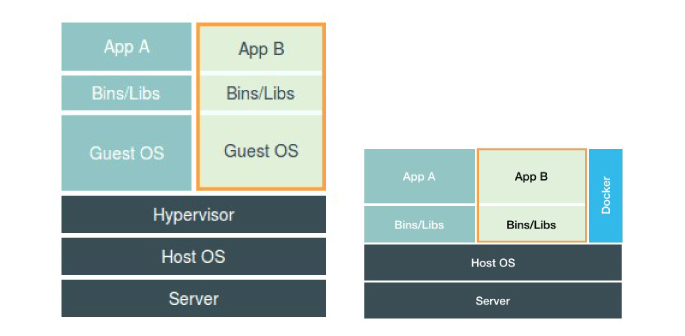
### 容器技术

对 Docker 项目来说，它最核心的原理实际上就是为待创建的用户进程：

1. 启用 Linux Namespace 配置；
2. 设置指定的 Cgroups 参数；
3. 切换进程的根目录（Change Root）

#### 容器和虚拟机的区分



Docker Engine 等这些容器管理工具并不会像Hypervcisor一样 对应用进程的隔离环境负责，不会创建任何实体的容器，实际依靠的还是宿主机OS本身。

Docker和宿主机上的其他进程一样，由HostOS统一管理。

* 容器更加“敏捷”和“高性能”，更加细粒度
* 容器相对的不足则是**隔离的不够彻底**。

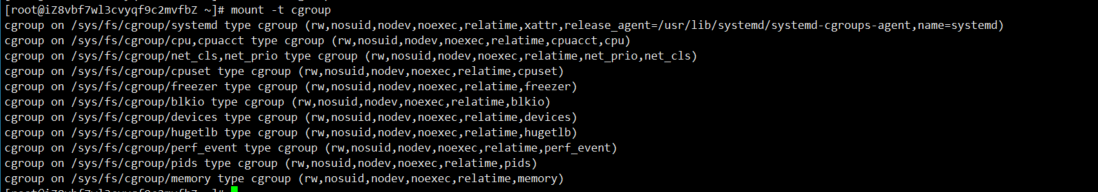
1. 容器必须**共享同一个宿主机内核**，因此不支持不同OS内核要求的应用运行。(Seccomp技术可以进行过滤与加固)
2. 在Linux当中，**并不是所有的资源和对象都可以被Namespace化**，比如**时间**。Cgroups也不能限制所有的资源和对象，例如/proc存储的内核运行状态文件，因为/proc 文件系统不了解 Cgroups 限制的存在，所以无法限制。

### 限制与隔离

Namespace对进程进行视图隔离。可以看关于虚拟化和容器化技术的文档。

Cgroups对进程所能使用的资源和对象进行限制，还可以对进行进行优先级设置、审计、以及将进行挂起和恢复等操作。

Cgroups通过文件系统给出用户接口，都在/sys/fs/cgroup,其下的cpu、cpuset、pids等是**子系统（Sub System)**,表示当前可以被 Cgroups 进行限制的资源种类。



* 本质上，容器是一个**单进程**模型。

### 镜像和容器

**镜像和容器的关系**类似于程序与进程的关系，镜像类似于静态的程序，安静的存储在磁盘当中。而容器类似于动态的进程，依据镜像在CPU、内存中运行。

#### 镜像

首先了解一下，Docker和其他PaaS最重要的区别：镜像功能。

**挂载在容器根目录上、用来为容器进程提供隔离后执行环境的文件系统，就是所谓的“容器镜像”。它还有一个更为专业的名字，叫作：rootfs（根文件系统）**。

**rootfs 只是一个操作系统所包含的文件、配置和目录，并不包括操作系统内核**。在 Linux 操作系统中，这两部分是分开存放的，**操作系统只有在开机启动时才会加载指定版本的内核镜像。**

rootfs的重要特性： **一致性**，解决了因为本地和云服务环境的不同，繁琐的应用打包过程，**这种深入到操作系统级别的运行环境一致性，打通了应用在本地开发和远端执行环境之间难以逾越的鸿沟。**

**rootfs 里打包的不只是应用，而是整个操作系统的文件和目录，也就意味着，应用以及它运行所需要的所有依赖，都被封装在了一起。**因为OS才是一个应用运行所需要的最完整的依赖库。

如何制作rootfs?? 每次开发应用或者升级应用，都要重复制作吗？？？

解决方法：

* 在制作rootfs的过程当中，每操作修改一次，就保存一个rootfs,给其他人提供重复使用的基础。但是很容易产生大量不同的rootfs，极度碎片化。
* 采用**增量**的方式修改，维护相对于base rootfs修改的内容，而不是每次都copy一个副本。

#### 镜像的分层

Docker 引入层(Layer)的概念，用户制作镜像的每一步操作，都会生成一个层，也就是一个增量 rootfs。

使用到 **联合文件系统（UnionFS）**的技术：主要功能将多个不同位置的目录联合挂载到同一个目录下。

Linux下有多种UnionFS的实现：AuFS、OverlayFS和BtrFS。  
 OverlayFS的实现原理：<https://cloud.tencent.com/developer/article/1684523>

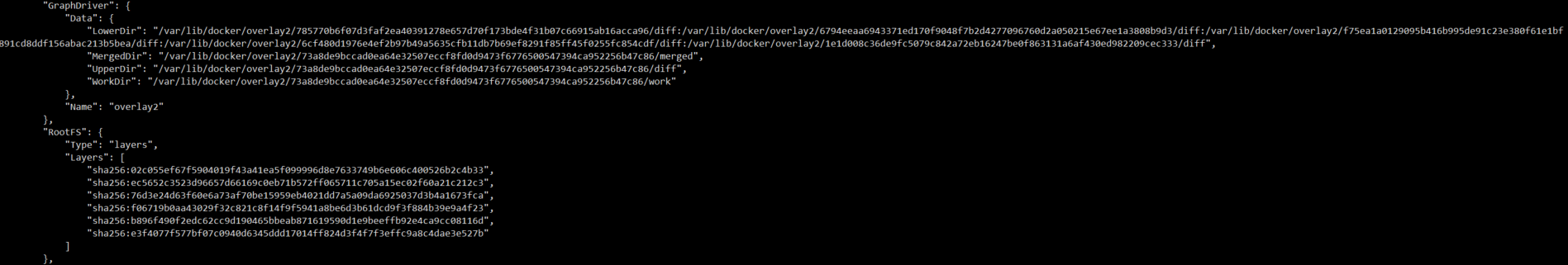
具体实现，待后续研究.............................................................

常见发行版所使用的实现：

CentOS, Storage Driver: overlay2、overlay  
debain, Storage Driver: aufs  
RedHat, Storage Driver: devicemapper

在实际中看到的分层如下：

docker image inspect redis:latest

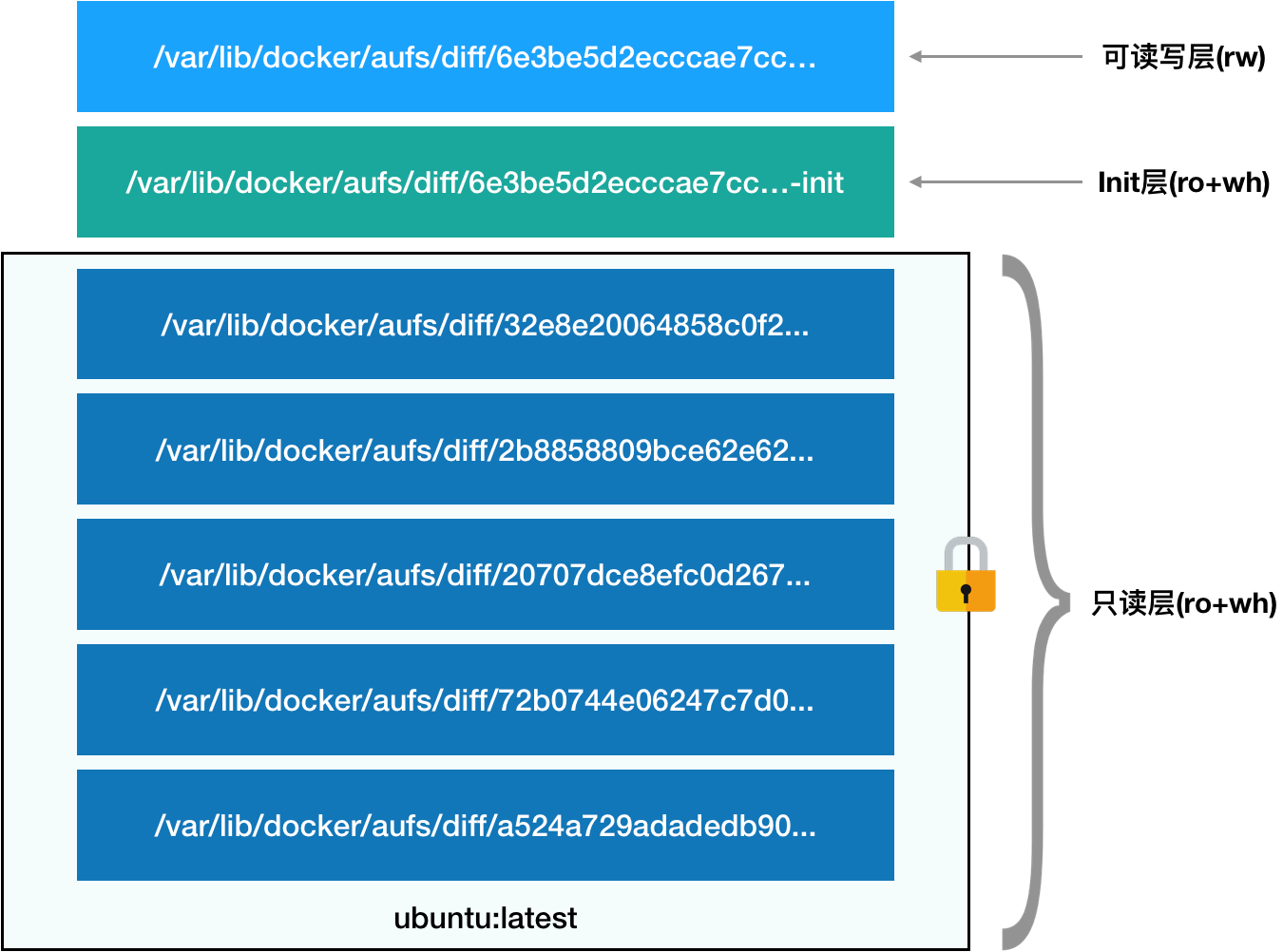


GraphDriver管理实际存储进行的文件系统。 RootFS表示增量的几个层，增量的方式分别包含了redis的一部分。

这三个层就是三个增量 rootfs每一层都是对应应用的文件系统的一部分，使用镜像时，Docker 会把这些增量联合挂载在一个统一的挂载点上。

其他容器的示例图：

*逻辑分层示例图*



如何修改只读层的文件？

FS 一般会在可读写层创建一个 **whiteout** 文件，**把只读层里的文件“遮挡”起来**。

最上面这个**可读写层**的作用，就是专门用来存放你修改 rootfs 后产生的增量，无论是增、删、改，都发生在这里。

**init层**：Init 层是 Docker 项目单独生成的一个内部层，专门用来存放 /etc/hosts、/etc/resolv.conf 等信息。

Docker 做法是，**在修改了这些文件之后，以一个单独的层挂载了出来**。而**用户执行 docker commit 只会提交可读写层，是不包含这些内容的。**

docker commit，实际上就是在容器运行起来后，把最上层的“可读写层”，加上原先容器镜像的只读层，打包组成了一个新的镜像。

#### Volume机制

解决的问题：

1. 容器里进程新建的文件，怎么才能让宿主机获取到？
2. 宿主机上的文件和目录，怎么才能让容器里的进程访问到？

**Volume 机制，允许你将宿主机上指定的目录或者文件，挂载到容器里面进行读取和修改操作**。

#### 本质的看待

一个正在运行的 Linux 容器，其实可以被“一分为二”地看待：

1. 一组联合挂载在 `/var/lib/docker/aufs/mnt（根据使用的UnionFS实现有所不同） 上的 rootfs，这一部分我们称为“容器镜像”（Container Image），是容器的静态视图；
2. 一个由 Namespace+Cgroups 构成的隔离环境，这一部分我们称为“容器运行时”（Container Runtime），是容器的动态视图。