##背景

在没有镜像仓库的场景下，通过将容器镜像保存（isula/docker save)的方式进行文件分发。

##优势

尽量减小分发带宽和存储带宽。

##示例描述

举一个比较常见的 java 应用的容器镜像例子：base+jvm+app

假如有多个 APP，依赖的都是同样的 base 镜像（基础镜像，纯 rootfs，通过 isula/docker import 导入）和 jvm，那假设 base 有 100M，jvm 有 200M，app 有 300M，也就是 save 下来的镜像是 600M

假设有 10 个 APP，我们就需要在集群内分发 600M*10 大小的文件

分层镜像工具将上面的 base+jvm+app 抽象为 base+lib+app 这几层，实现暴力地将 save 下来的容器镜像分成 1+N+M 的格式。这样，如果有 10 个镜像，那么我们只需要分发：100M+200M+300M*10 大小的文件。

前提:

这 10 个容器镜像的 base 层和 lib 层镜像都是一样的，举个例子，app 是通过如下方式构建出来的。

isula import ---->base:latest

FROM base:latest---->lib:latest

FROM lib:latest--->app1:latest

FROM lib:latest--->app2:latest

...

##分层镜像保存工具

成果：一个二进制可执行文件

输入：一个容器镜像 tar 包（save）/app1:latest，lib 的层数（默认为 0），一个文件夹路径

输出：文件夹下生成 base.tar.gz*1,lib.tar.gz*1,app.tar.gz*1,manifest.json*1,其中，manifest.json 描述每个压缩包的 sha256sum,避免镜像合成导入之后与原来的镜像有差异。

说明：base 镜像固定为 1 层，因为从零开始构建的镜像通常需要 import 一个基础镜像。

##分层镜像导入工具

成果：一个二进制的可执行文件

输入：分层镜像保存工具的输出目录，目录内有 3 个压缩包+一个 manifest.json

输出：isula images 可以查看到导入的容器

##进阶

输入中指定一个 manifest 来描述镜像的分层关系和层数，不限于 base+lib+app

这 3 层

##陈俊宇同学的问题

结论：理解全部正确，你已经得到它了^_^。

1.是需要转移多个镜像才有效果，越多效果越大

2.OS 不同那就不能使用分层镜像工具，所以这个工具是有局限性的，但这个局限性在大多数场景不存在，通常一个公司或大部门，OS 层是经过裁剪且一致的

3.理解正确

4.docker save -o apps.tar [img1] [img2] ...的确可以复用相同层，没问题。但是从应用场景看，如果业务差距比较大，或者多个部门业务差异，部门间、业务间不能将镜像打包在一起；另外，base 和 lib 是可以归档的哦，比如 ftp 归档，之后 build 出来，直接分发 app 层即可