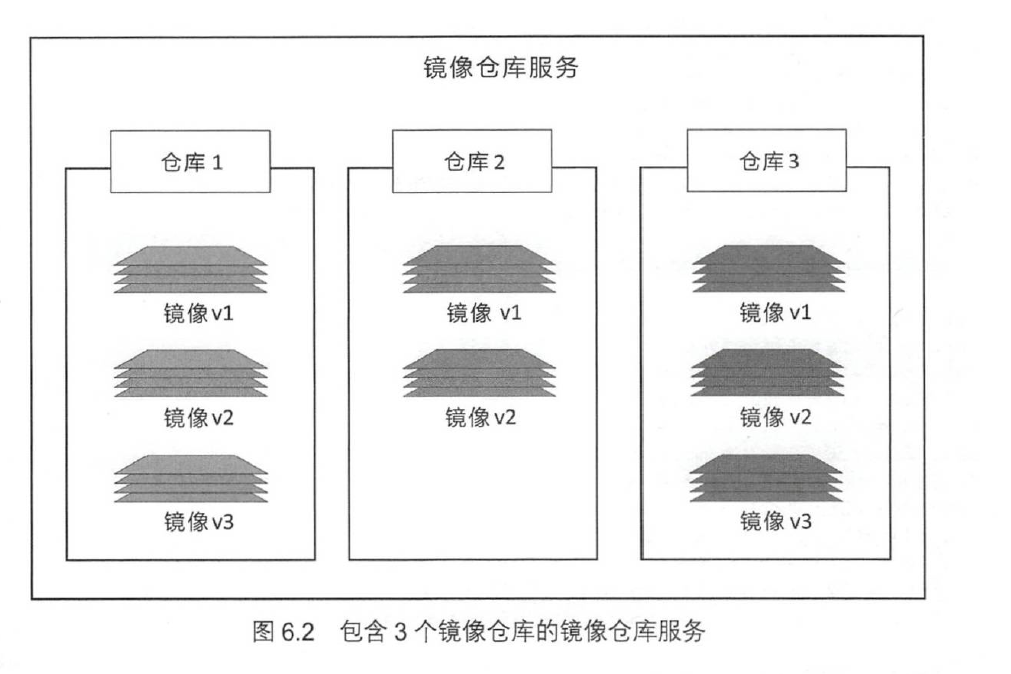
非官方的命名方式是： 仓库用户名/应用：tag

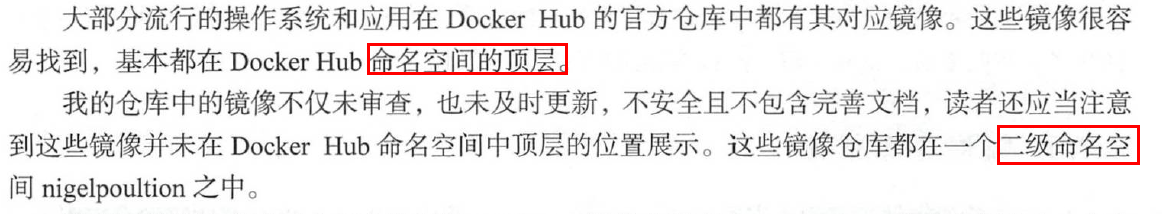


镜像存在哪??

一般是在/var/lib/docker/<storage-driver>

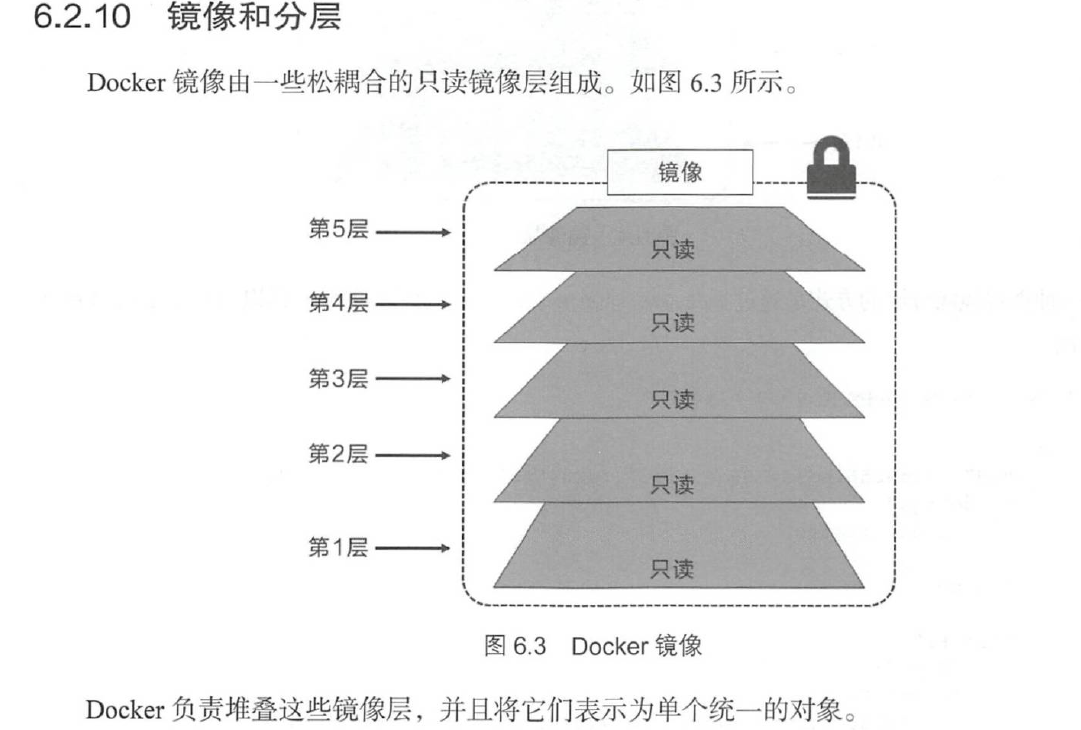
镜像仓库服务





对于同一个镜像,我们可不可以指定多个标签啊?? 可以 比如latest标签可以指向之前的版本,这样拉取下来的就不是最新版本了

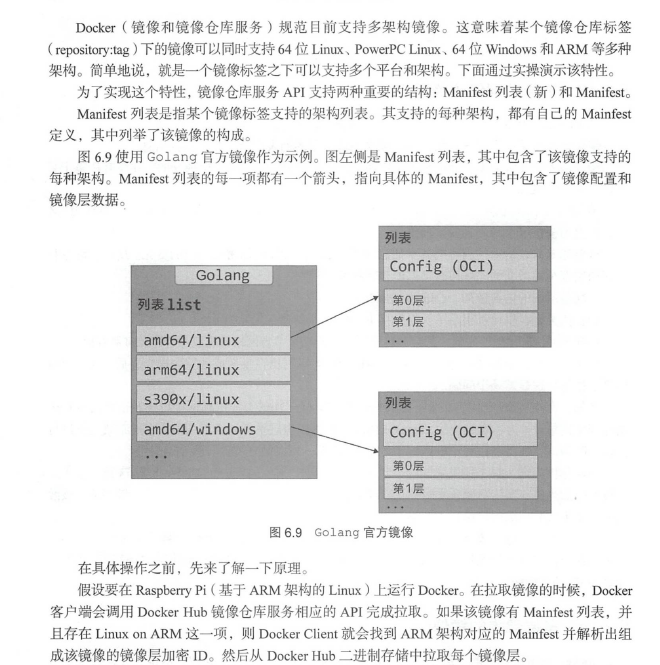
悬系镜像(dangling) 就是<none>标签,是因为构建了一个新镜像,同时命名了之前存在的镜像标签,导致之前的镜像变成dangling

使用docker inspect 查看镜像或者容器信息,layers代表镜像层

根据摘要信息拉拉取准确镜像

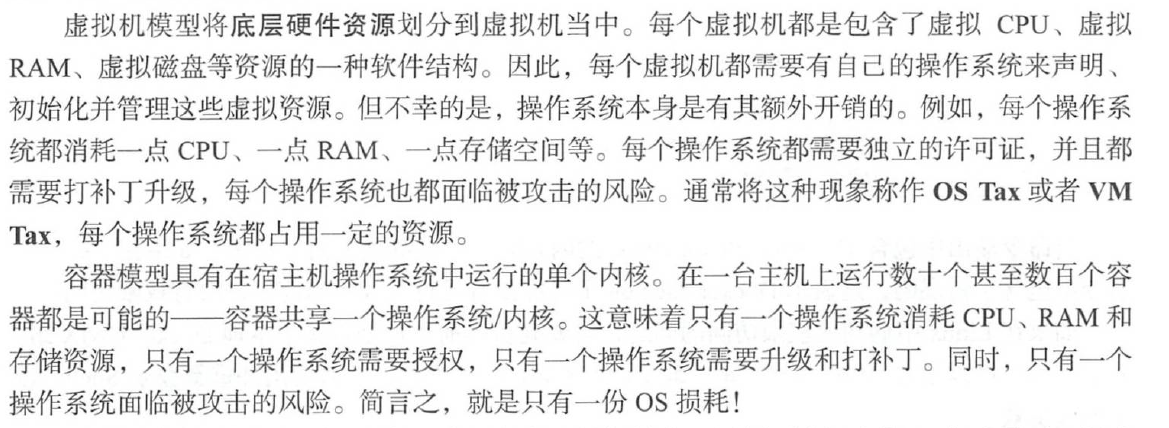
Docker images mysql –disgests 查看镜像摘要sha256值,通过那个值拉取镜像docker pull mysql@sha256....

多架构镜像: 目前获取的容器镜像已经支持多种架构

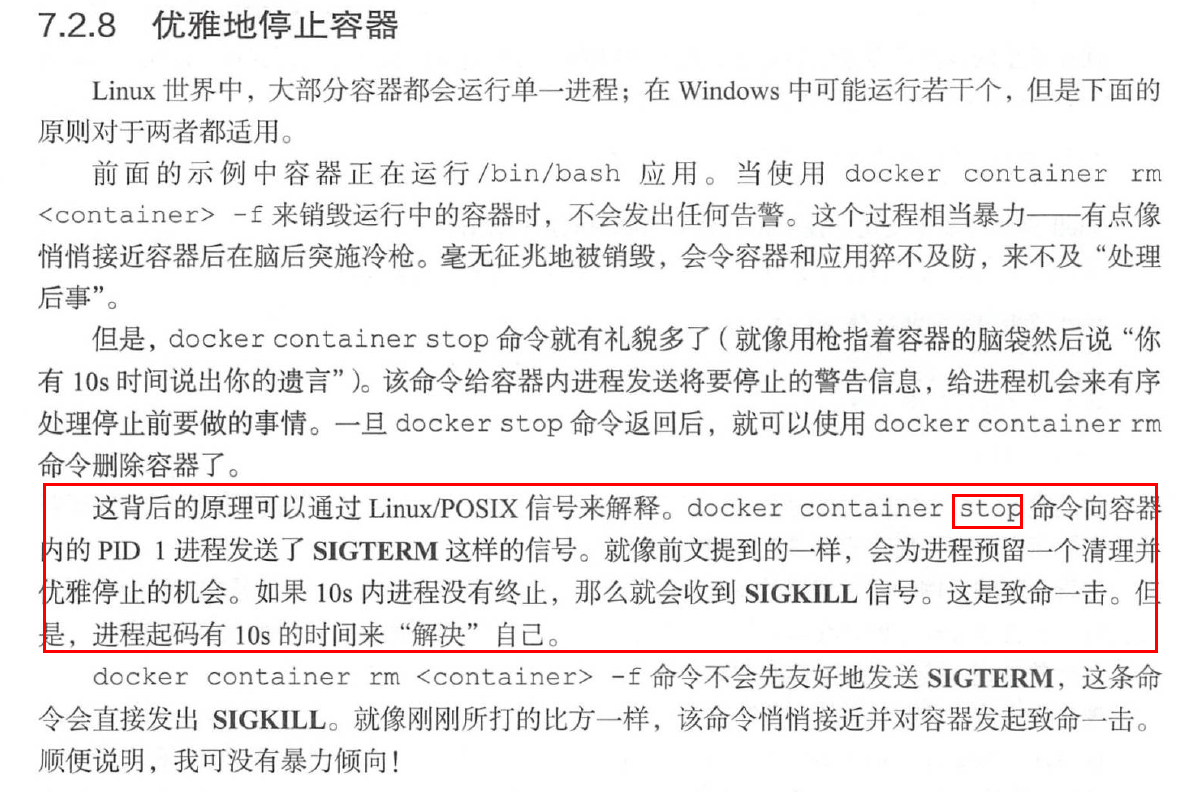


### Docker容器

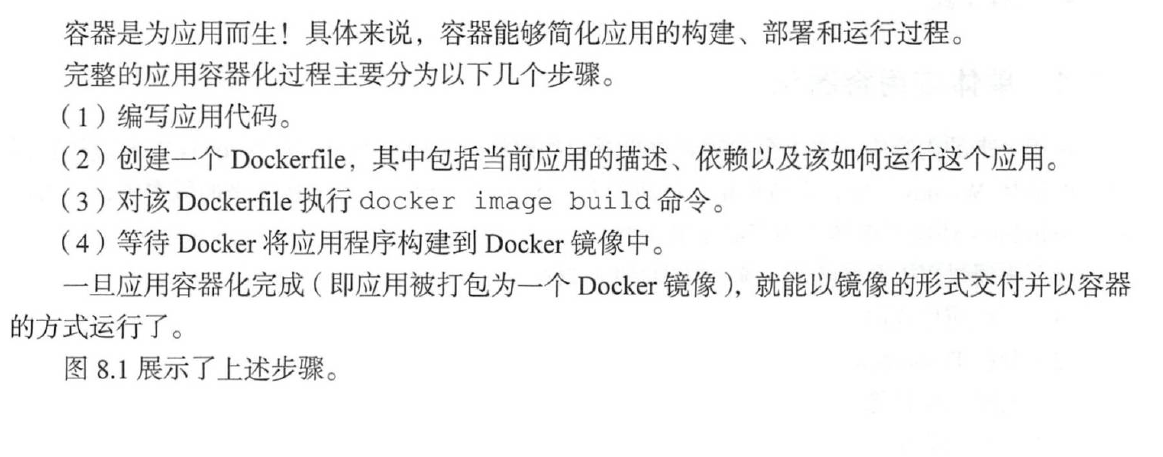
着重对比虚拟机和容器,hypervisor是硬件层面的虚拟,而容器是操作系统的虚拟



停止容器



### 应用容器化(重点)



DockerFile 主要包含两个功能:

1. 对当前应用进行描述
2. 知道Docker 完成应用的容器化

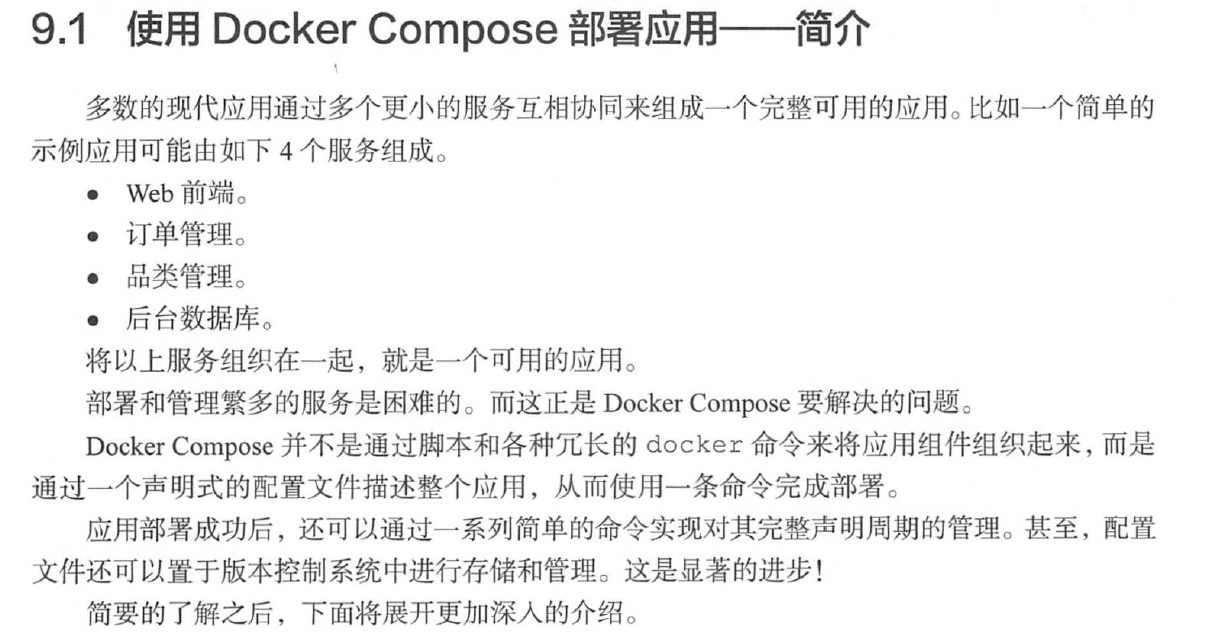
我们需要重视DockerFile 就像重视我们源代码一样,同时需要将DockerFile纳入版本控制中,

镜像层的产生: 是对上个镜像层有文件加入的操作,对于一些设置作者 工作目录 暴露端口 都不会产生新的镜像层,而是保存在镜像的元数据中

Docker compose (多容器部署): 使用声明式文件部署多个容器

日常中,我们可能一个应用是由多个容器组成的(比如web + 数据库) 怎么组织起来部署呢??

Docker compose并不是通过脚本和各种冗长的docker命令来完成,而是通过声明式的配置文件描述整个应用



Docker compose原身是fig ,把fig收购过来的,解析yaml文件通过docker api进行应用的部署和管理,yaml文件是JSON的一个子集 docker-conpose.yml文件定义compose怎么部署应用（就是多个services怎么去部署）,

启动部署命令: docker-compose up (在当前目录下寻找docker-compose.yml或者docker-compose.yaml文件,-f指定其他文件名) -d 参数就是后台运行了

做成的容器名称 是项目名称+资源名称(service key中定义名字)

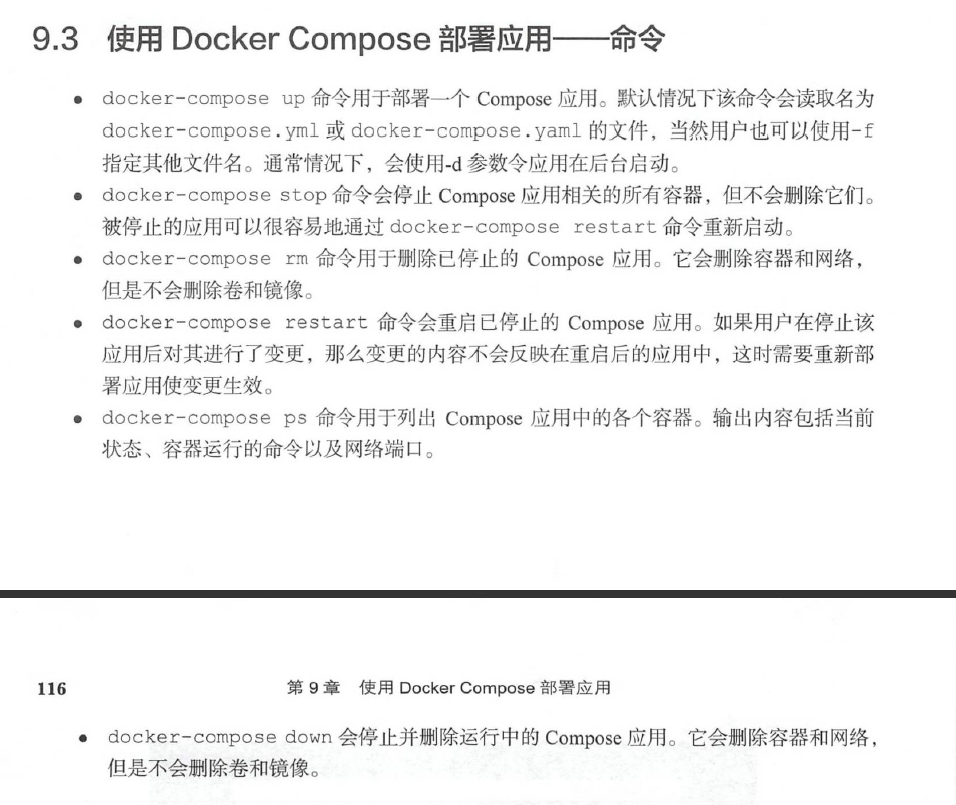
Docker-compose top 查看各个服务(容器)中的运行的进程

Docker-compose stop 停止掉容器

Docker-compose rm 删除容器

Down 停止和关闭应用 容器也会被删掉

Ps

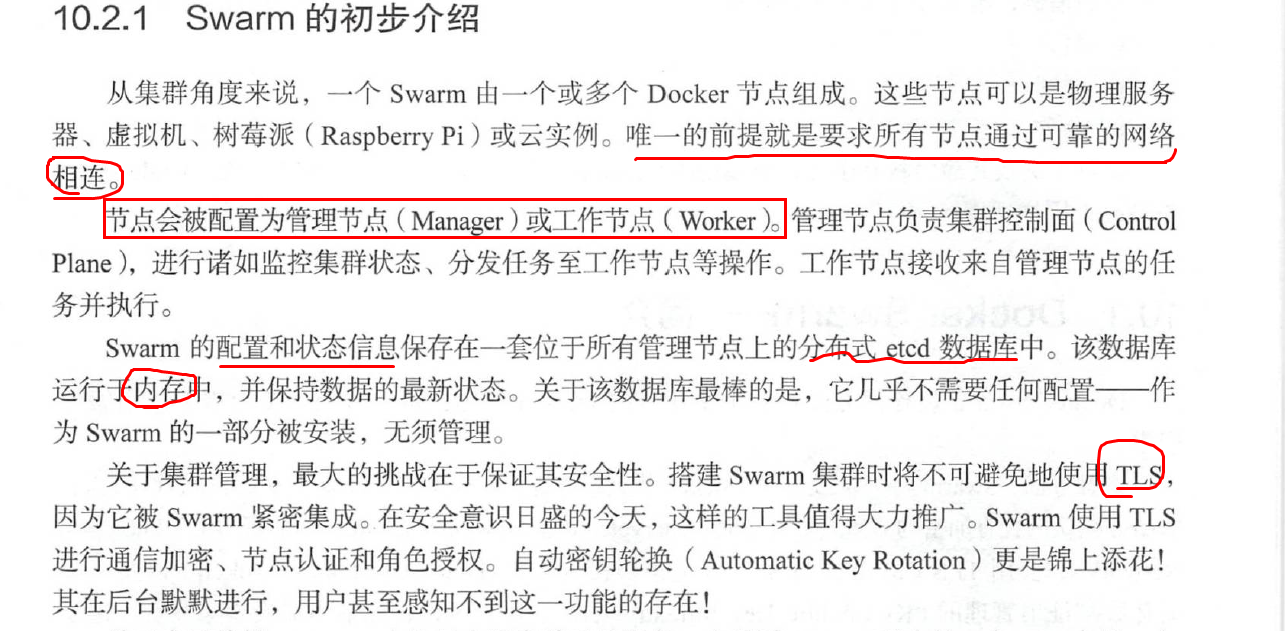


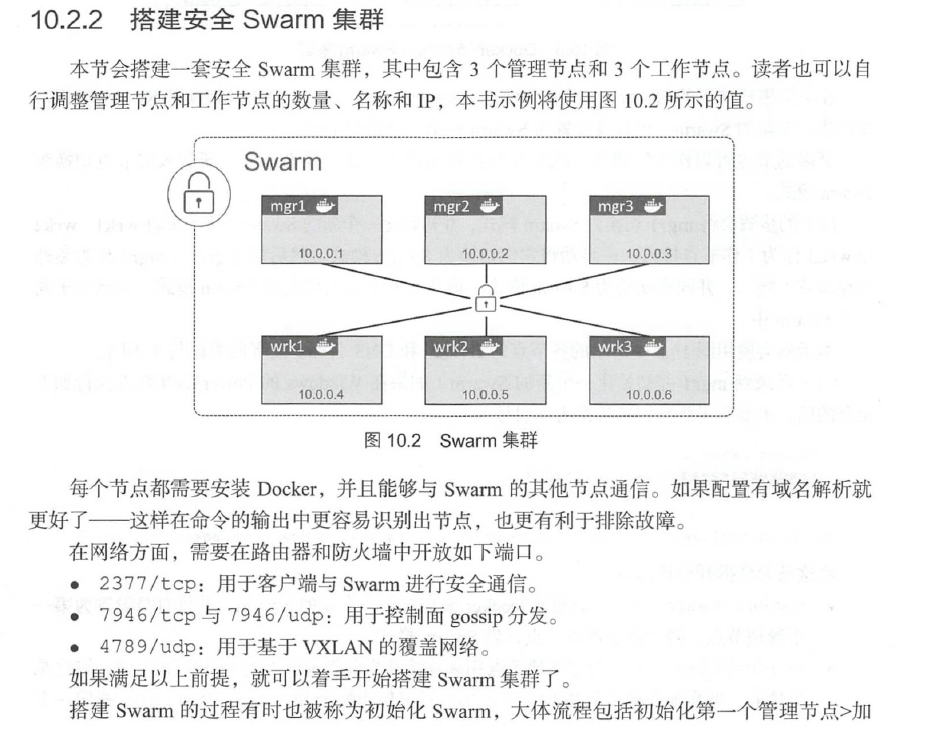
### Docker swarm

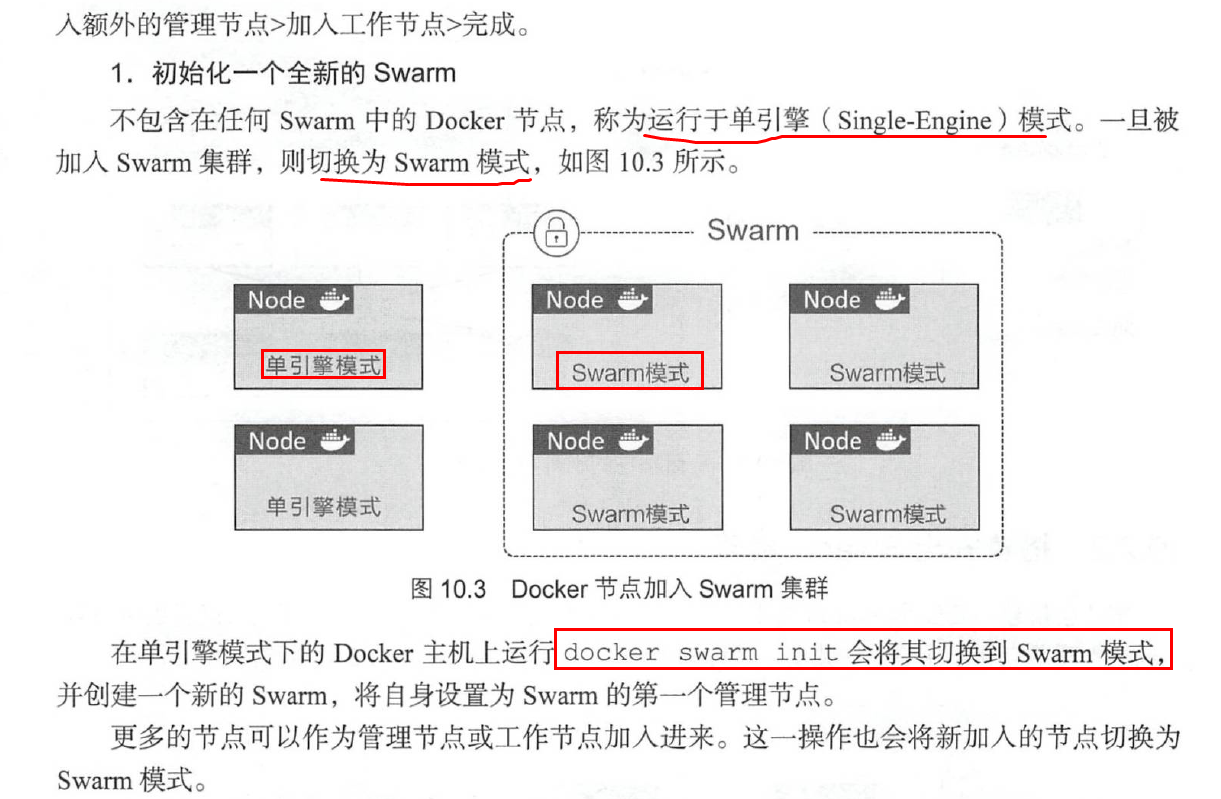
Docker的集群管理工具, swarm有两个核心组件

1. 安全集群
2. 编排引擎(微服务应用编排引擎)

一个Swarm由一个或者多个Docker节点组成,可以是物理服务器/虚拟机/树莓派/云实例 唯一的要求就是要通过可靠地网络连接!!







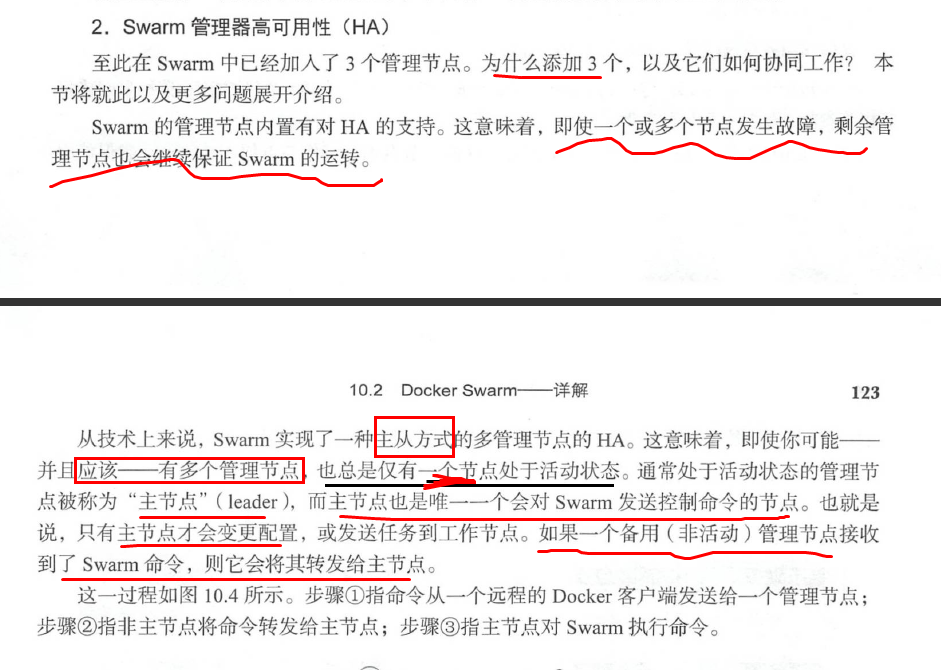
切换成swarm 模式后可以使用

Docker swarm join-token 命令设置连接凭证 工作节点后加 worker,管理节点加manager

Docker node 命令了 ls 列出节点 其中manager status 无数据的是工作节点

Swarm的管理器高可用性HA

Swarm 实现了主从方式的多管理节点的HA,一个或者多个节点挂了,剩余管理节点能够保证swarm的运转,



管理节点是leader或者follower; HA有两个最佳实践: 部署奇数个管理节点; 不要部署过太多管理节点(3或者5个)

Docker node 查看swarm模式里面的节点