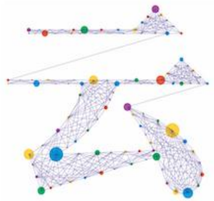
EasyStack
open cloud computing



第四届
全国高校
云计算
应用创新大赛

全国高校云计算应用创新大赛
官方培训班在线课程

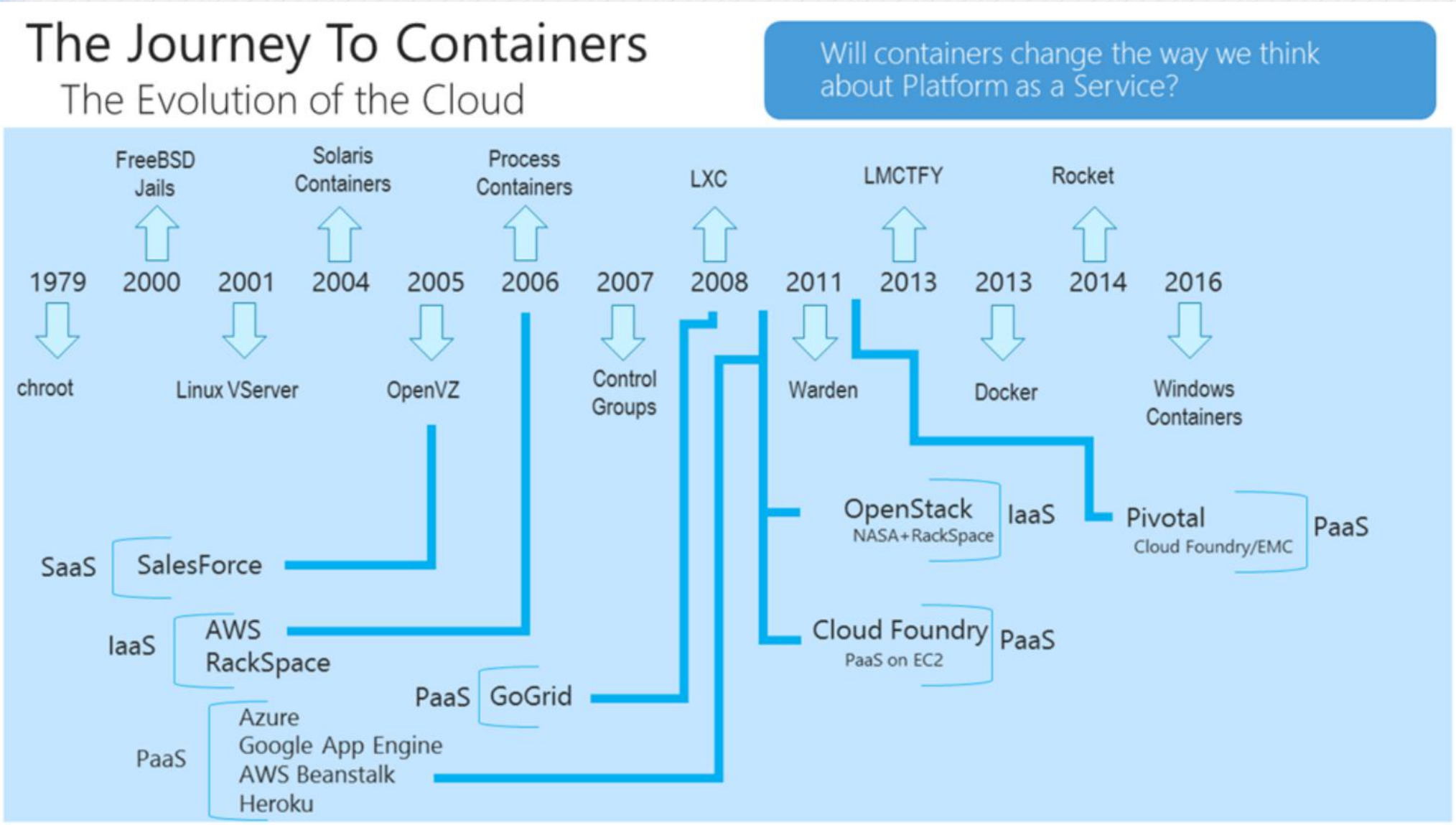
容器基础

胡正川

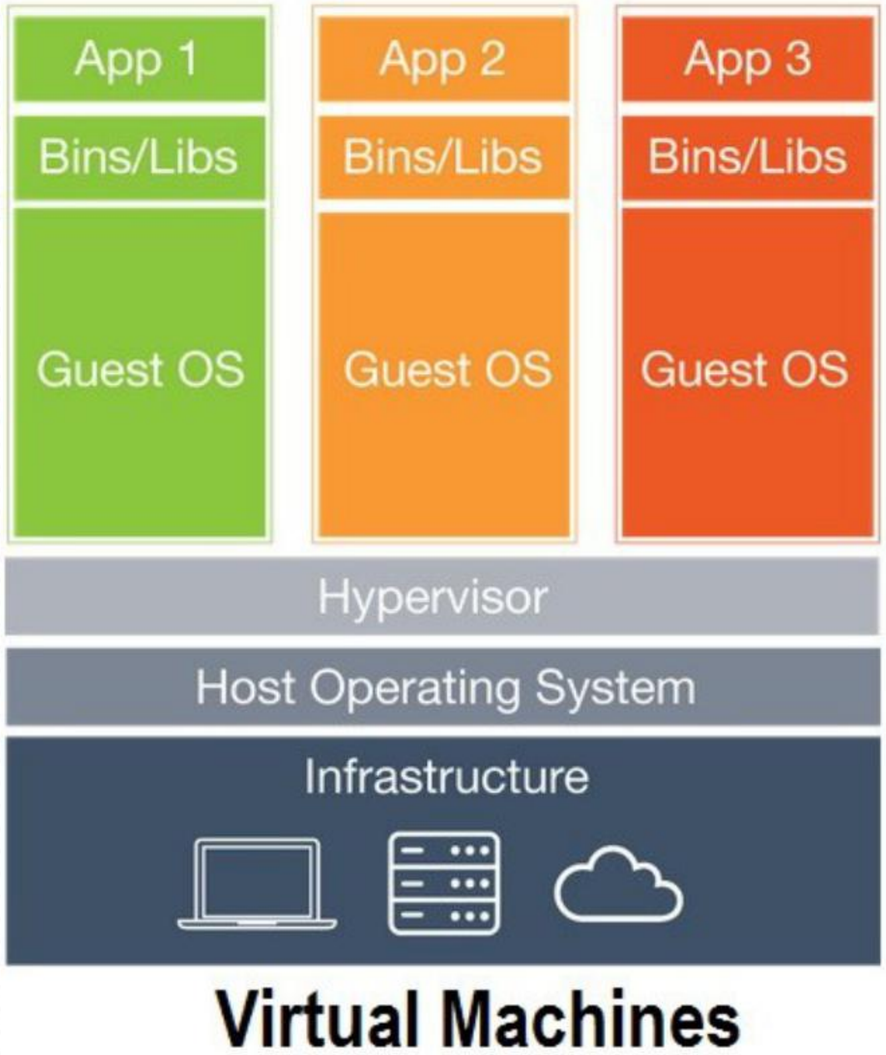
EasyStack研发工程师

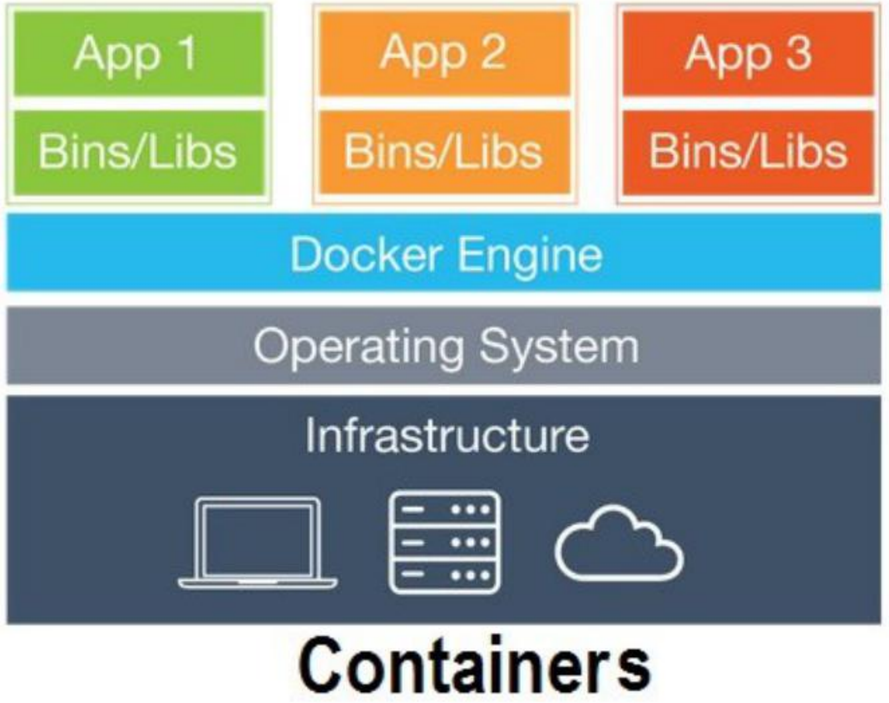
目录

容器


The Journey To Containers
The Evolution of the Cloud
Will containers change the way we think about Platform as a Service?
FreeBSD Jails
Solaris Containers
Process Containers
LXC
LMCTFY
Rocket
1979
2000
2001
2004
2005
2006
2007
2008
2011
2013
2013
2014
2016
chroot
Linux VServer
OpenVZ
Control Groups
Warden
Docker
Windows Containers
SaaS
SalesForce
IaaS
AWS
RackSpace
PaaS
GoGrid
PaaS
Azure
Google App Engine
AWS Beanstalk
Heroku
OpenStack
NASA+RackSpace
IaaS
Pivotal
Cloud Foundry/EMC
PaaS
Cloud Foundry
PaaS on EC2
PaaS

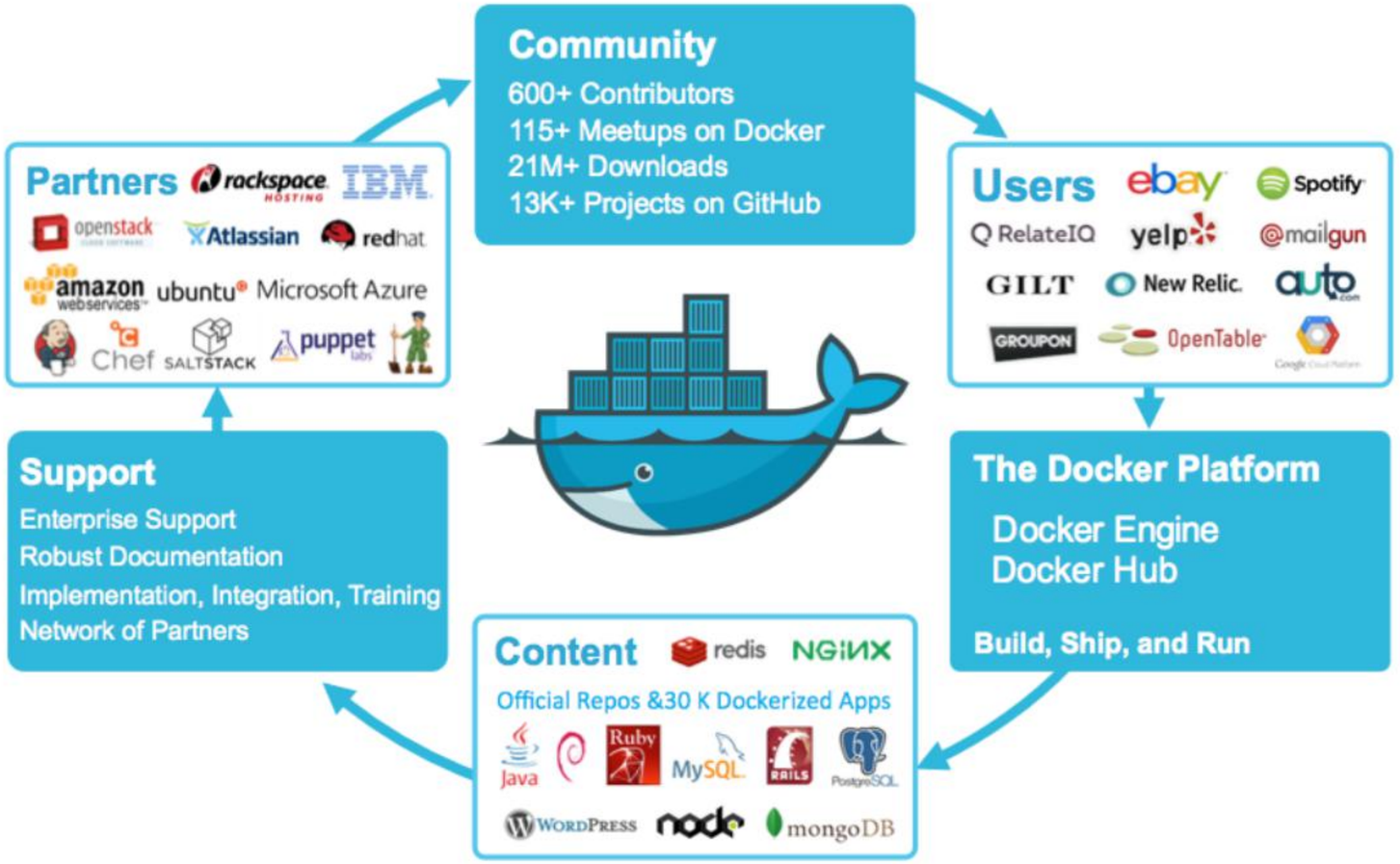
容器


App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure
Virtual Machines


App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Engine
Operating System
Infrastructure
Containers

容器

“Container technology, and Docker in particular, is hotter than hot.”–Steven J. Vaughan/ Nichols, Network World, January 14, 2015



Docker基础

namespace/cgroups

容器是依托于Linux的namespace/cgroups而运行的正常的系统进程.

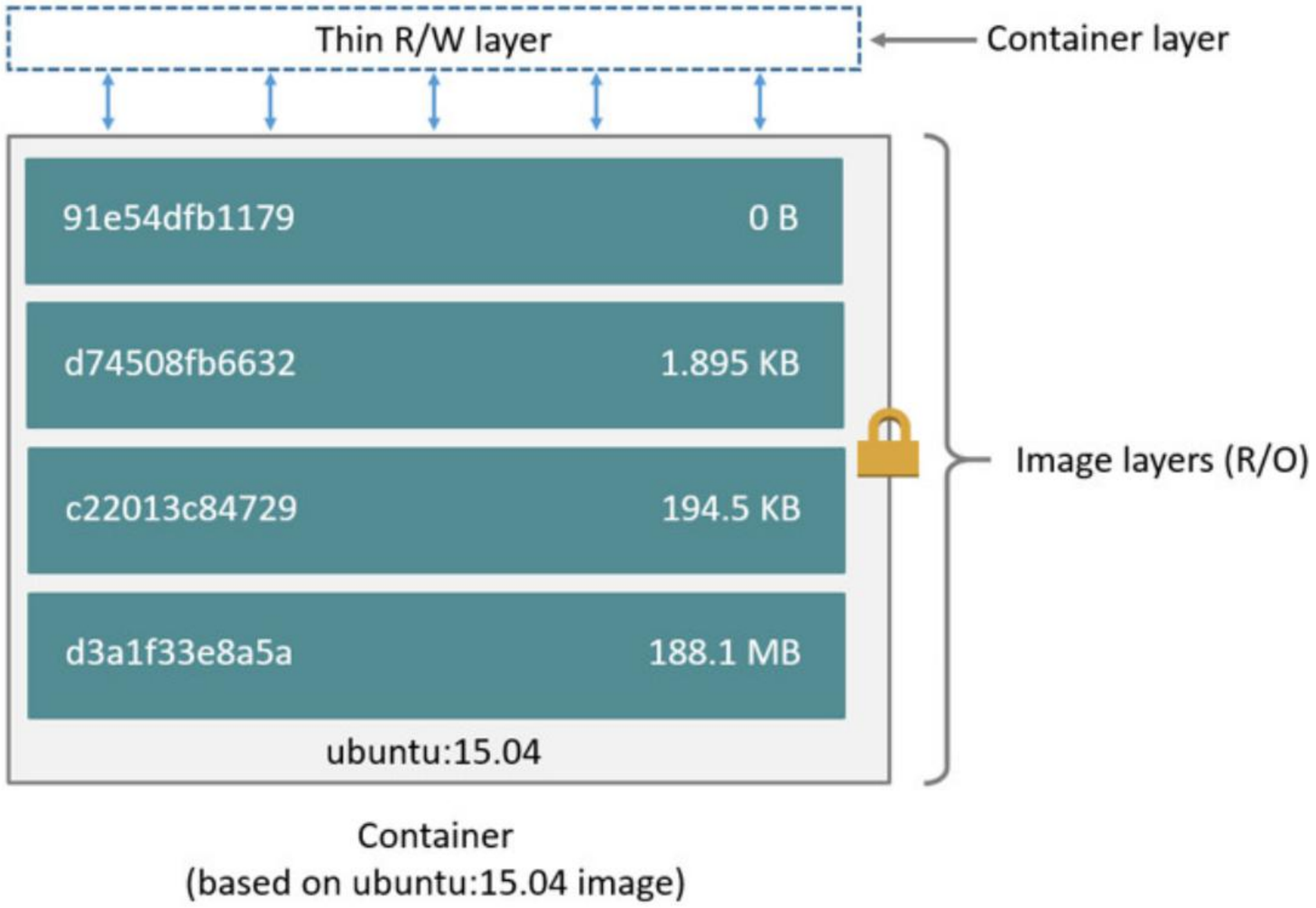
namespace
命名空间提供对于某个进程的一种“视图”，可以将其它命名空间的东西隐藏，从而为该进程提供它自己独立运行的环境。这使得进程之间互相不可见，也不可相互影响,命名空间包括如下：

- 主机名
- 进程ID（PID）
- 文件系统
- 网络接口
- 进程间通信（IPC）

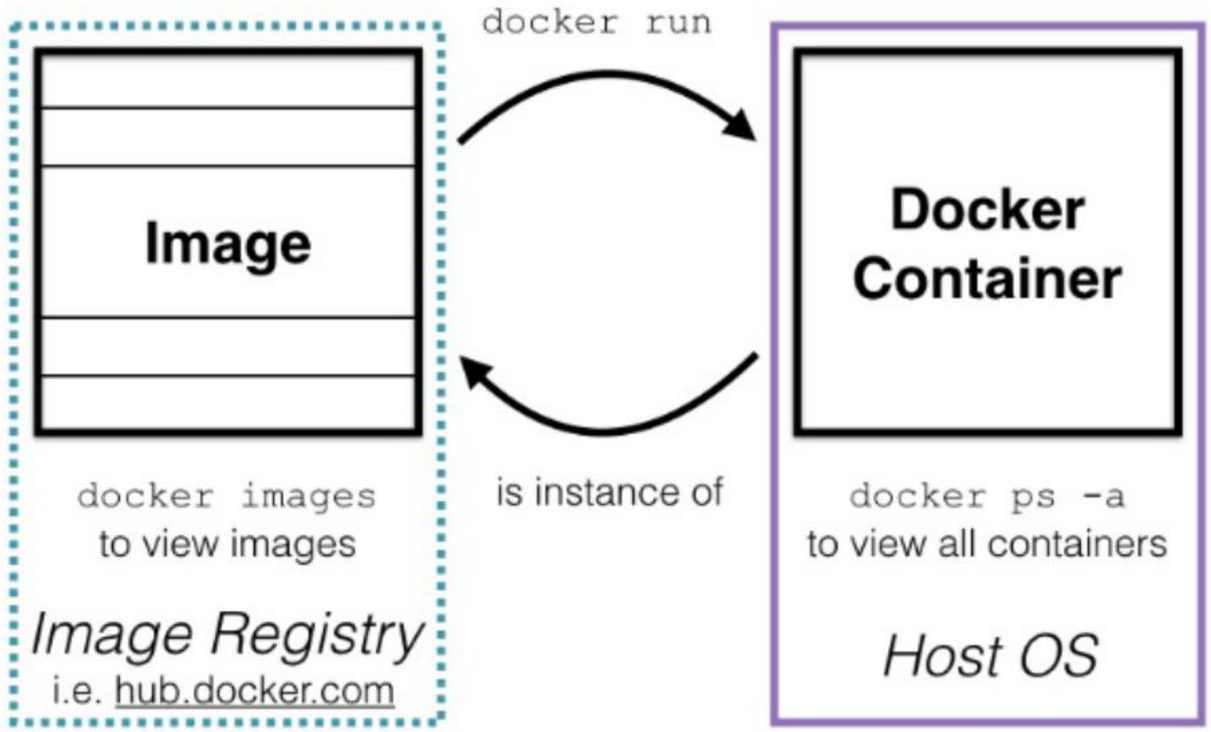
namespace/cgroups

Cgruops
cgroup限制了该进程可用的资源。所谓的“资源”包括CPU、内存、磁盘I/O、网络I/O等。CPU的分配可以以千分之一核为单位，内存的分配可以以字节（bytes）为单位。该进程可以正常运行，但是它只可以使用cgroup允许的CPU能力，如果所使用的内存超出cgroup设置的限制就会内存溢出。

Image


Thin R/W layer
Container layer
91e54dfb1179
0 B
d74508fb6632
1.895 KB
c22013c84729
194.5 KB
d3a1f33e8a5a
188.1 MB
ubuntu:15.04
Image layers (R/O)
Container
(based on ubuntu:15.04 image)

Container


docker run
Image
Docker Container
docker images
to view images
is instance of
docker ps -a
to view all containers
Image Registry
i.e. hub.docker.com
Host OS

Docker基础

镜像制作

Dockerfile文件

docker build ./

docker命令

docker run image

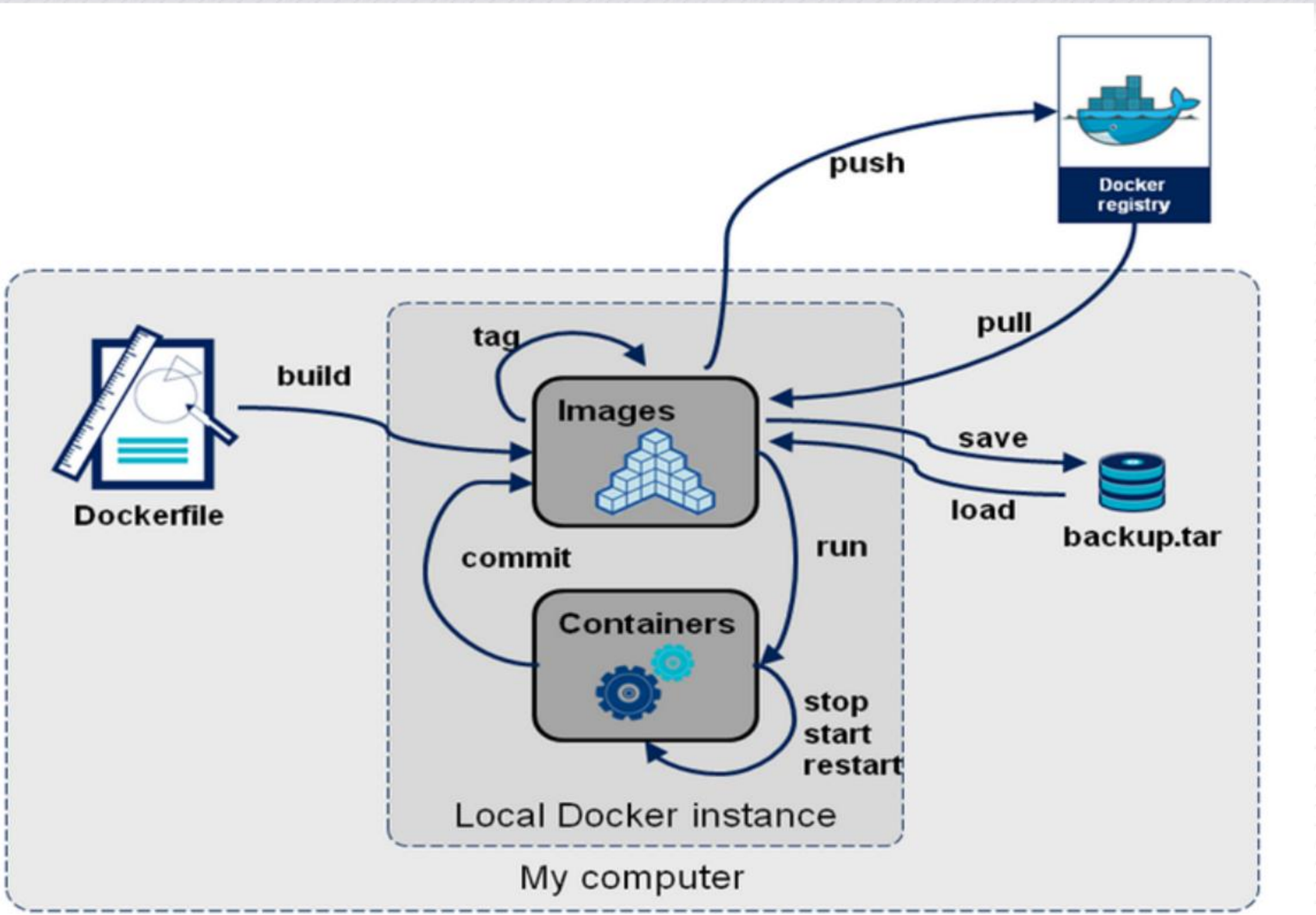
modify

docker commit

docker save new

docker tag

docker pull



Docker基础

Dockerfile

Dockerfile是一个镜像的表示，可以通过Dockerfile来描述构建镜像的步骤，并自动构建一个容器镜像.

常用命令：

- FROM
- RUN
- EXPOSE
- CMD
- ENTRYPOINT

```
FROM ubuntu

MAINTAINER Michael Crosby <michael@crosbymichael.com>

RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list
RUN apt-get update
RUN apt-get upgrade -y

RUN apt-get install -y python-software-properties
RUN add-apt-repository ppa:rethinkdb/ppa
RUN apt-get update
RUN apt-get install -y rethinkdb

# Rethinkdb process
EXPOSE 28015
# Rethinkdb admin console
EXPOSE 8080

# Create the /rethinkdb_data dir structure
RUN /usr/bin/rethinkdb create

ENTRYPOINT ["/usr/bin/rethinkdb"]

CMD ["--help"]
```

https://docs.docker.com/engine/reference/builder/#usage

Docker CLI

命令	
% docker run	
% docker start/stop	
% docker attach	
% docker build	
% docker commit	
% docker cp	
% docker create	
% docker logs	
% docker image	
% docker ps	

https://docs.docker.com/engine/reference/commandline/cli/

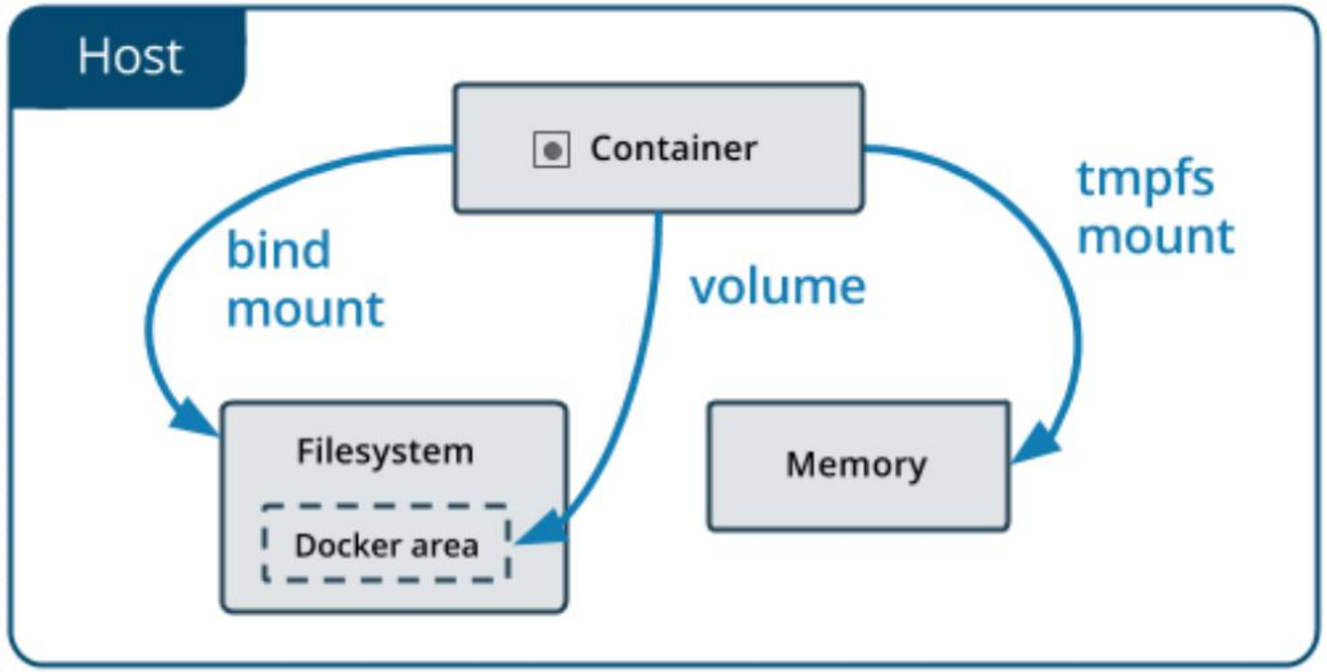
Docker存储

容器数据直接存储在容器的读写层存在问题？

- 数据无法持久化
- 数据无法在容器之间共享

类型

- Volumes
- bind mounts
- tmpfs mounts

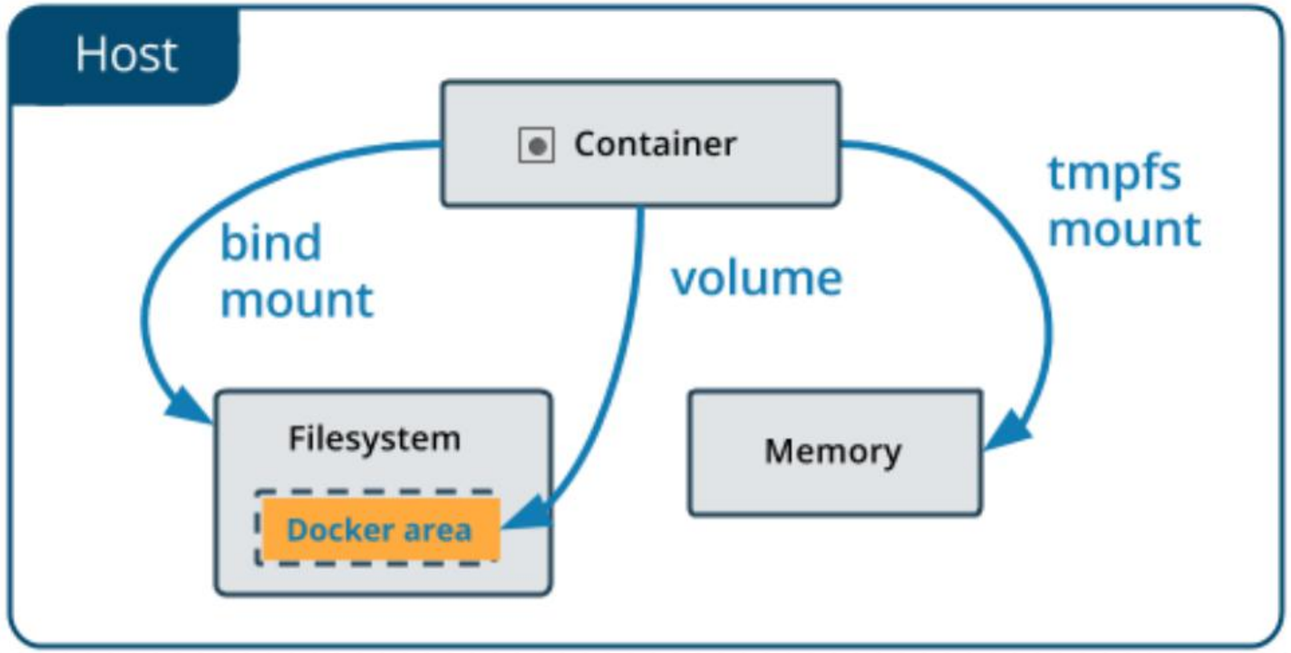


Docker存储

volumes

使用场景

- 在多个容器之间共享数据
- 存储数据到云端或者是远端
- 数据需要备份，恢复或者迁移到其他的Docker主机上



demo

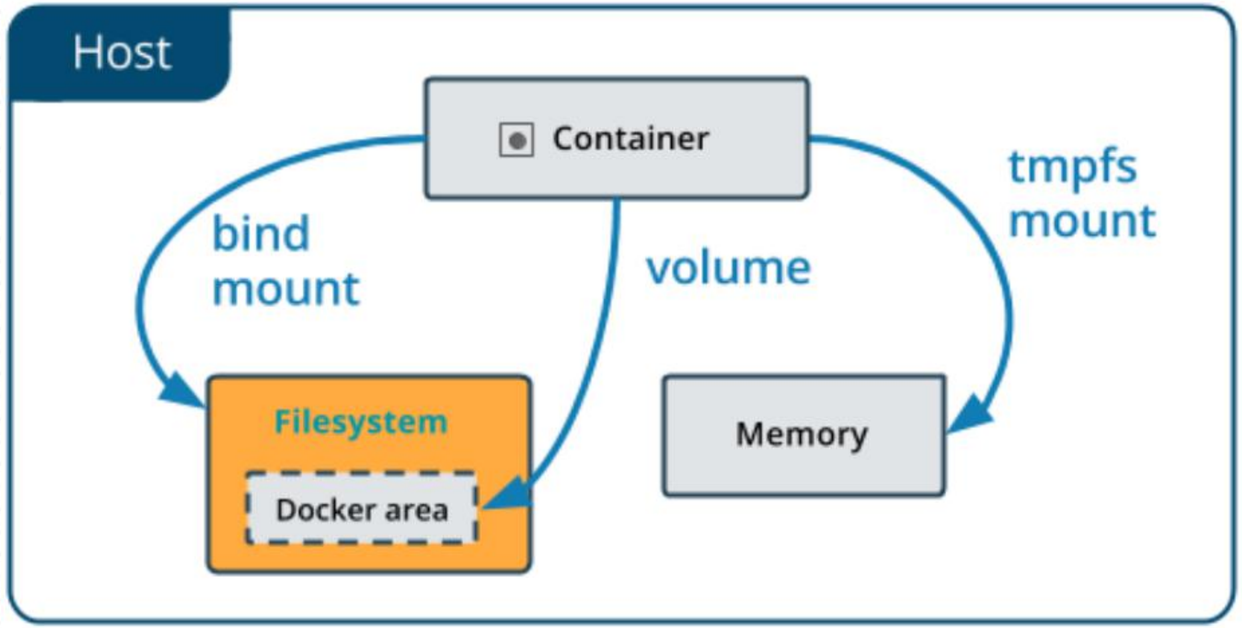
```
docker volume create myvol
docker run / d / it / / name devtest / / mount source=myvol,target=/app nginx:latest
```

Docker存储

bind mounts

使用场景

- 共享主机的数据或者文件到容器中，比如/etc/resolv.conf
- 在Docker主机和容器之间共享代码或者建立artifacts，用于编译环境的构建



Demo

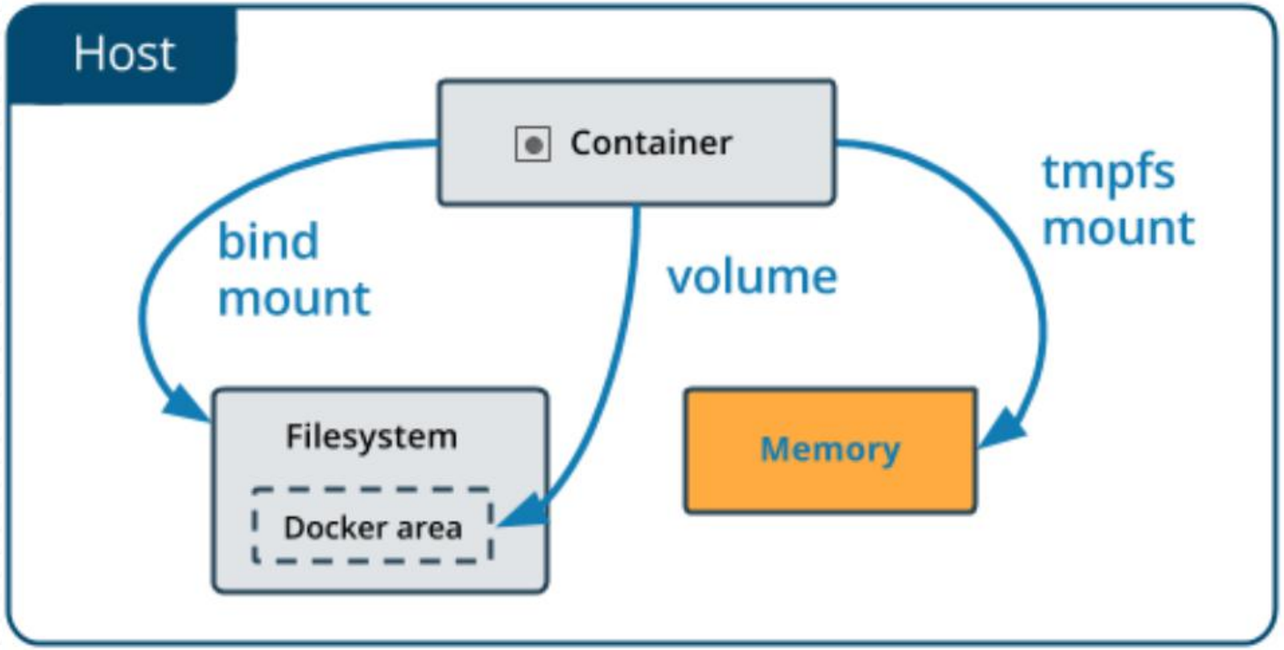
```
docker run / d / it / / name devtest / / mount type=bind,source=‘%(pwd)‘/target,target=/app nginx:latest
```

Docker存储

tmpfs mounts

使用场景

- 数据并不需要进行持久话存储
- 敏感数据



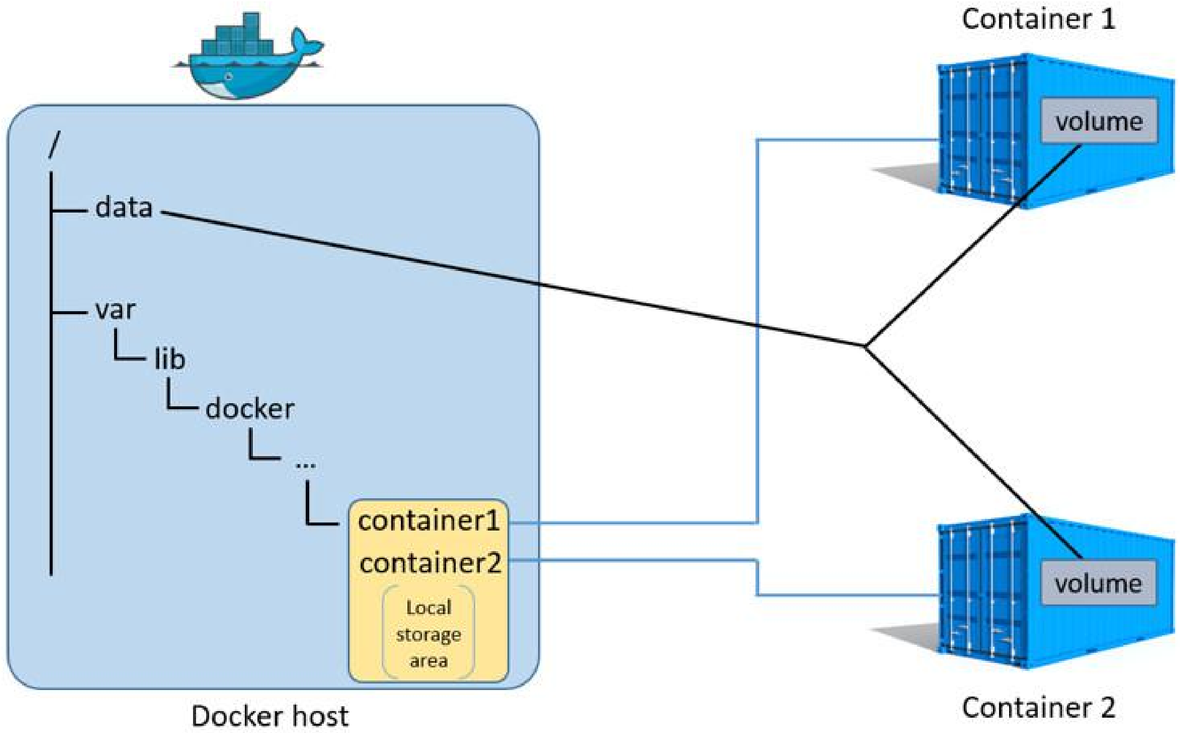
Demo

docker run / d / it / / name tmptest / / mount type=tmpfs,destination=/app nginx:latest

Docker存储

Docker Storage Drivers

volume主要解决的问题是持久化存储以及容器之间数据的共享. Docker使用不同的存储驱动来管理image和运行的容器中的文件系统.
实现层叠文件系统



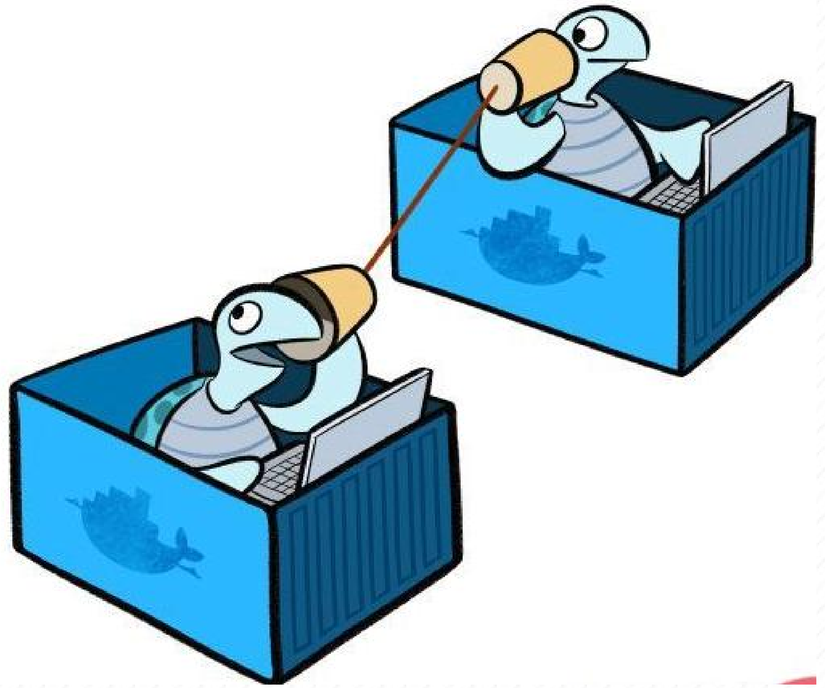
支持的driver
overlay, overlay2, aufs, devicemapper, zfs, vfs

https://docs.docker.com/engine/userguide/storagedriver/

Docker网络

需要解决的问题？

1. 容器本身的网络配置
2. 通信
    1. 同一个节点上容器通信
    2. 跨节点的容器通信



Docker网络

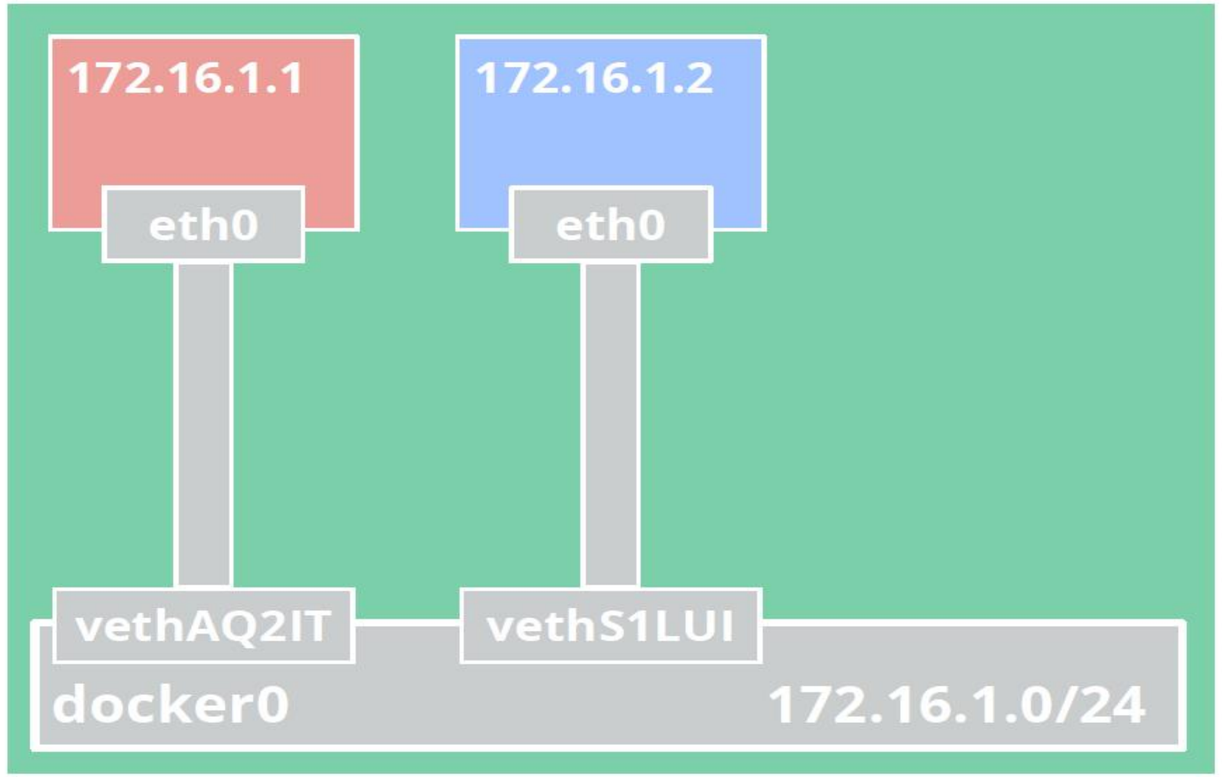
Docker容器的网络模式

- bridge模式
- host模式
- container模式
- none模式

Docker网络

bridge模式

docker默认采用veth的方式将container中的虚拟网卡同host上的一个docker bridge—docker0连接在一起。



Docker网络

host模式

docker run的时候指定—net=host, 不为docker容器创建网络协议栈，不会创建独立的network namespace.使用宿主机的网卡，IP和端口等信息，降低了网络方面的隔离性，会引起网络资源的竞争和冲突

container模式

指定新创建的容器和已经存在的某个容器共享同一个network namespace.两个容器的通过lo回环网卡进行通信

none模式

容器拥有自己的network namespace, 但是不为容器进行任何其他的网络配置，配置留给用户自己，灵活性更强。

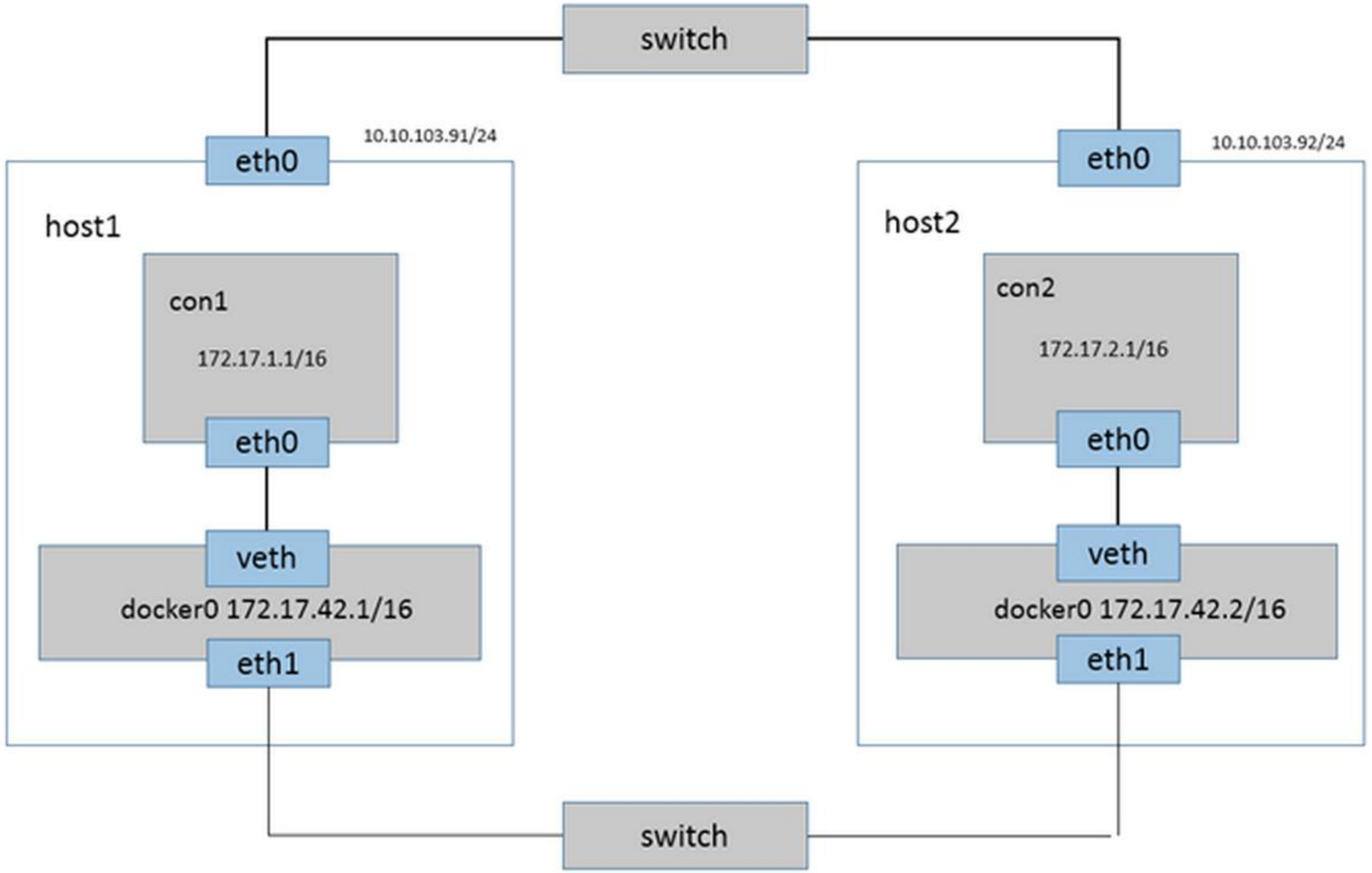
Docker网络

跨主机容器网络

- 大二层
- NAT方式
- Tunnel(overylay)方式
- ...

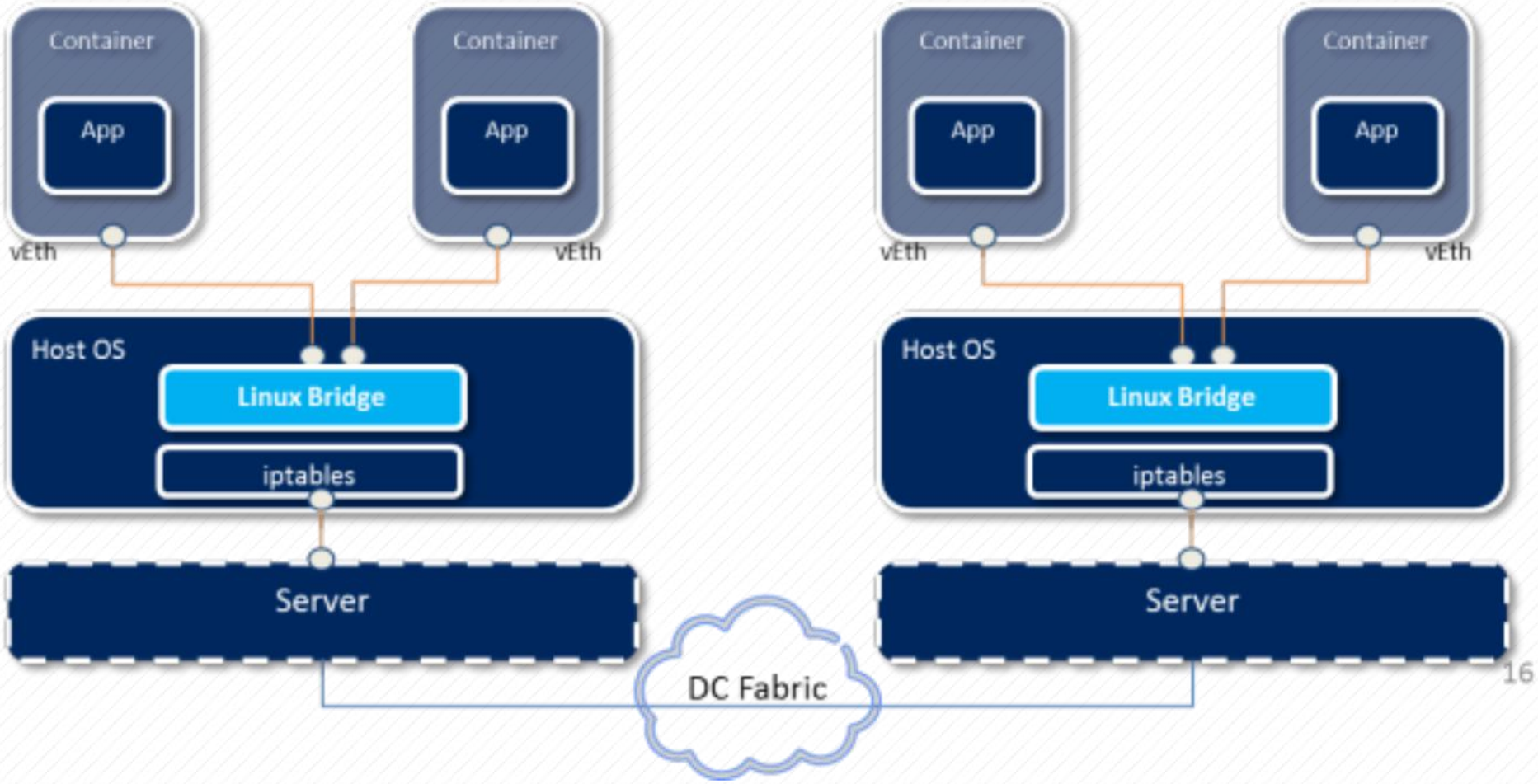
容器网络

跨主机容器网络/ 大二层



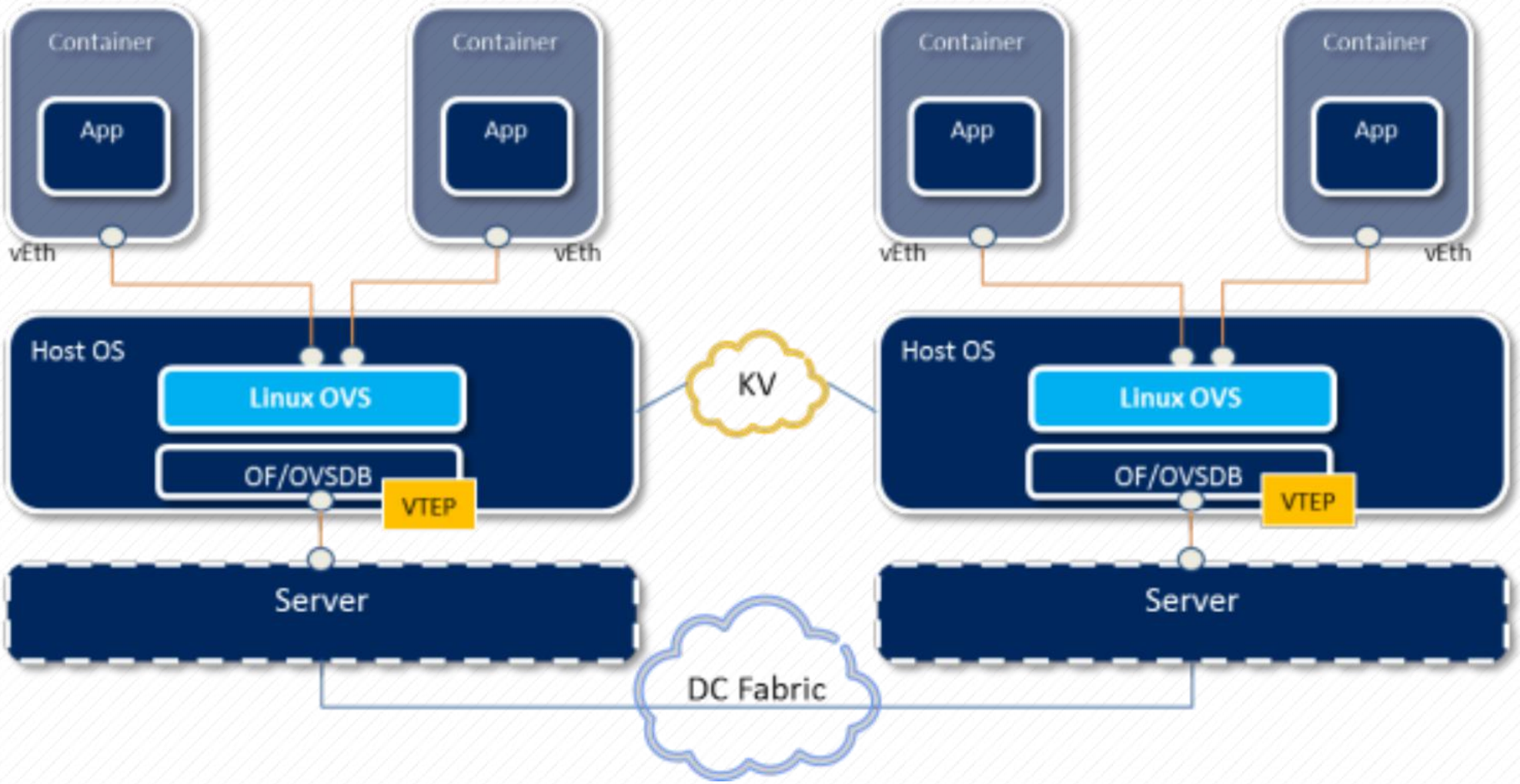
容器网络

跨主机容器网络/ NAT方式

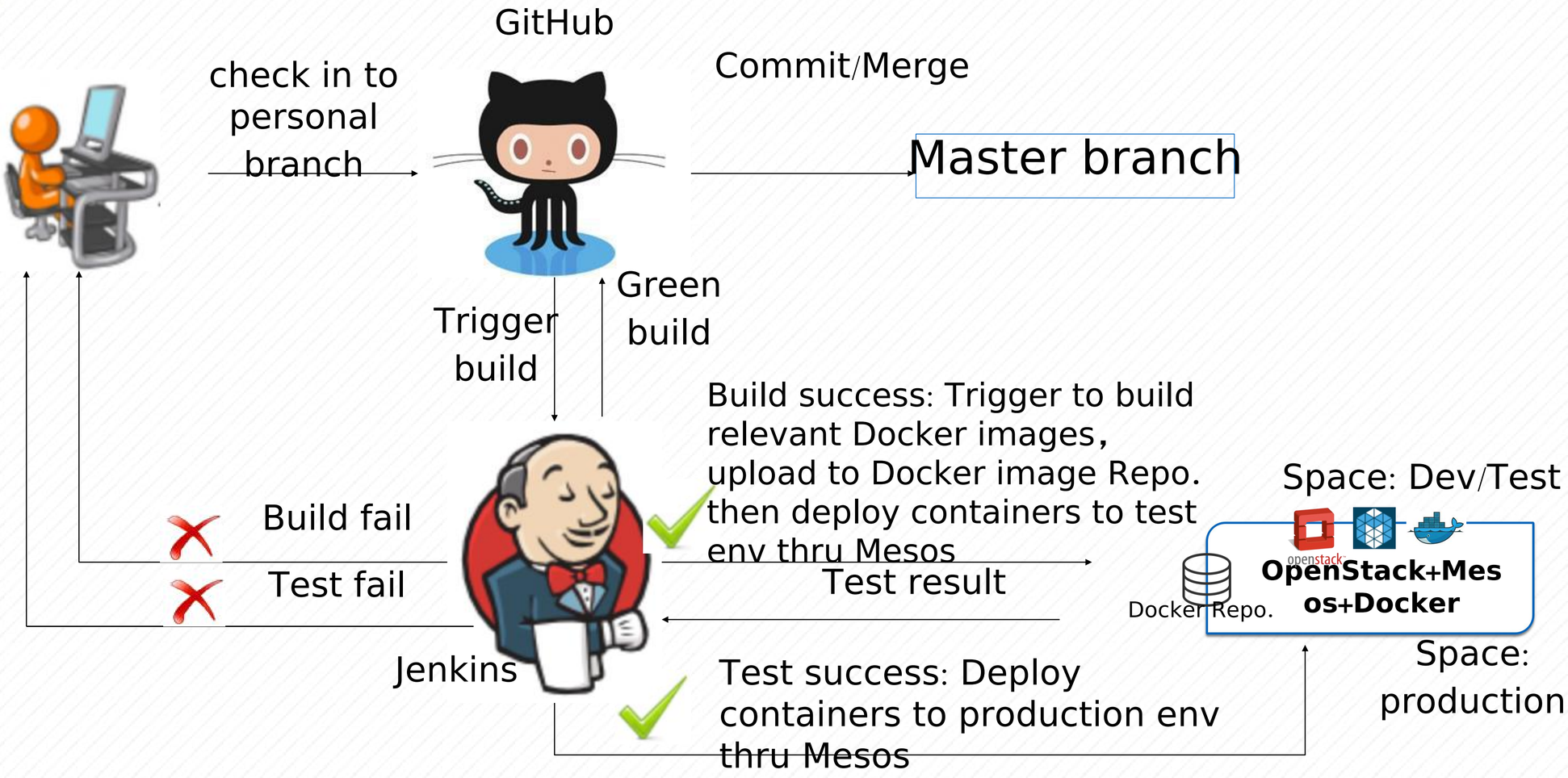


容器网络

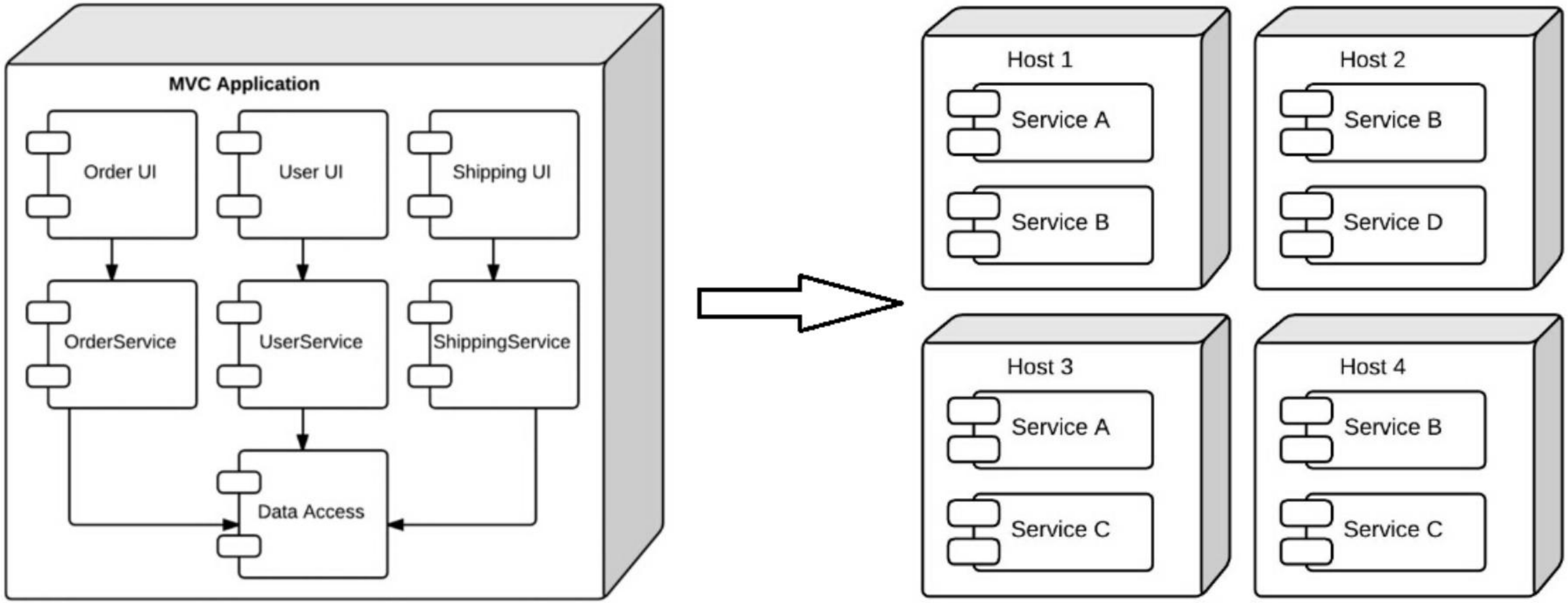
跨主机容器网络/ Tunnel方式



Docker典型应用


GitHub
check in to personal branch
Commit/Merge
Master branch
Green build
Trigger build
Build success: Trigger to build relevant Docker images, upload to Docker image Repo. then deploy containers to test env thru Mesos
Space: Dev/Test
Build fail
Test fail
Test result
OpenStack+Mesos+Docker
Docker Repo.
Jenkins
Test success: Deploy containers to production env thru Mesos
Space: production

Docker典型应用


MVC Application
Order UI
User UI
Shipping UI
OrderService
UserService
ShippingService
Data Access
Host 1
Service A
Service B
Host 2
Service B
Service D
Host 3
Service A
Service C
Host 4
Service B
Service C

THANK YOU