Pivotal

Docker入门
容器系列

June 2018

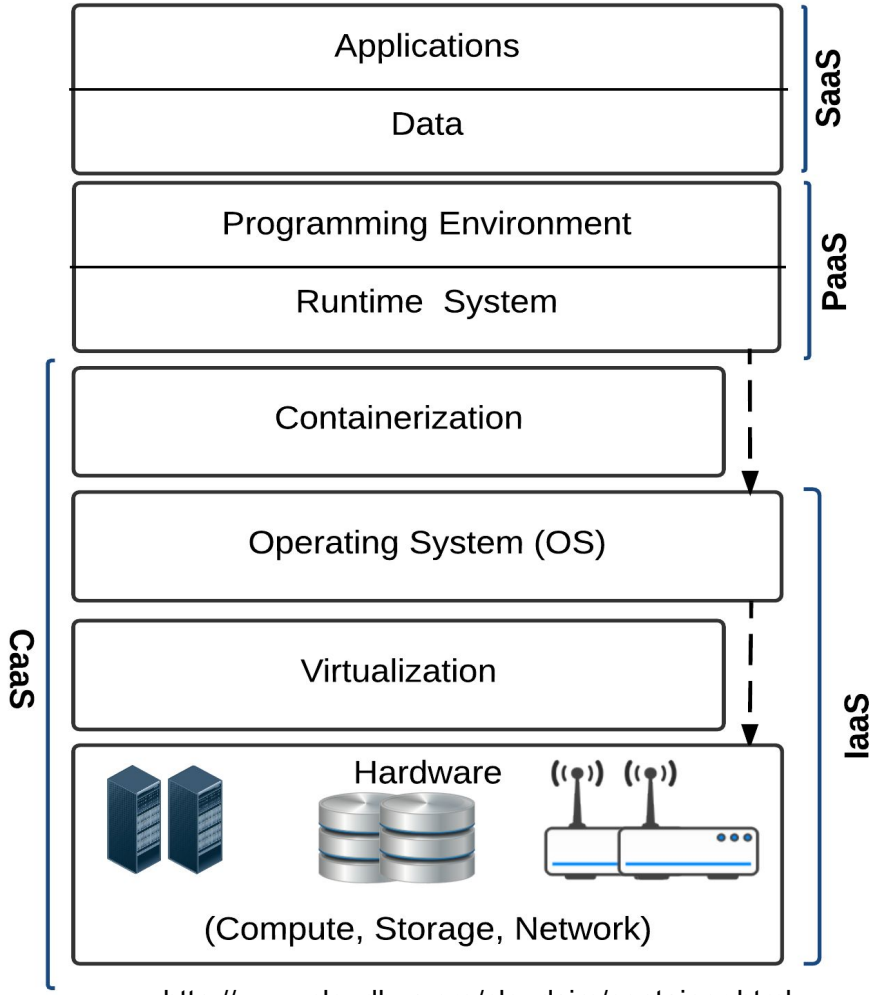
主要议题

docker及容器相关技术背景的概括介绍和操作实践

- 容器技术概述
- Docker介绍
- Docker操作实战
- Docker运行Greenplum

容器技术概述

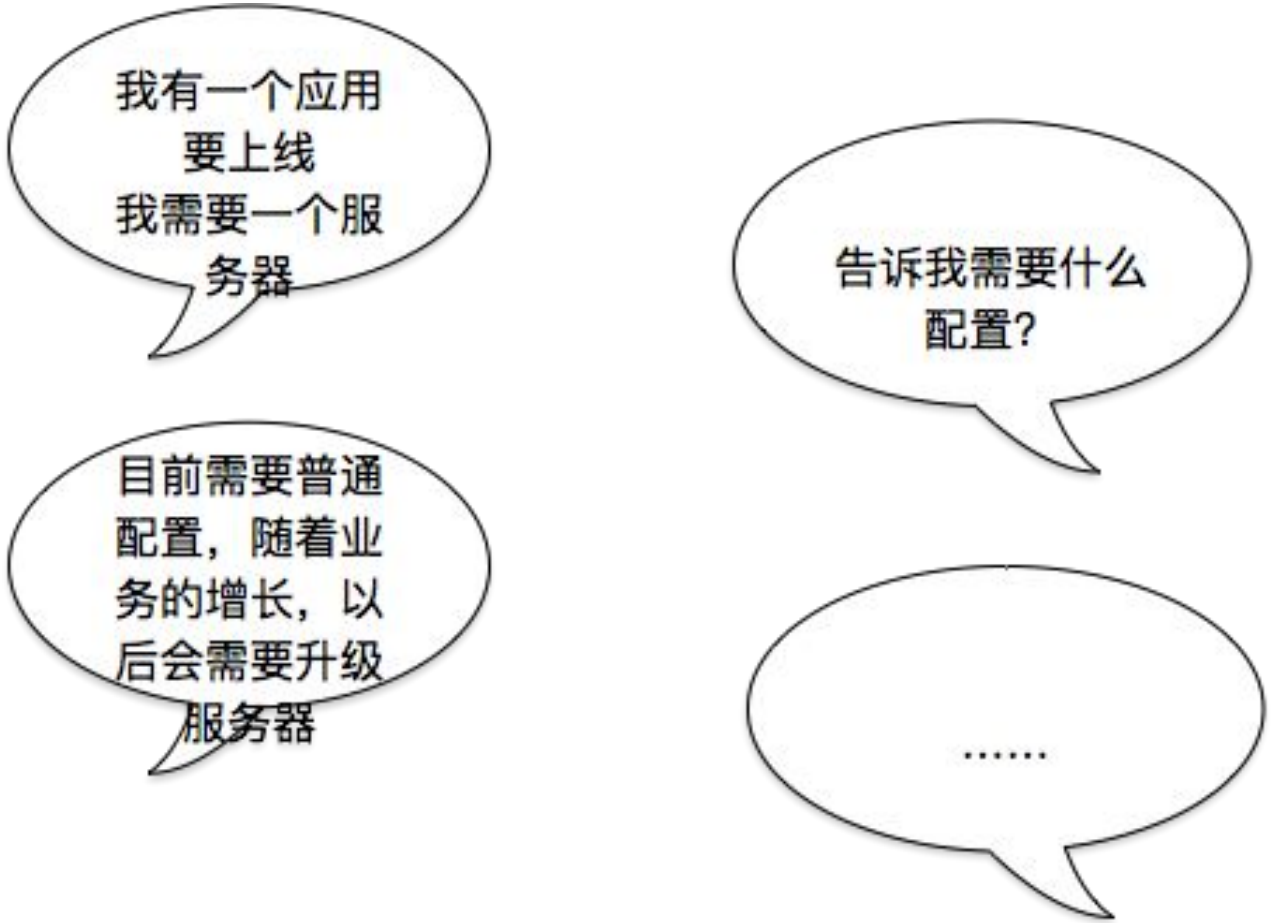
- 虚拟化的产生
- 虚拟机和容器的区别
- 容器的历史



http://www.cloudbus.org/cloudsim/container.html

小明的故事

小明是IT部门的系统管理员，有一天... ...



小明的故事

第二天.

Pivotal

虚拟化的产生

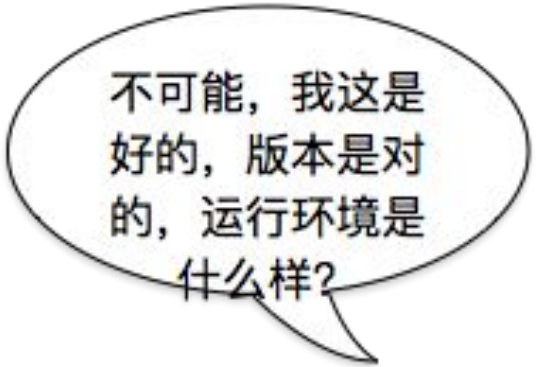
共享同一主机的资源的问题

- 运行环境相互影响
- 资源竞争
- 安全风险增大
- 维护成本高

通过虚拟化，实现不同应用的隔离，同时大大降低管理成本

小明的故事

后来

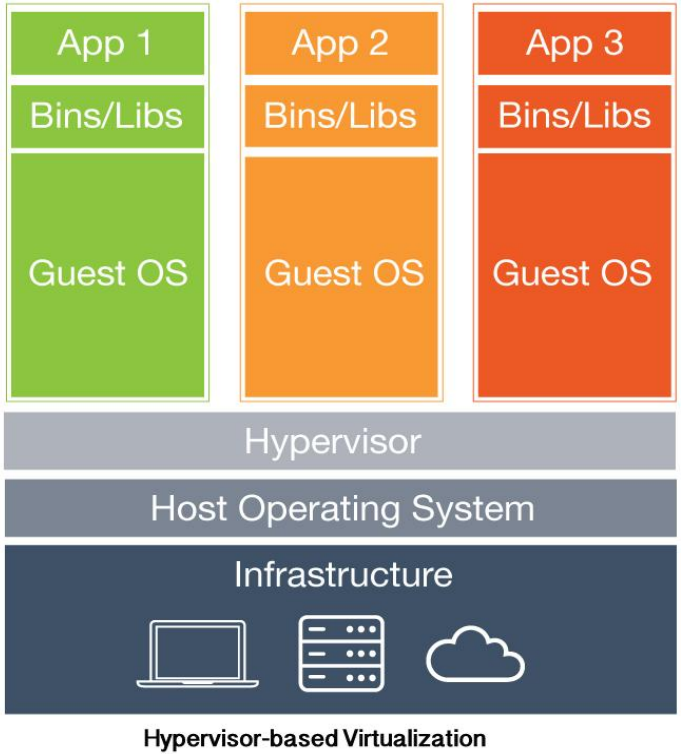


虚拟化的产生

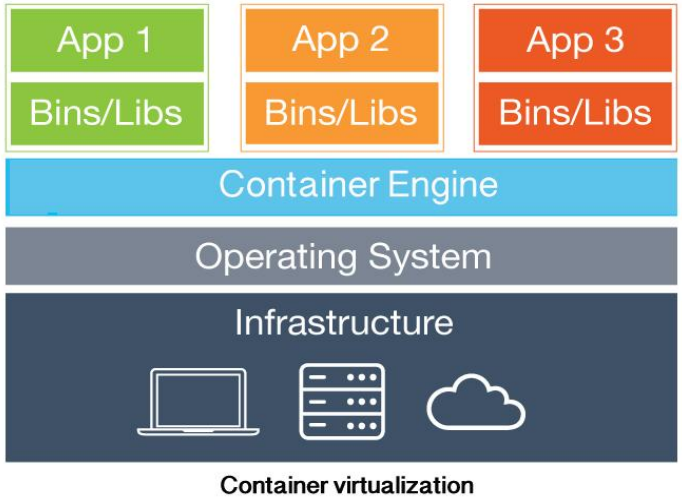
虚拟化的好处

- 应用隔离
  - 环境
  - 资源
  - 安全
- 易于管理
  - 按需扩展
  - 不依赖底层设备

虚拟机和容器的区别



Hypervisor-based Virtualization



Container virtualization

Pivotal

https://www.blackmagicboxes.com/?p=349

虚拟机和容器的区别

虚拟机

- 虚拟出一台主机
- 需要独立安装操作系统
- 虚拟机需要从模拟BIOS开始（分钟级）
- 升级依靠操作系统更新
- 大规模部署需要依赖专门工具
- 宿主机资源占用高

容器

- 虚拟应用的运行环境
- 与宿主服务器共享操作系统
- 容器以进程方式执行（秒级）
- 升级通过制作新的镜像
- 大规模部署迅速方便
- 宿主机资源占用低

容器技术的历史

1979: Unix V7

Unix V7引入了chroot，开启了虚拟系统的大门。chroot可以为进程的指定另外的root路径，从而为其子进程隔离一个独立的运行环境，从而开启了进程隔离的新进程。Chroot于1982年移植到BSD系统

容器技术的历史

2000: FreeBSD Jails

2000年，BSD系统提供了名为jails的隔离环境，它的初衷主要是安全因素。它运行将主机隔离为几个小的系统，并可以为每个子系统分配独立的IP地址。

容器技术的历史

2001: Linux Vserverr

Vserver提供了Linux平台上的Jail机制，它通过内核打补丁的方式实现了文件系统，网络，内存等子系统的隔离。最新的稳定版补丁发布于2006年

容器技术的历史

2002: Linux Namespace

Linux Namespace是Linux内核的一个模块，它可以用来将内核资源分配给不同的进程，从而将资源隔离出不同的空间。它是Linux容器中实现资源隔离的重要部分。

容器技术的历史

2004: Oracle Solaris Containers

2004年Oracle发布了Solaris的容器，它通过Zone(系统中独立运行相互隔离的服务进程)，实现系统资源的隔离和快照功能

Pivotal

容器技术的历史

2005: Open VZ

与Vserver类似的，通过内核补丁的方式实现资源隔离的容器解决方案。至今仍在某些VPS系统中使用

容器技术的历史

2006: Process Containers

由Google在2006年发布的，用于限制系统资源使用（CPU，内存，磁盘IO，网络）的内核模块。一年之后改名为cgroup。

容器技术的历史

2008: LXC

LXC(LinuX Containers)是第一个完整功能的Linux容器管理器，它依赖cgroup和namespace和原生的Linux系统内核

容器技术的历史

2011: Warden

Warden是Cloudfoundry在2011年推出的容器管理器。最开始使用LXC后来实现了自己的容器管理方案。后来用go重新实现，成为今天的garden。

容器技术的历史

2013: Docker

Docker出现于2013年，最初是dotcloud工程的一部分。它最初也是使用LXC来控制容器运行，在0.9版本替换成自己的libcontainer

容器技术的历史

2015: Open Container Initiative

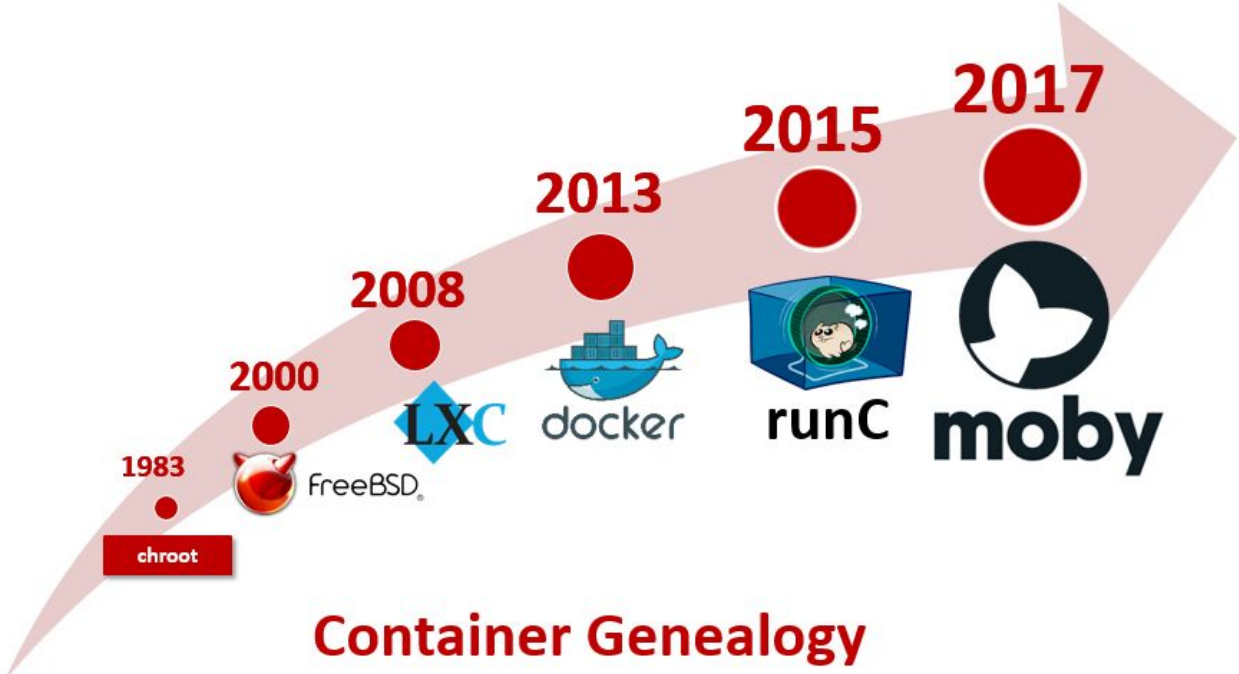
Docker与CoreOS等其它公司一起成立的标准组织，定义了容器文件格式（image）和运行环境（run）的标准。

容器技术的历史

2017: moby

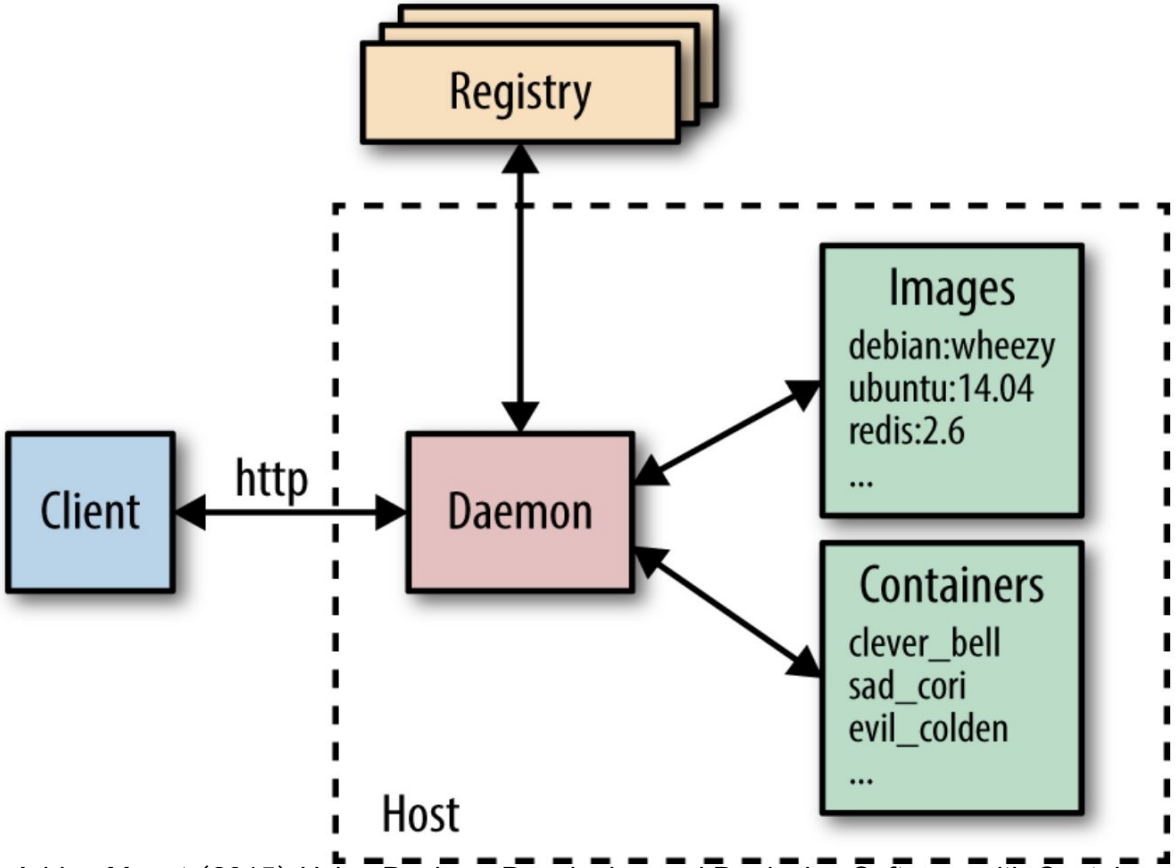
Docker的核心源码更名为moby，它提供了开放的容器管理框架，可以集成各个模块，使得任何人都可以组装自己的容器管理方案

容器技术的历史



https://techblog.rakuten.co.jp/2017/07/26/moby-and-linuxkit/

Docker的系统架构



Pivotal

Adrian Mouat.（2015）Using Docker - Developing and Deploying Software with Containers. O'Reilly Media

Docker的关键技术

Docker基础设施建立在三个最主要的基础设施上

- 隔离系统资源的Linux namespace
- 限制资源使用的cgroup
- 叠加文件系统unionfs

Docker的主要优势

- 操作简便，用户体验好
- 性能损失小
- 可以简单方便的自定义镜像
- 方便的镜像管理
- Dockerhub上丰富的镜像资源
- 便于集成现代软件开发方式
- 开放的社区和生态接口

Docker registry

用于保存和发布Docker镜像的服务

- 默认为dockerhub，可以指定其他服务
- 支持共开和私有的镜像仓库
- 与docker镜像管理命令深度集成

镜像 vs 容器

镜像 => 程序

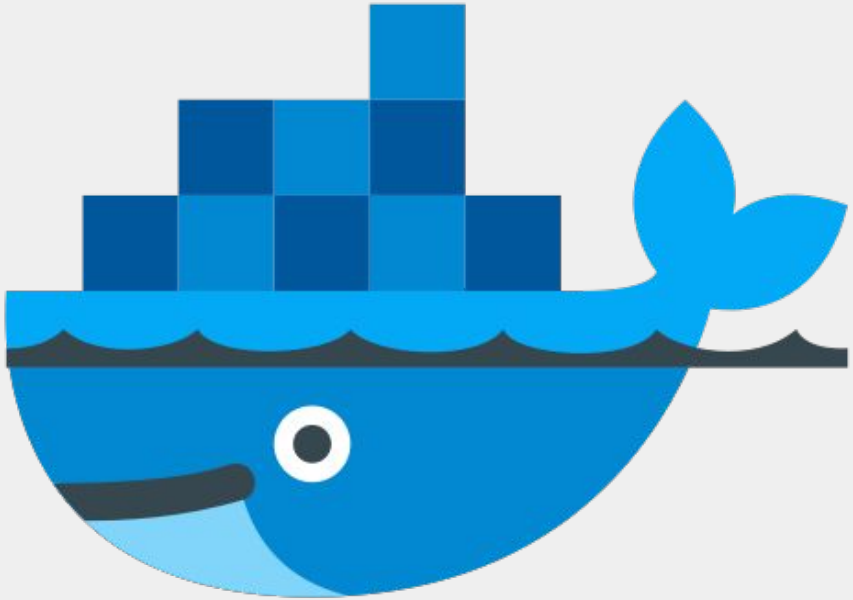
- 静态
- 磁盘上的一个文件
- 运行时所有的依赖文件组成的快照
- 同一个镜像可以运行在多个容器中
- 通过docker images查看

容器 => 进程

- 动态
- 正在运行的一个/一组进程
- 运行镜像的虚拟环境
- 每个容器运行一个特定的镜像
- 通过docker ps查看

Docker实战

- 运行容器
- 管理容器
- 管理镜像
- 制作自己的镜像
- docker-compose



Pivotal

http://www.cloudbus.org/cloudsim/container.html

Hello, world

docker **run alpine /bin/echo hello, world**

查看docker信息

docker info

docker version

运行容器

docker run **-i -t --rm --name** my container
ubuntu:latest /bin/bash
docker run **-d** --name myhttpserver **-p** 80:80 **-v**
/some/content:/usr/share/nginx/html:ro nginx:alpine

docker run -v `pwd`/config:/etc/nginx/conf.d/default.conf:ro -v
/Users/jasper/share:/usr/share/nginx/html:ro -p 80:80 -d nginx

容器操作

docker run

docker start/stop

docker rm

docker kill

docker export

容器状态

docker ps [-a] [-l] [-a]

docker inspect

docker top

镜像操作

docker images

docker pull

docker push

docker images

docker rmi

docker history

docker build

docker import

实战：容器中运行Postgres

docker run **--name** mypg **-e**
POSTGRES_PASSWORD=mysecretpassword **-d postgres**

容器中访问Postgres

docker **exec** -it mypg psql -h localhost -U postgres

其它容器中使用Postgres

docker run **--link mypg:postgres** -d some_app

其它容器中通过命令行访问Postgres

docker run --rm -it **--link mypg:postgres** postgres psql -h postgres -U postgres

通过dockerfile创建镜像

Docker build . -t name:tag

Dockerfile介绍

FROM：基础镜像
RUN：创建镜像时需要运行的命令
CMD：指定运行镜像时执行的命令
EXPOSE：容器运行时可以导出的端口
ENV：容器运行使用的环境变量
ADD：像容器中添加文件
ENTRYPOINT：指定执行镜像时使用的命令

Dockerfile介绍

RUN vs CMD vs ENTRYPOINT

- Run命令生成新的Image文件系统，在docker build时运行
- CMD指定默认执行的命令，可被docker run重载
- ENTRYPOINT指定的命令从docker run中读取额外参数

Dockerfile示例

```
FROM ubuntu:16.04
ENTRYPOINT ["/bin/echo"]
```

用docker编译Greenplum

```
# cd ~/meetup
# docker run --rm -u gpadmin -it  -v `pwd`/gpdb:/home/gpadmin/gpdb -v
`pwd`/greenplum-db:/usr/local/gpdb lyasper/gpdev  /bin/bash
./configure --disable-orca --without-perl --with-python --with-libxml
--without-gssapi --disable-pxf
# make && make install
# exit
# docker build . -t mygreenplum
```

docker运行Greenplum

https://github.com/lij55/gphost/tree/oss

Transforming How The World Builds Software

加入技术微信讨论群组

活动群组专用号

扫一扫上面的二维码图案，加我微信

欢迎大家加入技术微信讨论群组！

- Greenplum官方技术二群
- Cloud Foundry 官方技术讨论群

微信扫一扫二维码加为好友，我们将把您邀请进群组

Pivotal