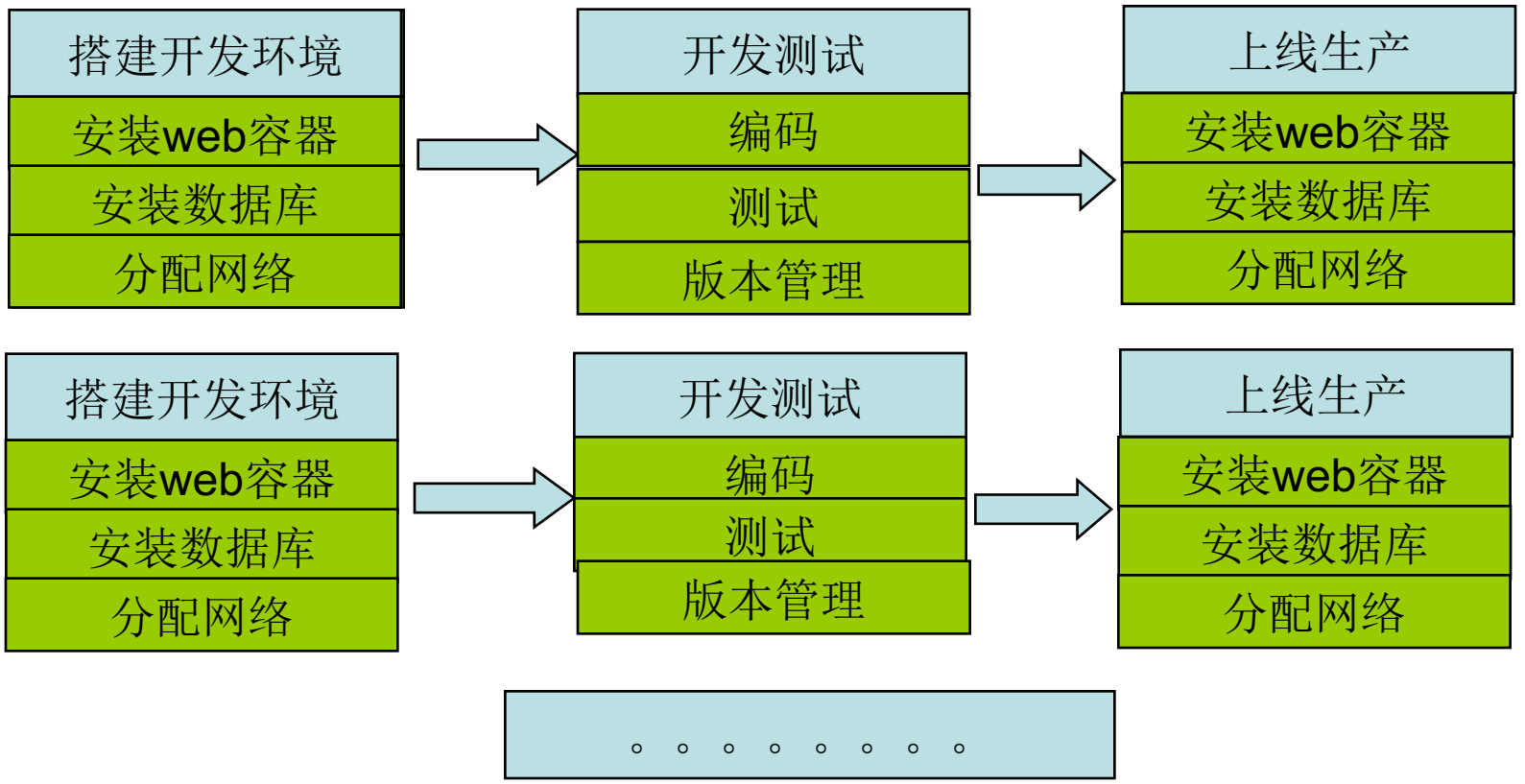
第一讲 docker基础

田翀翀
tianccm@gmail.com

目录

传统的开发部署流程

（仅包括纯技术部分）：



传统模式的问题

1、资源利用效率低

2、单物理机多应用无法有效隔离（进程空间，cpu资源，磁盘）

3、运维部署不便

4、测试、版本管理复杂

5、迁移成本高

6、传统虚拟机，空间占用大，启动慢，管理复杂

。 。 。 。 。

这里是剧本1.0

while (true) {

测试：“有**Bug**！” (潜台词：嘿嘿、小样儿~)

开发：“你看，这段代码在我机器上没问题啊！
你环境不对吧？”（潜台词：滚）

测试：“哦，我回去看看。”(奇怪。。。)

开发：“我也看看吧。
shit，服务器配置写错了”(偷偷改一下^_^)

}

Docker是什么

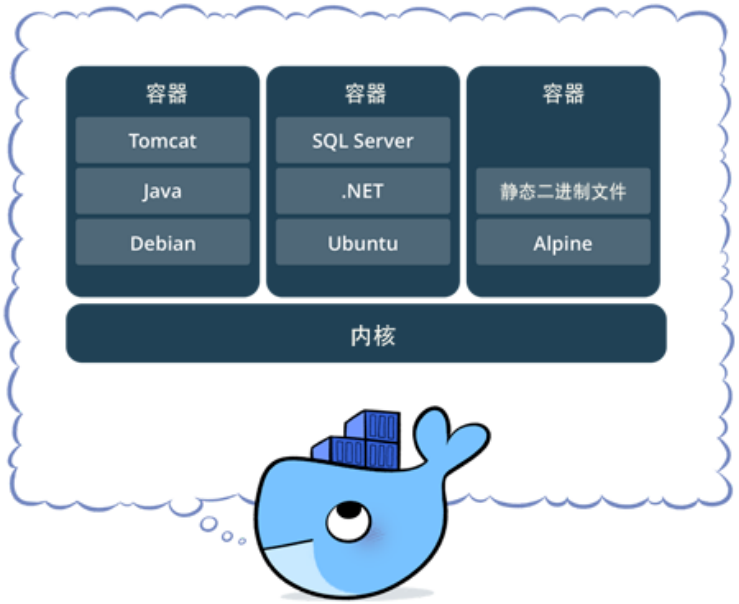
- Docker是基于容器技术的轻量级虚拟化解决方案
- Docker是容器引擎，把Linux的cgroup、namespace等容器底层技术进行封装抽象，为用户提供了创建和管理容器的便捷界面（包括命令行和API）
- Docker 是一个开源项目，诞生于 2013 年初，基于 Google 公司推出的 Go 语言实现
- 微软，红帽Linux，IBM，Oracle等主流IT厂商已经在自己的产品里增加对Docker的支持。
- Google 每周启动超过20亿个容器进行业务服务，于上个世纪90年代已经开始大规模使用容器技术

Docker是什么

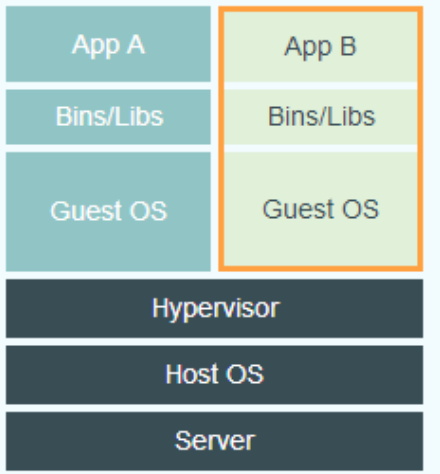
- **一句话概括容器：容器就是将软件打包成标准化单元，以用于开发、交付和部署。**
- **容器镜像是轻量的、可执行的独立软件包**，包含软件运行所需的所有内容：代码、运行时环境、系统工具、系统库和设置。
- **容器化软件适用于基于Linux和Windows的应用，在任何环境中都能够始终如一地运行。**
- **容器赋予了软件独立性**，使其免受外在环境差异（例如，开发和预演环境的差异）的影响，从而有助于减少团队间在相同基础设施上运行不同软件时的冲突。

Docker是什么

- 通俗的描述，容器就是一个存放东西的地方，就像书包可以装各种文具、衣柜可以放各种衣服、鞋架可以放各种鞋子一样。现在所说的容器存放的东西可能更偏向于应用比如网站、程序甚至是系统环境。

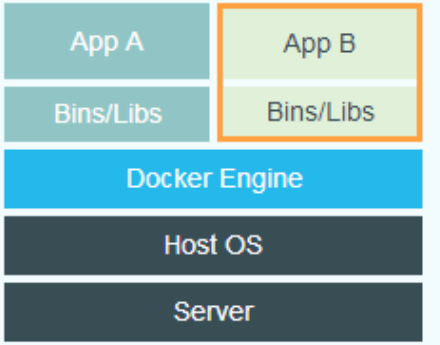


Docker与VM区别



Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

Docker与VM区别

物理机

物理机

Docker与VM区别

虚拟机：

一栋楼包含多套房，
一套房一户人家，
共享地基，共享花园，独立卫生间、厨房和宽带

Docker与VM区别

容器：

一套房隔成多个小隔间(胶囊式公寓)，每个胶囊住一位租户，共享地基，共享花园，还共享卫生间、厨房和宽带

Docker--轻量级虚拟化容器技术

1、秒级启动，秒级停止，空间资源占用极少（几M）

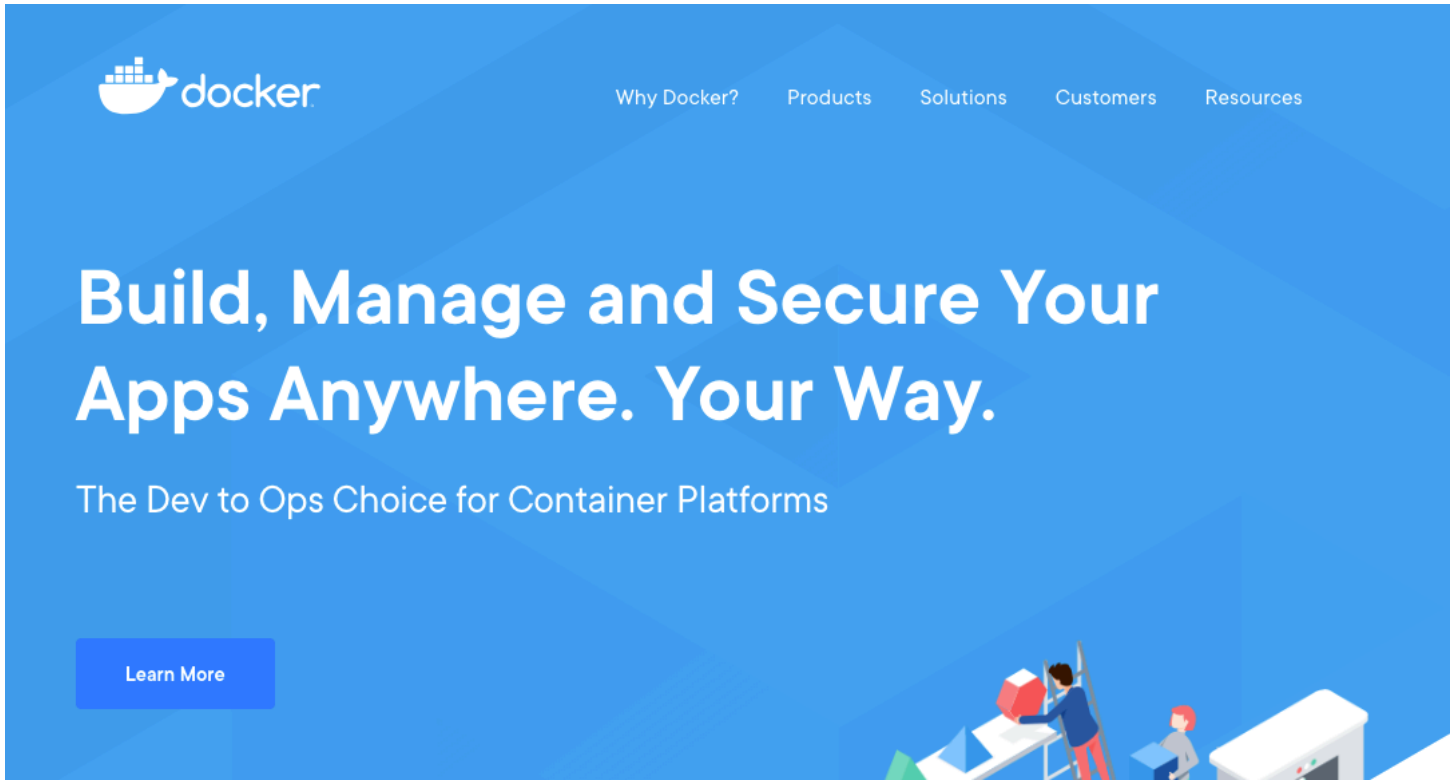
2、实现进程级别的隔离

3、可在普通服务器上建立上百个docker实例

4、加快开发测试部署的速度

5、简化版本管理

Docker的slogan



开发和运维的有效隔离

一个IT系统应该包含如下几个层次:

- 应用程序
- 运行时平台（bin/framework/lib）
- 操作系统
- 硬件（基础设施）

开发人员的主要工作是应用程序的编码、构建、测试和发布，涉及应用程序和运行时平台这两层。而运维人员的工作则涉及从硬件、操作系统到运行时平台的安装、配置、运行监控、升级和优化等工作。docker提供了一种运行时环境,隔离了上层应用于下层操作系统和硬件的关联,使得术业有专攻

这里是剧本2.0

while (true) {

测试：“有Bug！” (嘿嘿、小样儿~)

开发：“都是一套镜像，怎么可能不一样！！（滚！）

测试：“哦，真是一样，我再看看。”(真是我的问题？)

开发：“我得儿意地笑，我得儿意地笑~~”

}

开始安装docker

由于apt官方库里的docker版本可能比较旧，所以先卸载可能存在的旧版本：

$ sudo apt-get remove docker docker-engine docker-ce docker.io

更新apt包索引：

$ sudo apt-get update

安装以下包以使apt可以通过HTTPS使用存储库（repository）：

$ sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common

添加Docker官方的GPG密钥：

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

使用下面的命令来设置stable存储库：

$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

再更新一下apt包索引：$ sudo apt-get update

安装最新版本的Docker CE：$ sudo apt-get install -y docker-ce

在生产系统上，可能会需要应该安装一个特定版本的Docker CE，而不是总是使用最新版本：列出可用的版本：$ apt-cache madison docker-ce

验证安装是否正确

docker run hello-world

验证安装是否正确

```
root@i-l1t981t3:~# clear
root@i-l1t981t3:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9db2ca6ccae0: Pull complete
Digest: sha256:4b8ff392a12ed9ea17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/

root@i-l1t981t3:~# _
```

docker背后干了什么

- 我们创建了一个容器
- 它拥有：
  文件系统（基于hello-world镜像）
  网络栈（具有私有网络服务）
  进程空间
- 自动安装
  docker会自动检查本地是否有hello-world镜像，如果没有则自动下载并启动
- 还可以向宿主机打印屏幕信息

docker的重要概念

- 镜像（image）
- 容器（container）
- 数据卷（volumes）
- 链接（links）
- 仓库（Repository）
- 网络（Networks）

docker-镜像

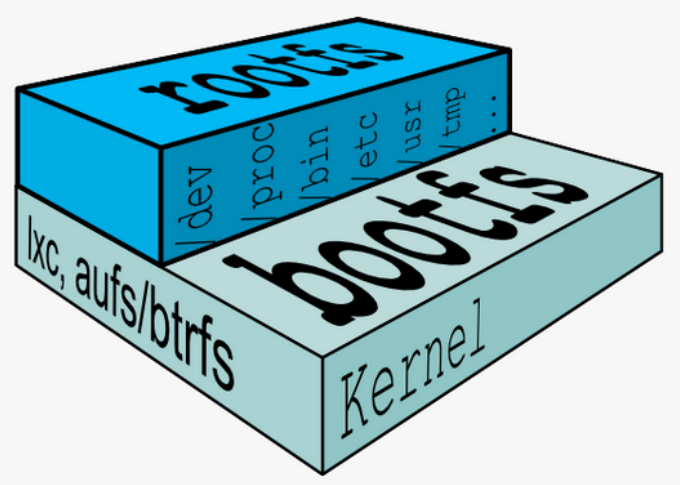
- Docker 的镜像类似虚拟机的快照，但是更轻量

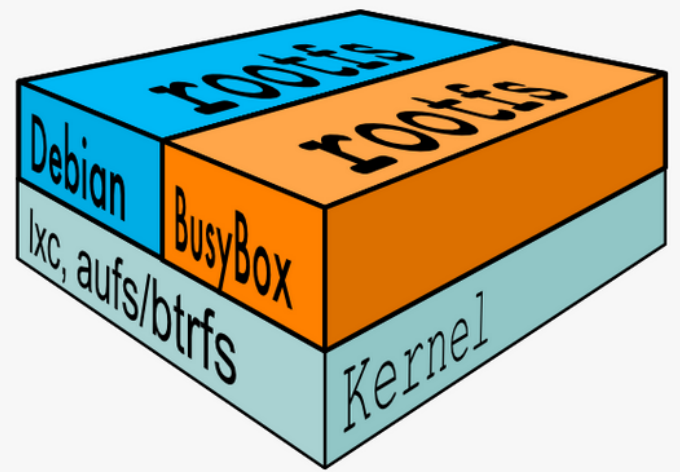
- 例如：一个镜像可以包含一个完整的 Linux 操作系统环境，里面仅安装了 Tomcat或用户需要的其它应用程序

- 镜像用来创建容器，可迁移

Docker的文件系统

- 典型的启动Linux运行需要两个FS: bootfs + rootfs:





AUFS (AnotherUnionFS)

- AUFS 是一种 Union FS, 简单来说就是支持将不同目录挂载到同一个虚拟文件系统下的文件系统。

- 更进一步的理解, AUFS支持为每一个成员目录(类似Git Branch)设定 readonly、readwrite 和 whiteout-able 权限, 同时 AUFS 里有一个类似分层的概念, 对 readonly 权限的 branch 可以逻辑上进行修改(增量地, 不影响 readonly 部分的)。

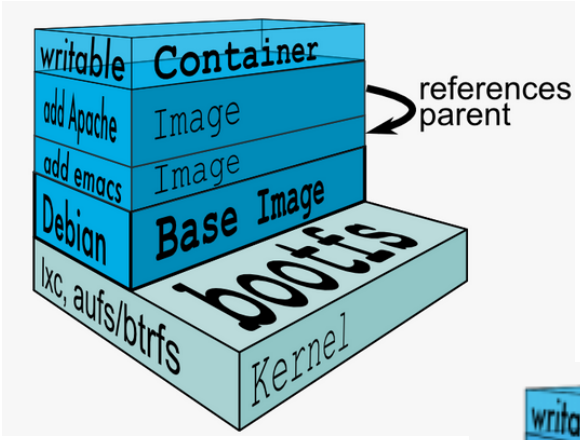
Docker的文件系统

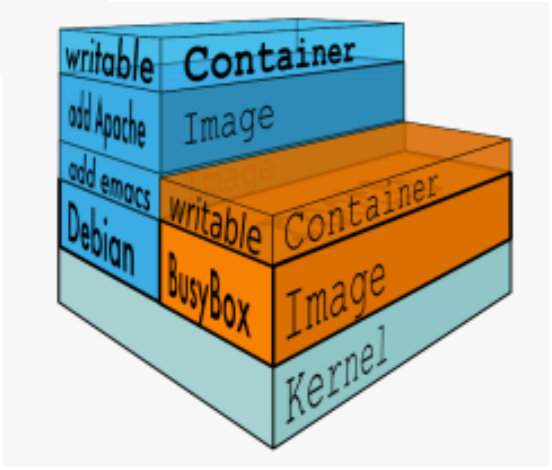
- 通过分层的方式记录文件的累积变化情况。
- 例如 tomcat, oracle, weblogic镜像都可以共享同一个Linux基础镜像，每一个镜像保存的只是在基础镜像上他们修改的部分
- Docker的镜像可以非常多，但是每个都很小，而且加载启动非常快。

Docker的文件系统

Docker AUFS特性

- Docker镜像位于bootfs之上
- 每一层镜像的下面一层称为其父镜像(父子关系)
- 第一层镜像为Base Image
- 容器在最顶层
- 其下的所有层都为readonly
- Docker将readonly的FS层称作"image"





Docker-镜像

- 获取镜像
  - docker pull mysql:5.7.23 不带仓库名称则默认从Docker Hub下载
  - docker pull ubuntu 不带版本版本号则默认下载latest版本
  - docker pull d1.dockerpool.com:5000/ubuntu 指定仓库下载

- 上传镜像
  - docker push registry:5000/centos_cms:v1.1

- 查看镜像
  - docker images

- 搜索镜像
  - docker search mysql

Docker-镜像

- 删除镜像
  - docker rmi registry:5000/centos_cms:v1.0
  - Docker rmi 5506ed32sd3w2

- 提交镜像
  - docker commit 890sadfae12se2 registry:5000/centos_cms:v1

- 从容器导出、导入镜像
  - docker export 890sadfae12se2 > centos_cms11.tar
  - cat centos_cms11.tar | docker import - registry:5000/centos_cms:v1

- 存出和载入镜像
  - docker save registry:5000/centos_cms:v1.0 > centos_cms11.tar
  - docker load < centos_cms11.tar

Docker-容器

- 等同于从快照中创建虚拟机
- 容器是从镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。
- 可以把容器看做是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。
- 镜像本身是只读的，容器从镜像启动之后，Docker会在镜像的最上层创建一个可写层，而镜像本身将保持不变

Docker-容器

- 新建并启动容器
  - docker run [OPTIONS] IMAG [COMMAND]
  - OPTIONS:
    - -ti -t 让docker分配一个伪终端并绑定到容器的标准输入上，-i表示让容器的标准输入保存打开
    - -d 让docker容器在后台以守护态形式运行
    - -p 桥接模式，端口映射
    - --net=host host模式启动
    - --restart=always 一直重启
    - --privileged=true 高级权限
    - --log-driver=none 不打印容器级别日志
    - --name 容器命名
  - COMMAND
    - /run.sh

Docker-容器

- docker run -d -p 3307:3306 -p 5667:5666 -name mysqlos --restart=always --privileged=true --log-driver=none registry:5000/centos_mysql_os:v5.6.2 /run.sh

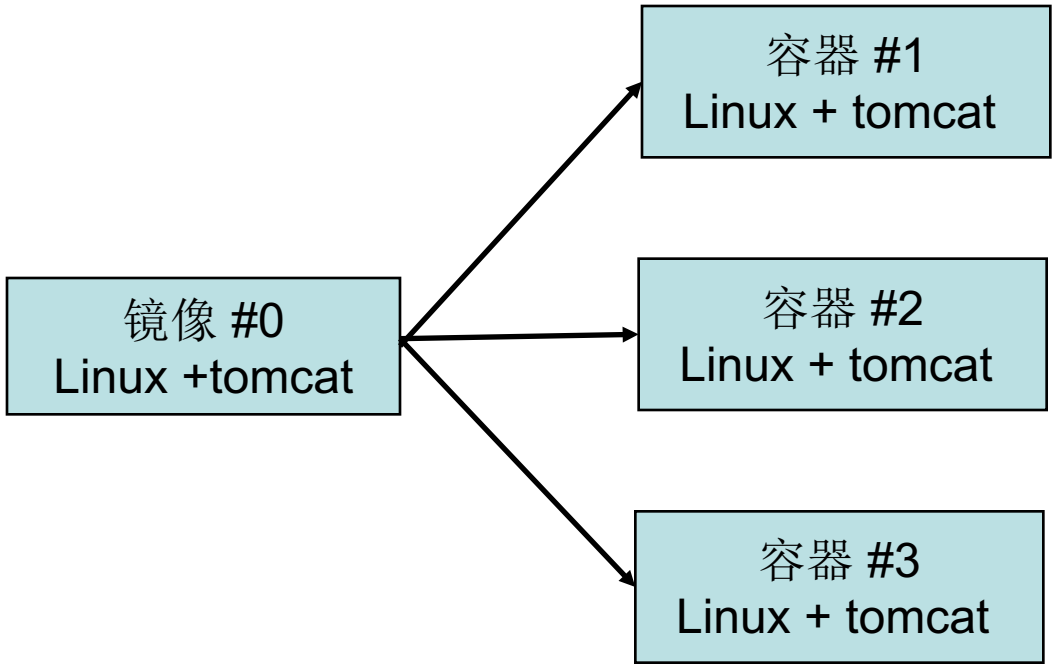
- docker run -d --net=host –name os --restart=always -v /home/data/osdata:/usr/local/nginx/html/mnt --privileged=true --log-driver=none registry:5000/centos_os:v2.1.0 /run.sh

- 进入容器
  - exec
    - docker exec -it containerID /bin/bash
  - ssh
    - ssh –p 222 root@hostIP

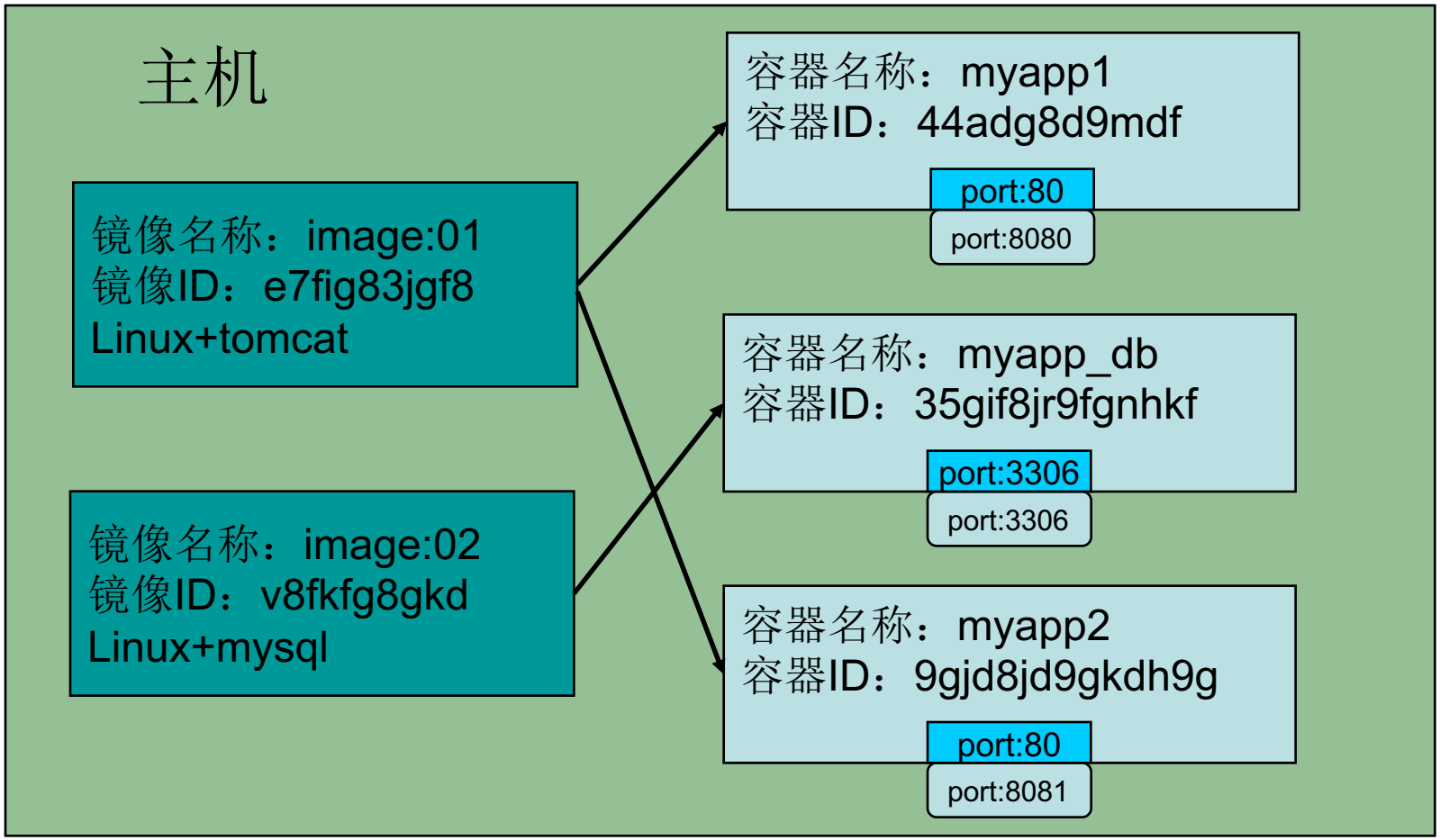
Docker-容器

- 停止容器
  - docker stop containerID
- 启动容器
  - docker start containerID
- 重启容器
  - docker restart containerID
- 删除容器
  - docker rm –f containerID

从同一个镜像启动多个容器



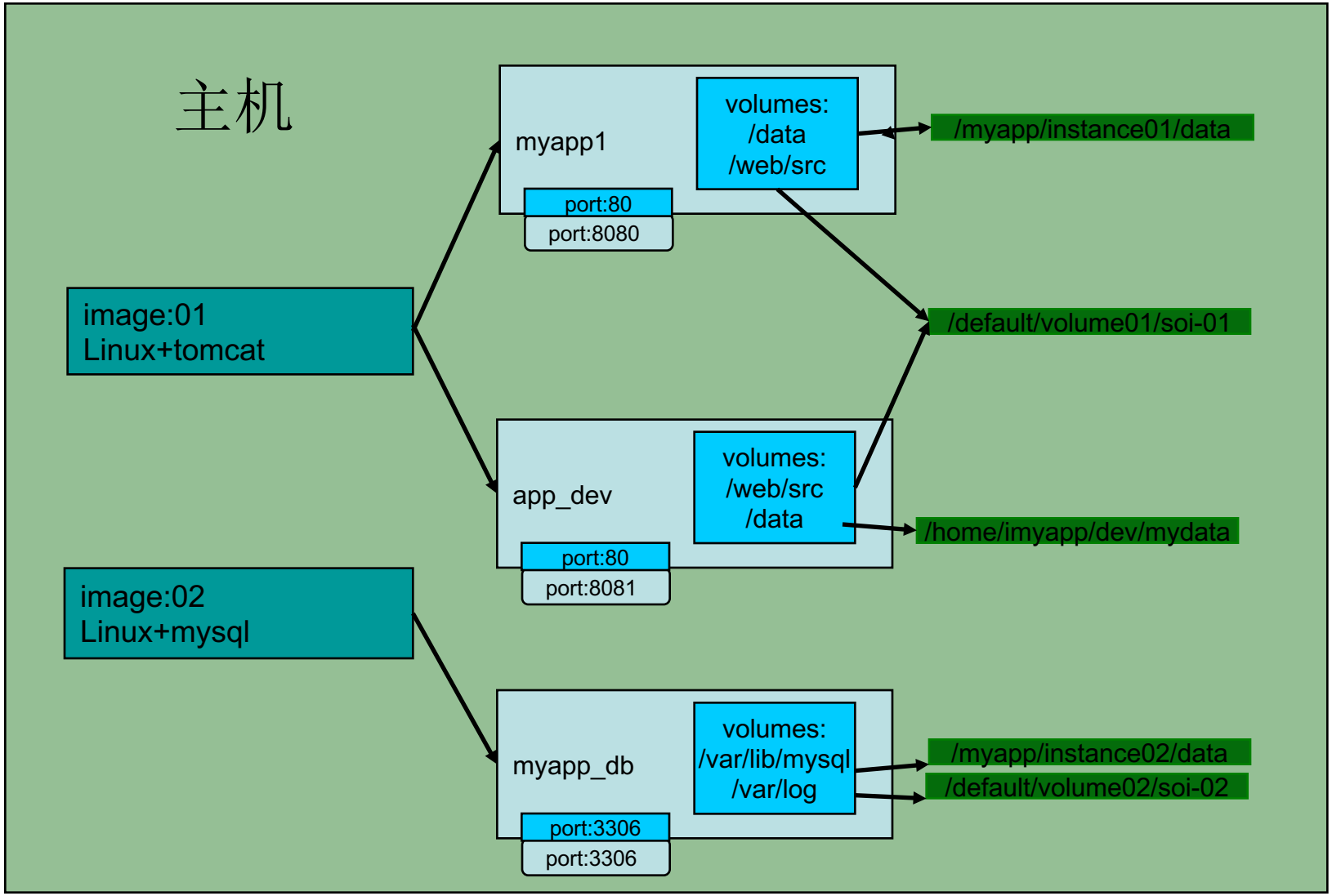
容器端口映射


主机
镜像名称：image:01
镜像ID：e7fig83jgf8
Linux+tomcat
镜像名称：image:02
镜像ID：v8fkfg8gkd
Linux+mysql
容器名称：myapp1
容器ID：44adg8d9mdf
port:80
port:8080
容器名称：myapp_db
容器ID：35gif8jr9fgnhkf
port:3306
port:3306
容器名称：myapp2
容器ID：9gjd8jd9gkdh9g
port:80
port:8081

数据卷

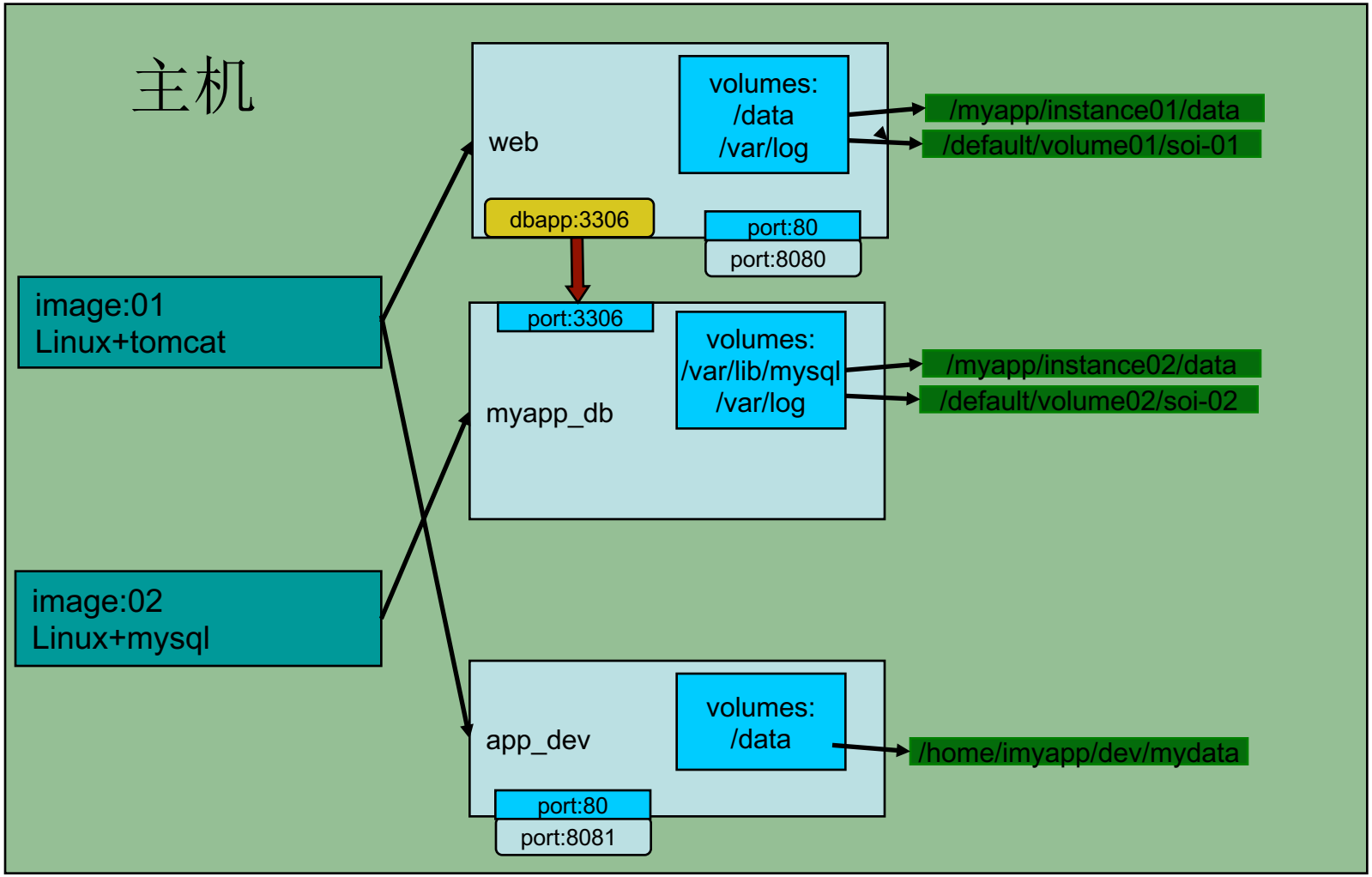
数据卷是一个可供一个或多个容器使用的特殊目录

- 进程和数据的分离
- 实际保存在容器之外，从而允许在不影响数据的情况下销毁、重建、修改、丢弃容器
- 可用于数据持久化
- 数据卷的使用，类似于 Linux 下对目录或文件进行 mount。
- 数据卷的共享， 可以在多个容器之间共享数据卷


主机
myapp1
volumes:
/data
/web/src
/myapp/instance01/data
port:80
port:8080
image:01
Linux+tomcat
/default/volume01/soi-01
app_dev
volumes:
/web/src
/data
/home/imyapp/dev/mydata
port:80
port:8081
image:02
Linux+mysql
myapp_db
volumes:
/var/lib/mysql
/var/log
/myapp/instance02/data
/default/volume02/soi-02
port:3306
port:3306

链接

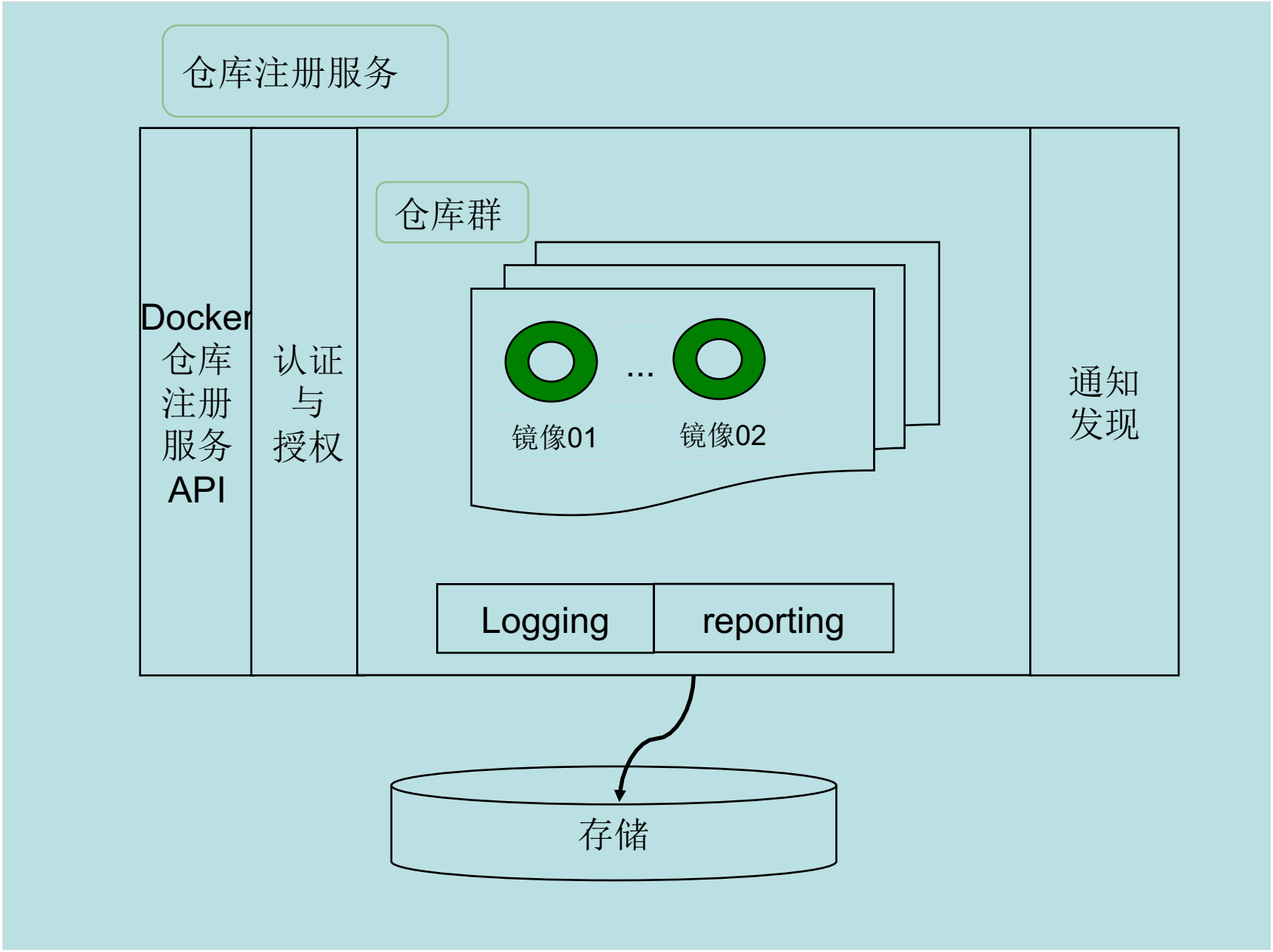
- 容器的连接（linking）系统是除了端口映射外，另一种跟容器中应用交互的方式
- 在源和接收容器之间创建一个隧道，接收容器可以看到源容器指定的信息
- Docker 在两个互联的容器之间创建了一个安全隧道，而且不用映射它们的端口到宿主主机上。从而避免了暴露关键系统（如数据库）端口到外部网络上.



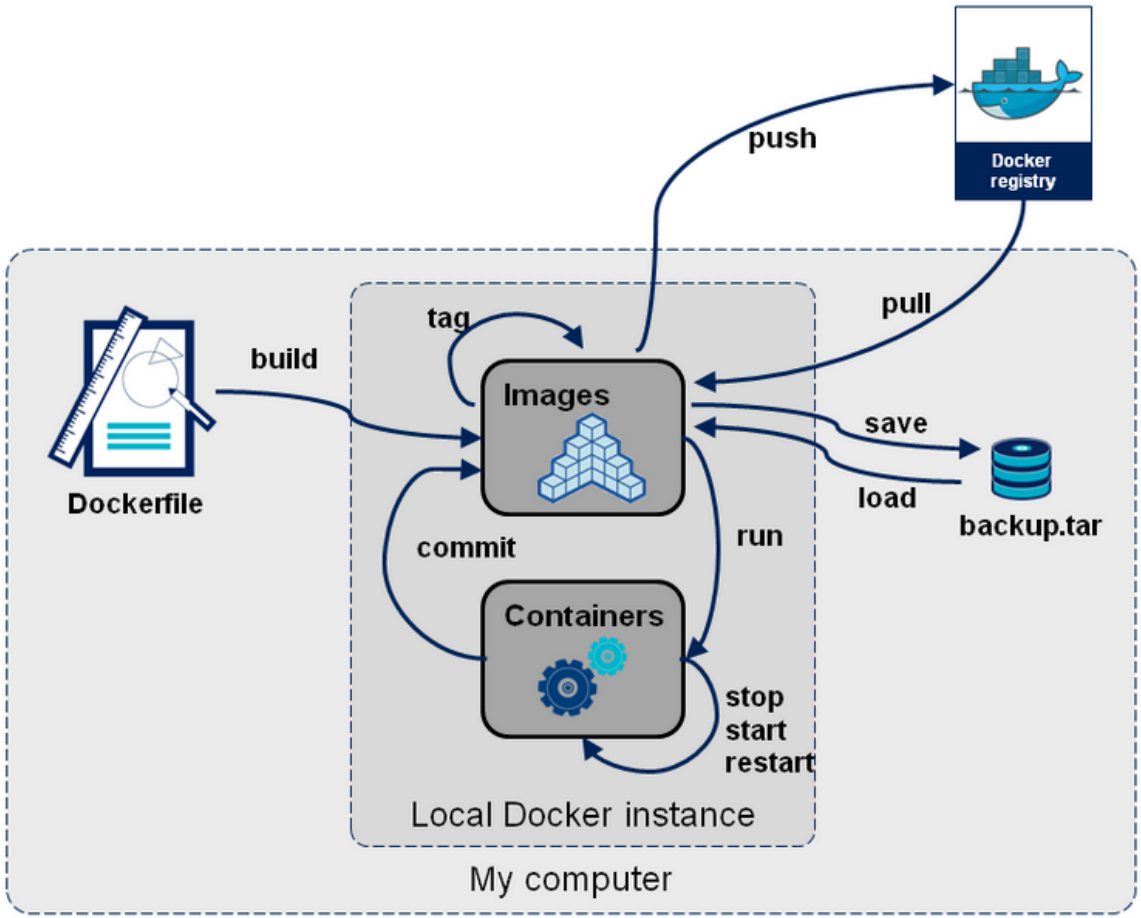
这里实际上3306端口只有web容器和myapp_db容器是可见的，对其他容器是不可见的。

仓库及仓库注册服务器

- 仓库是集中存放镜像文件的场所
- 仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签
- 仓库分为公开仓库（Public）和私有仓库（Private）两种形式
- push 镜像到仓库,从仓库pull下镜像


仓库注册服务
Docker
仓库
注册
服务
API
认证
与
授权
仓库群
...
镜像01
镜像02
通知
发现
Logging
reporting
存储

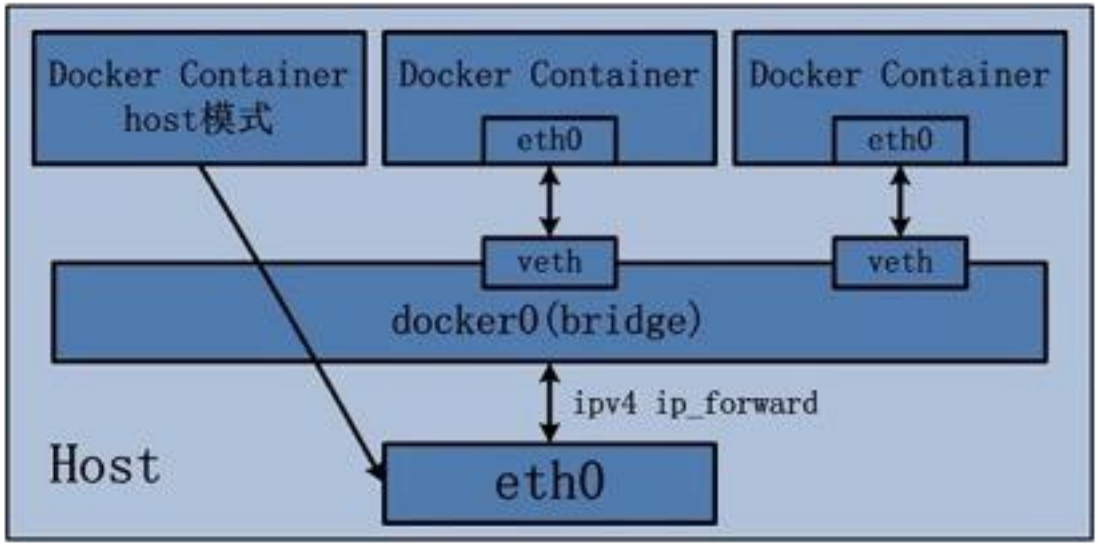
镜像、容器、仓库的关系


Docker registry
push
pull
tag
build
Images
save
load
backup.tar
Dockerfile
commit
run
Containers
stop
start
restart
Local Docker instance
My computer

网络|Host

- Docker使用了Linux的Namespaces技术来进行资源隔离，如PID Namespace隔离进程。
- 一个Network Namespace提供了一份独立的网络环境，包括网卡、路由、Iptable规则等都与其他的Network Namespace隔离。一个Docker容器一般会分配一个独立的Network Namespace。
- host模式下，容器将不会获得一个独立的Network Namespace，而是和宿主机共用一个Network Namespace。容器将不会虚拟出自己的网卡，配置自己的IP等，而是使用宿主机的IP和端口。

网络|Host


Docker Container
host模式
Docker Container
eth0
Docker Container
eth0
veth
veth
docker0(bridge)
ipv4 ip_forward
Host
eth0

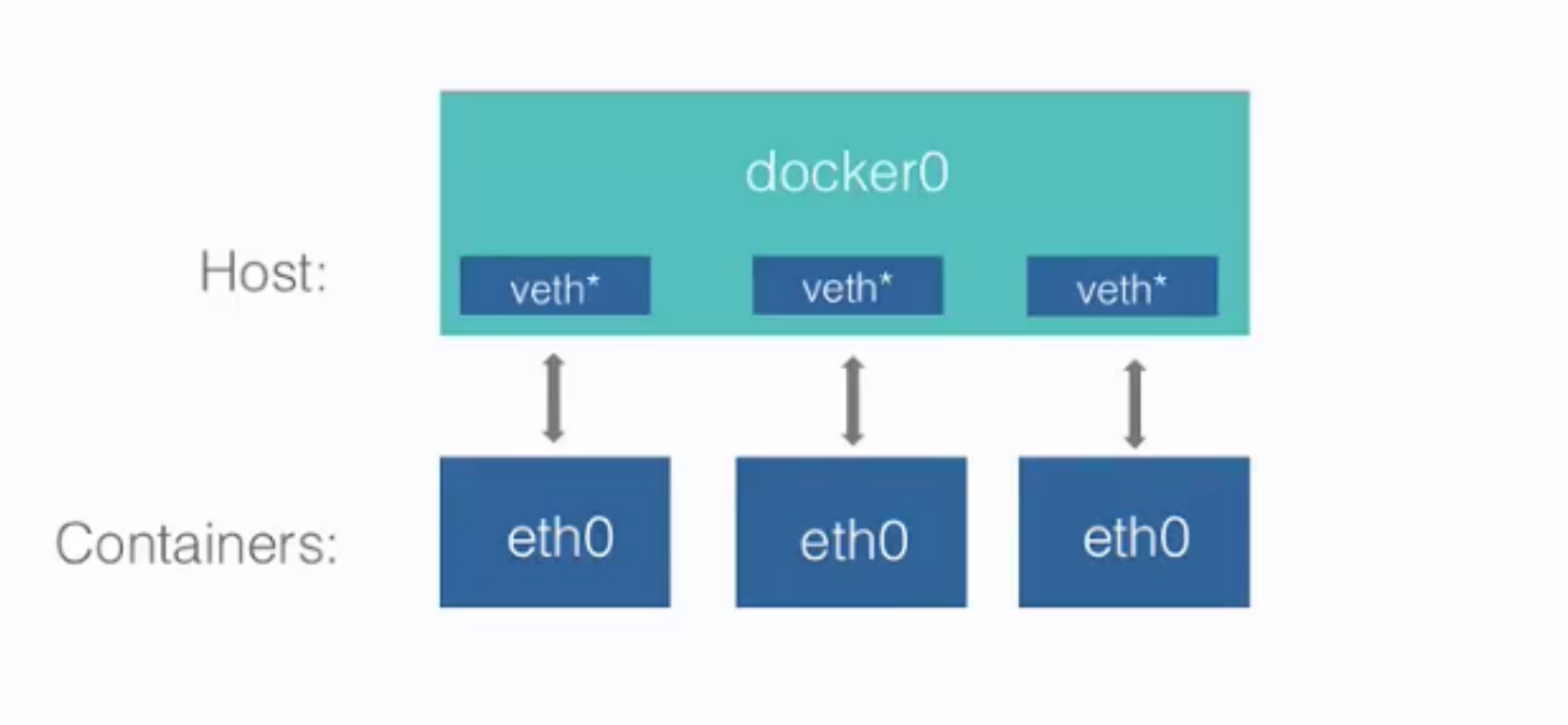
网络|Host

- Host模式使用场景
  - 服务端与客户端进行通信的端口不能提前指定，只能临时协商，例如FTP服务
  - 信源输入、结果数据输出有物理网卡的限制，即容器内部需要绑定物理网卡信息，例如编码器、源服务器
  - 其他一些只能在host模式下才能正常工作的场景，例如容器内部需要挂载外部nfs路径的场景等

- Host注意事项
  - 安全性，能够连接到物理主机上的外部主机，均能够连接到容器中（桥接模式指定网卡，只有与网卡相同的VLAN的主机才可连接到容器中）
  - 端口资源占用

网络|NAT桥接

- Docker默认的网络设置，此模式会为每一个容器分配Network Namespace、设置IP等，并将一个主机上的Docker容器连接到一个虚拟网桥上



网络|NAT桥接

- 向外通信：
  - IP包首先从容器发往自己的默认网关docker0，包到达docker0后，也就到达了主机上。然后会查询主机的路由表，发现包应该从主机的eth0发往主机的网关，接着包会转发给eth0，并从eth0发出去。
  - 数据包相当于做SNAT转换，将源地址换为eth0的地址。这样，在外界看来，这个包就是从eth0上发出来的，Docker容器对外是不可见的

- 向内通信
  - 主机eth0收到的目的端口为port1的数据流量进行DNAT转换，将数据流量发往对应容器的port2端口

网络|NAT桥接

- NAT桥接适用场景
  - 桥接模式是Docker默认的网络连接方式，几乎适用于所有的应用
  - 桥接模式有网络性能的损耗，不适用于对网络性能要求较高的应用（比如nginx）
  - NAT 机制导致无法使用容器 IP 进行跨服务器通讯
  - 若容器需要对外提供服务，桥接模式需要在宿主机之上开放服务端口，这在部署上势必带来一些不便，例如容器无法漂移，NAT模式难于理解等等
  - 适用于某些特殊场景，例如桥接与HOST模式配合，来搭建对信源、输出有需求，且环境长期不会变的的场景

如何使用docker

- 创建镜像
- 创建容器
- 在需要时暴露端口，创造卷
- 通过链接将几个容器连接在一起
- 还有更高级的应用，比如创建网桥自行组网等，请参考手册

Let’s do it！

实际体验一下吧

启动一个Nginx

docker run –d --name nginx001 -p 80:80 nginx

启动一个Nginx

怎么在宿主机直接修改容器内部文件？

$ mkdir –p /data/docker/nginx/html

$ vim index.html

$ docker run --name nginx002 -p 90:80 -v /data/docker/nginx/html:/usr/share/nginx/html:ro -d nginx

DockerFile-容器定制

- Dockerfile是一个文本格式的配置文件，用户可以使用Dockerfile快速创建自定义镜像。

- Dockerfile由一行行的命令语句组成。并且支持以#开头的注释行。一般Dockerfile分为四个部分基础镜像信息、维护者信息、镜像操作指令和容器启动时的指令

Docker File

FROM	• 这个镜像的妈妈是谁？（指定基础镜像）
MAINTAINER	• 告诉别人，谁负责养它？（指定维护者信息）
RUN	• 你想让它干啥（在命令前面加上RUN即可）
ADD	• 给它点创业资金（COPY文件，会自动解压）
WORKDIR	• 我是cd,今天刚化了妆（设置当前工作目录）
VOLUME	• 给它一个存放行李的地方（设置卷，挂载主机目录）
EXPOSE	• 它要打开
CMD	• 奔跑吧，兄弟！（指定容器启动后的要干的事情）

DockerFile-自定义容器

```
FROM alpine:3.8
MAINTAINER tianccm <tianccm@gmail.com>
RUN echo "https://mirror.tuna.tsinghua.edu.cn/alpine/v3.8/main/" > /etc/apk/repositories
RUN apk update \
&& apk upgrade \
&& apk add --no-cache bash bash-doc bash-completion \
&& rm -rf /var/cache/apk/* \
&& /bin/bash
```

Docker-compose 容器编排

- 下载**docker-compose**
- $ sudo curl -L https://github.com/docker/compose/releases/download/1.17.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

- 授权
- $ sudo chmod +x /usr/local/bin/docker-compose

- 查看版本信息
- $ docker-compose --version

Docker-compose 容器编排

```
version: '3'
services:
  mysql:
    image: mysql:5.7.23
    volumes:
      - db-data:/var/lib/mysql
    ports:
      - 3308:3306
    environment:
      - MYSQL_ROOT_PASSWORD=caiqiu502
  wordpress:
    image: wordpress
    ports:
      - "80:80"
    environment:
      - WORDPRESS_DB_HOST=mysql
      - WORDPRESS_DB_USER=root
      - WORDPRESS_DB_PASSWORD=caiqiu502
volumes:
  db-data:
```

Docker-应用方向

- 1、简化配置

  应用配置能够无缝运行在任何平台，将应用环境和底层环境实现了解耦

- 2、代码管道化管理

  代码从开发者的机器到生产环境机器进行管道化管理，能够平滑迁移。

- 3、应用隔离

  多个应用服务部署在多个Docker中，实现应用之间的解耦

- 4、服务合并

  合并多个服务，减少机器占用

Docker-应用方向

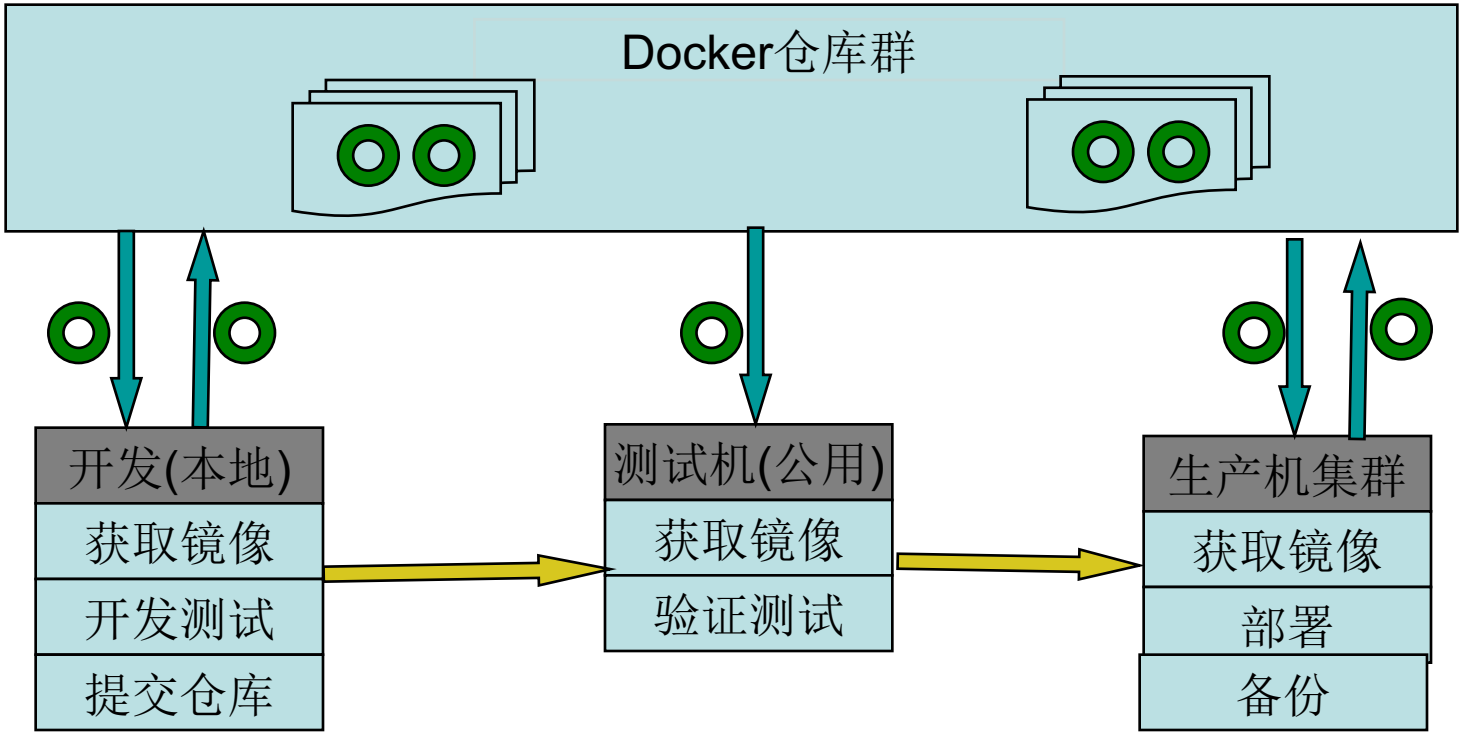
- 5、快速部署

快速的启动速度，极小空间占用

- 6、开发环境

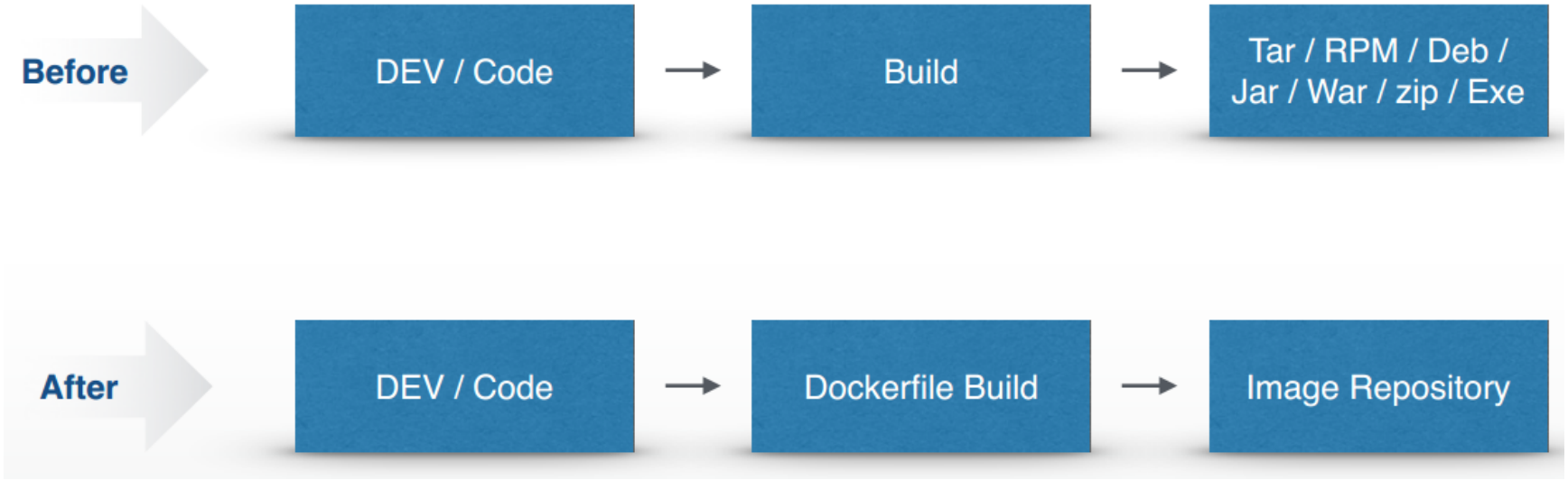
能够在单机上搭建分布式集群服务，用以测试在真正生产环境下的代码

以Docker为单位的开发部署流程设计



- 以docker为单位的开发测试部署流程,简化了环境搭建的步骤,提高了资源利用效率和开发测试部署的速度,降低了迁移的成本

以Docker为单位的开发部署流程设计



思考题

1、 Docker使用的都不是新技术，为什么现在变得流行了？

2、哪些环境下适合使用Docker技术，哪些情况下不适合？为什么？

3、思考一下，工作中哪些步骤可以用docker技术和思想来提高效率和稳定性？

What's the NEXT?

1、Volume详解、容器数据持久化

2、Dockerfile实战

3、微服务Docker-Compose实战

4、一些Docker的高级话题
系统隔离、网络、文件系统等

The End