嘉为教育

容器技术基础

讲 师：张俊豪
邮 箱：junezhang@canway.net

学习目标

- 学完本课程后，您应该能够：
  - 了解docker容器的诞生背景
  - 掌握Docker的核心技术架构
  - 掌握Docker的安装方法和常见操作
  - 掌握构建自定义镜像的方法
  - 能够基于开源项目搭建私有镜像仓库
  - 了解 docker的网络模型
  - 理解数据卷和数据卷容器的概念

目录

第一章

容器概述

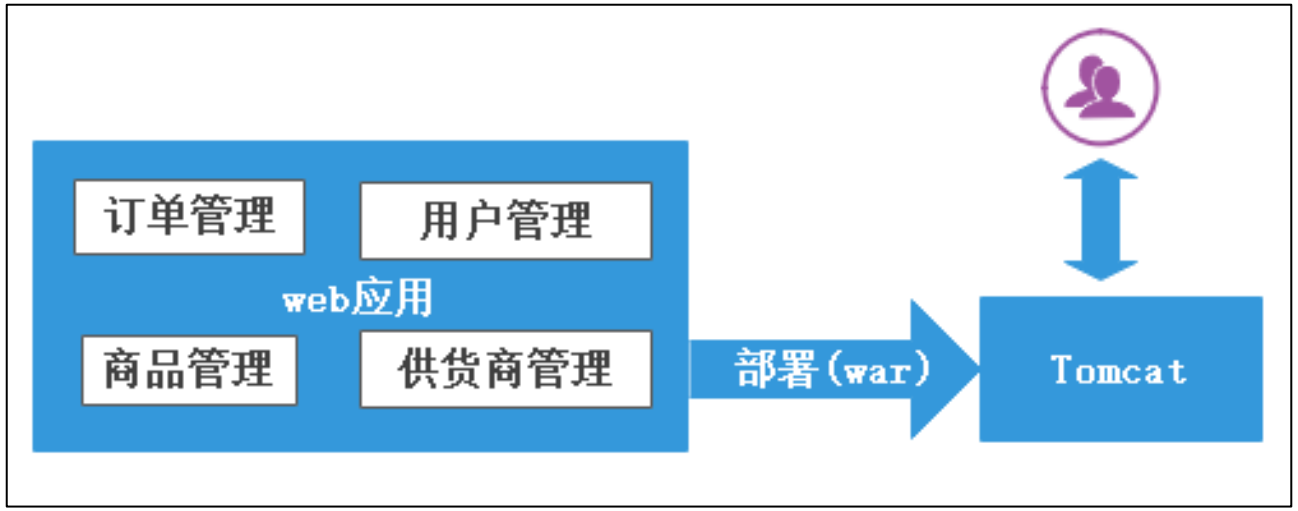
本章目录

1.1 容器的诞生

- 要想了解容器，首先需要知道软件架构的演进历程，软件架构依次经历了：

单体式架构

- 传统的项目会包含很多功能，是一个大而全的“超级”工程。
  - 例如：以普通架构方式实现的电商平台包含：登录、权限、会员、商品库存、订单、收藏、关注、购物车等功能的多个单一项目。随着项目业务越来越复杂、开发人员越来越多，相应开发、编译、部署、技术扩展、水平扩展都会受到限制。



单体式架构（续）

- 传统的单体应用无法适应快速增长的业务需求：

  - 单体应用的系统比较膨胀与臃肿，导致进行可持续开发和运维很困难。
  - 系统复杂：内部多个模块紧耦合，关联依赖复杂，牵一发而动全身
  - 运维困难：变更或升级的影响分析困难，任何一个小小修改都可能导致单体应用整体运维出现故障。
  - 无法扩展：很难通过水平扩展、多机部署的方式来提升系统的吞吐量，一般只能通过纵向不断堆单个机器或者群集的性能配置来提升。面临海量的互联网访问需求，很容易出现故障。

单体式架构（续）

如何解决?

SOA架构

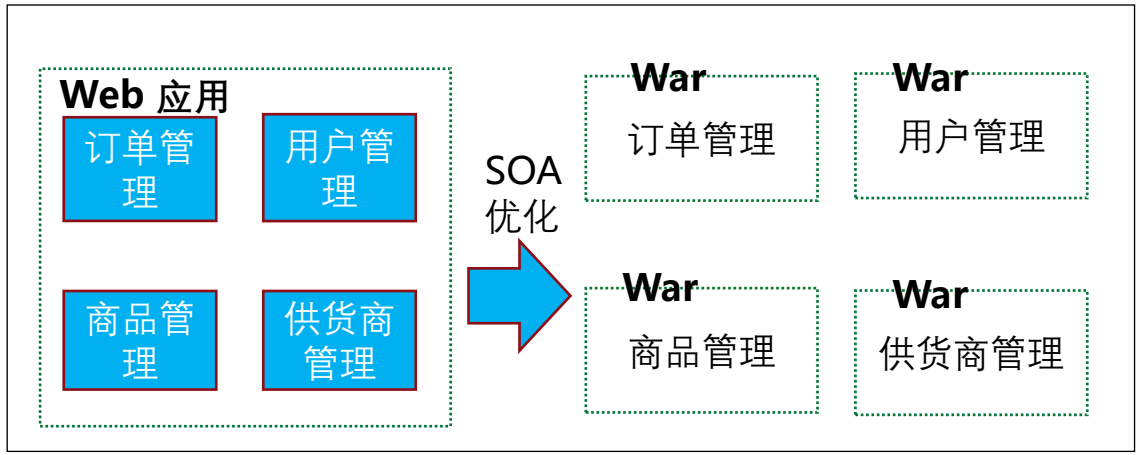
拆

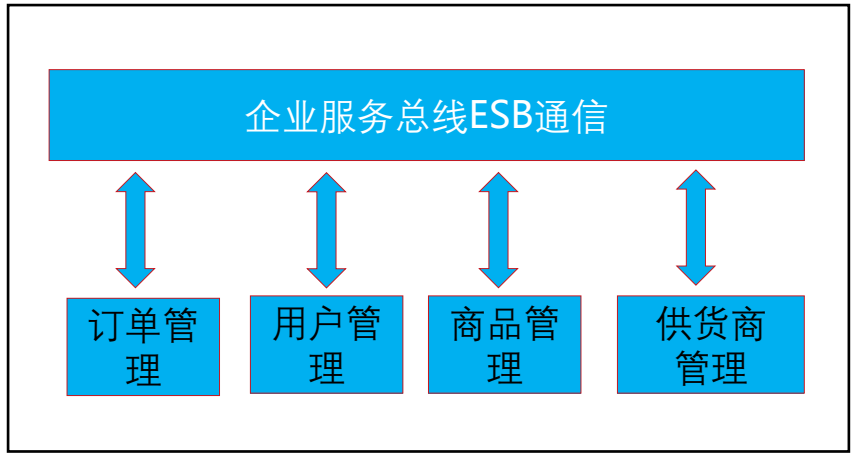
我们可以将一个庞大的单体应用拆分成多个服务模块，然后再将这些服务模块按照业务逻辑串起来，对外提供应用服务。这就是SOA（面向服务架构）的思路。

SOA：即面向服务的架构（SOA），是集成多个较大组件（一般是应用）的一种机制，它们将整体构成一个彼此协作的套件。一般来说，每个组件会从始至终执行一块完整的业务逻辑，通常包括完成整体大action所需的各种具体任务与功能。组件一般都是松耦合的，但这并非SOA架构模式的要求。

SOA架构

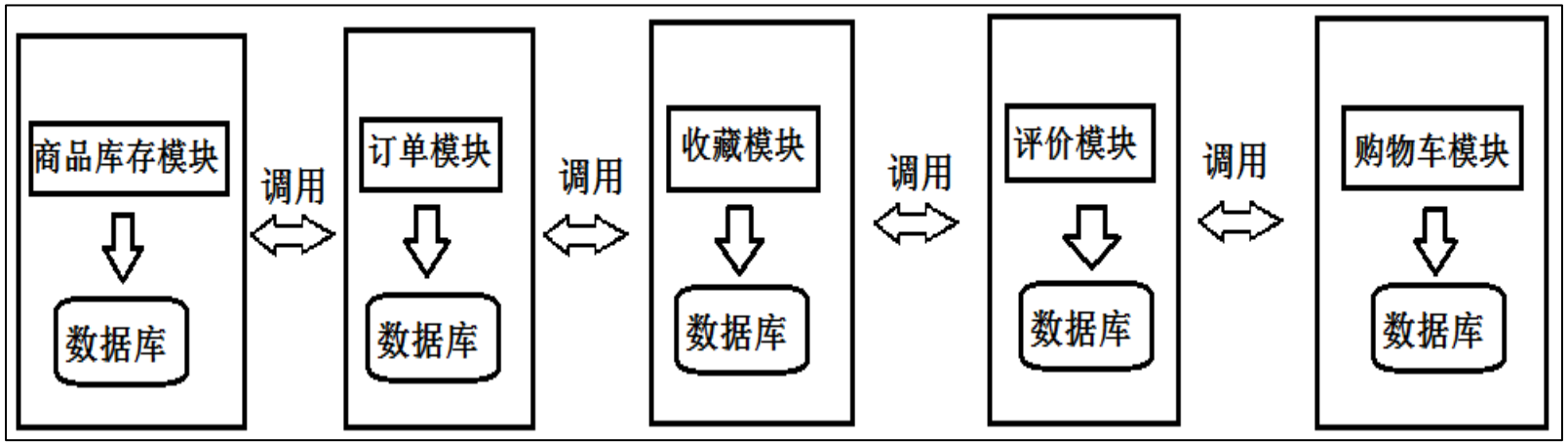
- SOA架构的特征
  - 基于SOA服务思想进行功能的抽取(重复代码问题解决),以服务为中心来管理项目。
  - 各个系统之间要进行调用,所以出现ESB来管理项目(可以使用各种技术实现：webservice，rpc等)。
  - ESB是作为系统与系统之间桥梁,很难进行统一管理。



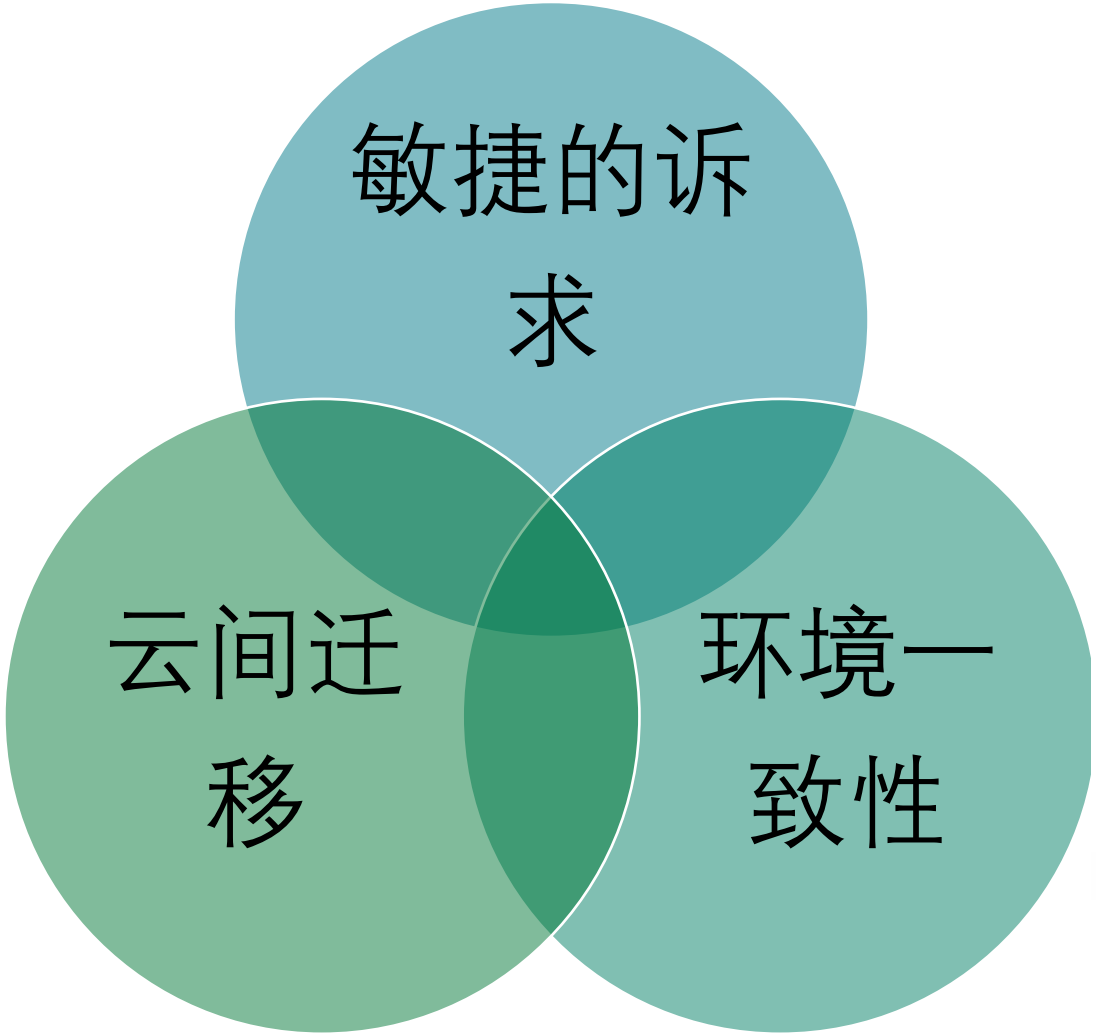


微服务架构

- 核心思路是拆分。单体项目的问题，通过把项目拆分成一个个小项目就可以解决。
- 与SOA区别：
  - 微服务不再强调传统SOA架构里面比较重的ESB企业服务总线，真正地实现服务自治；
  - 微服务的思想进入到单个业务系统内部，实现真正的组件化。



应用落地部署面临的挑战



小结

- 软件架构演变发展是怎么样的?
- 企业在应用落地部署时面临哪些挑战?
- 容器是如何解决以上问题的?

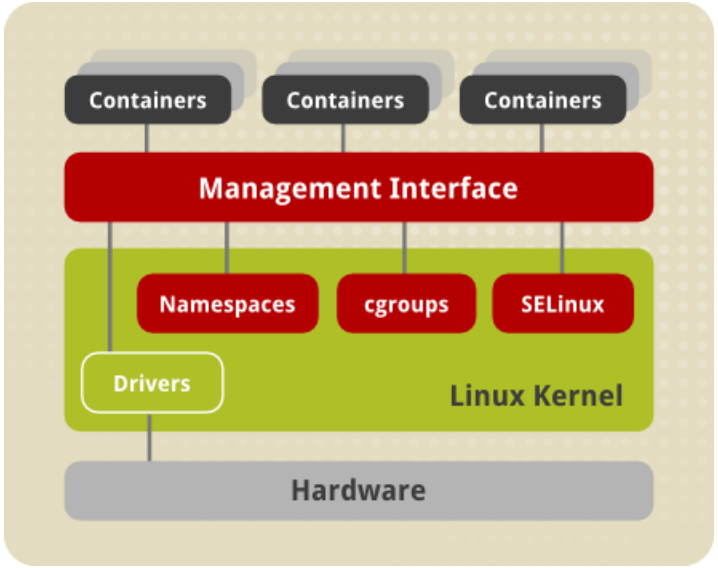
本章目录

1.2 Docker与容器

本小节，主要解决如下问题：

- 容器是什么？
- Docker是什么？
- Docker = 容器 ？

容器是什么？



Linux Container

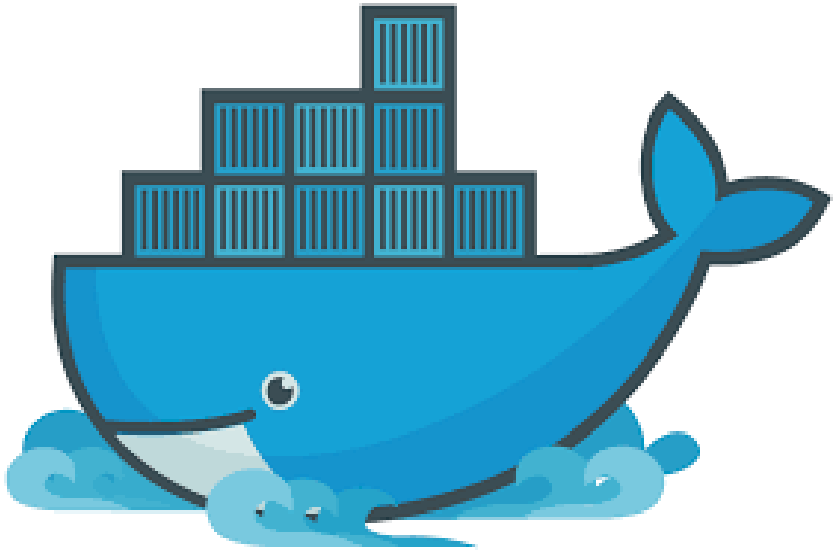
在Linux中，容器技术是一种进程隔离的技术，应用可以运行在一个个相互隔离的容器中，与虚拟机相同的是可以为这些容器设置计算资源限制，挂载存储，连接网络，而与虚拟机不同的是，这些应用运行时共用着一个Kernel，容器技术大大提升了对系统资源的利用率，也提高了应用的部署迁移效率。

Docker是什么？

2010年，28岁的Solomon Hykes在旧金山成立专做PaaS的公司，起名dotCloud，定位为为软件开发提供相关的配套设施，包括语言环境，运行环境，存储等基础服务。

在PaaS激烈竞争环境下，dotCloud为了生存，不得已开源核心引擎Docker，不料柳暗花明，dotCloud看到社区及众多PaaS大佬（Amazon, Google, IBM, MS, RedHat, VMware）对Docker的追捧及认可，顺势而为。

Docker与容器



Docker

Docker实际上是一家公司，在2013年这家公司还叫做DotCloud，Docker是他们公司的一个容器管理产品，**2013年初**，DotCloud决定将Docker开源，Docker在短短几个月间风靡全球，**DotCloud公司也更名为Docker**。

第二章

Docker核心技术及架构

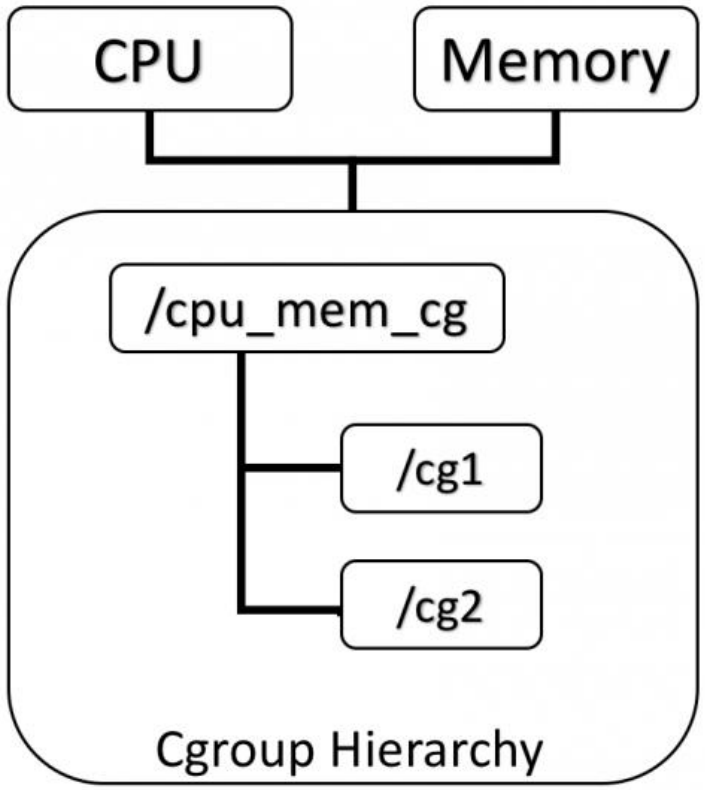
本章目录

2.1 进程隔离技术——namespace

Linux Namespace提供了一种内核级别的隔离系统资源的方法，通过将系统的全局资源放在不同的Namespace中，实现资源隔离的目的，不同的Namespace程序，可以享有一份独立的系统资源。

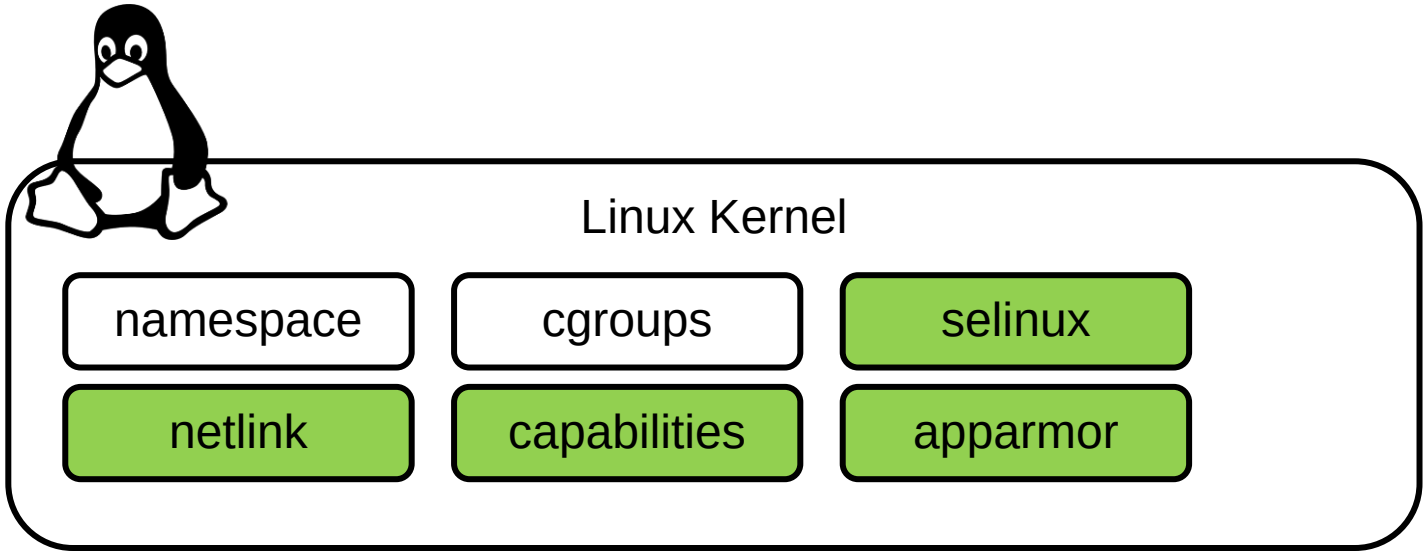
Namespace	隔离的内容
UTS	主机名与域名
IPC	信号量、消息队列和共享内容
PID	进程编号
Network	网络设备、网络栈、端口
Mount	文件系统
User	用户和用户组

进程资源配额技术-cgroups



利用Namespace可以构建一个相对隔离的容器，而通过cgrous，可以为容器设置系统资源配额，包括CPU、内存、IO等。

其他进程隔离技术



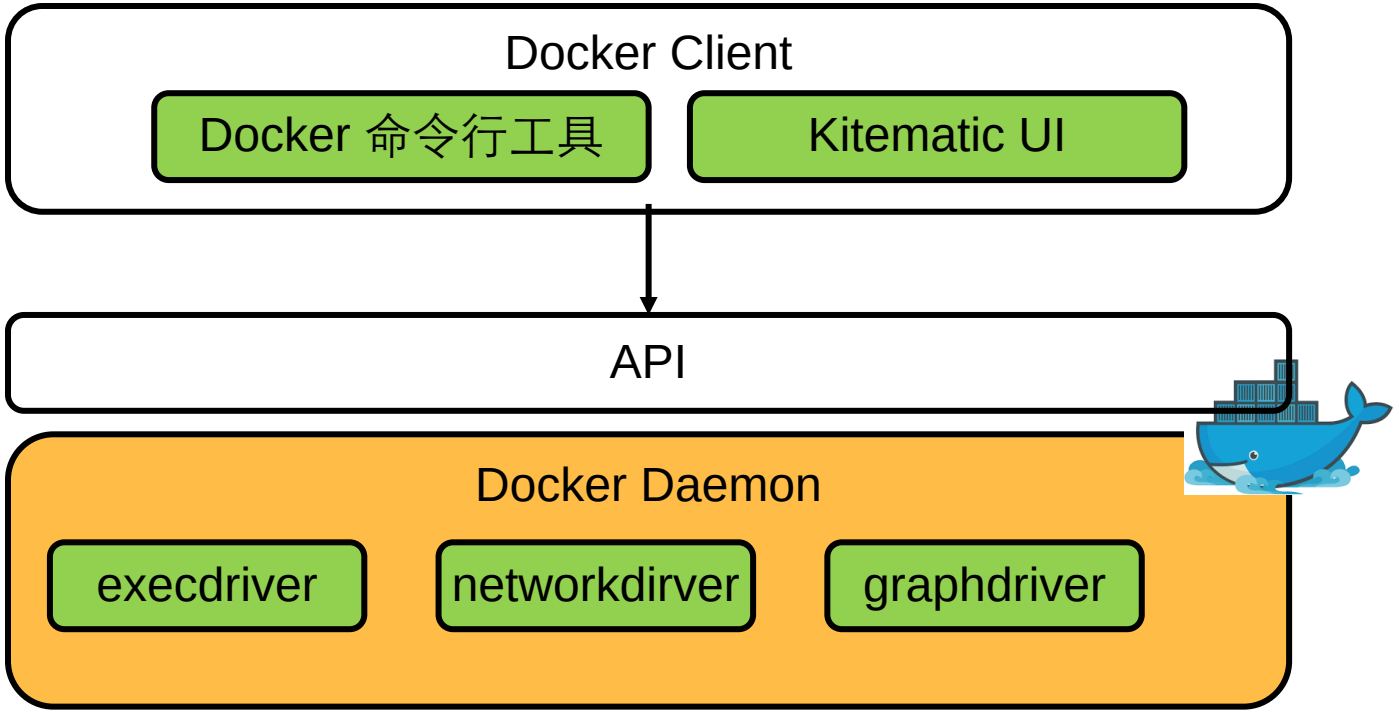
Selinux和apparmor可以增强对容器的访问控制；

Capability的主要实现在于将超级用户root的权限分割成多种不同的capability权限，从而更严格的控制容器的权限。

Netlink技术可完成Docker容器的网络环境配置与创建。

这些Linux 内核技术，从安全、隔离、防火墙、访问等方面为容器的成熟落地打下了坚实的基础。

Docker构造：client-server

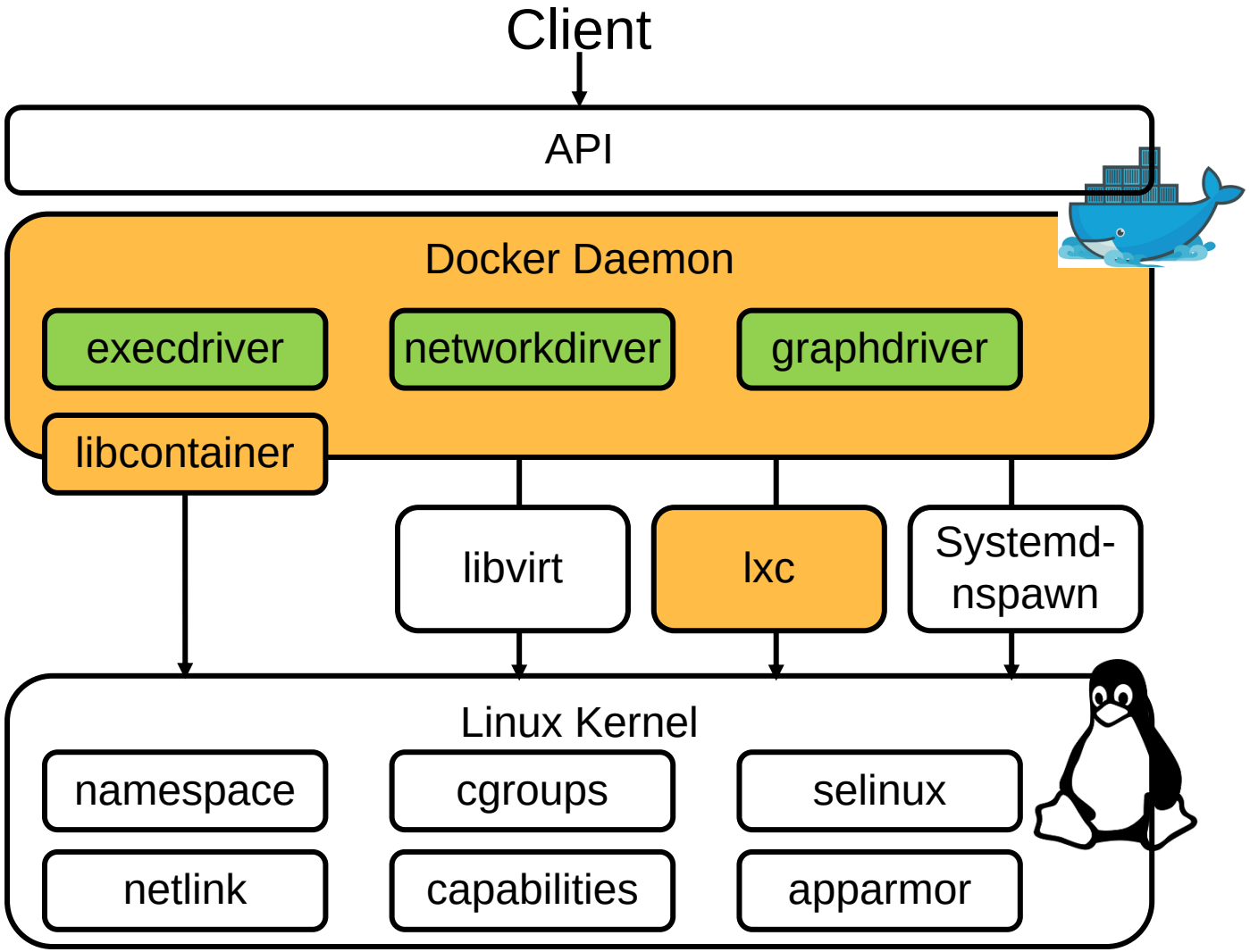


Execdriver存储了容器的定义的配置信息；

Networkdriver的作用是完成docker容器网络环境的配置，包括容器的IP、端口、防火墙策略及与主机的端口映射等；

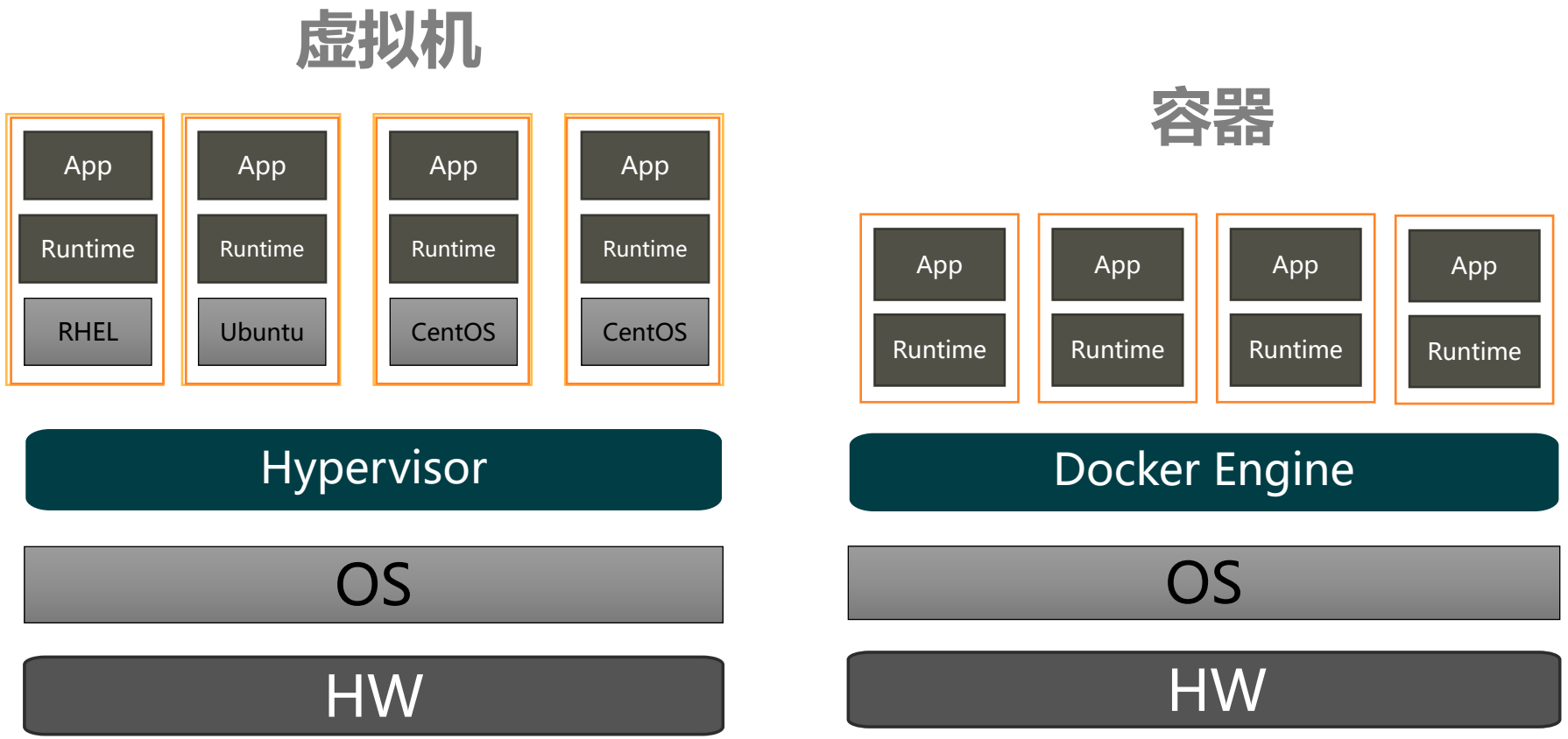
Graphdriver则负责对容器镜像的管理；

Docker Daemon的工作过程

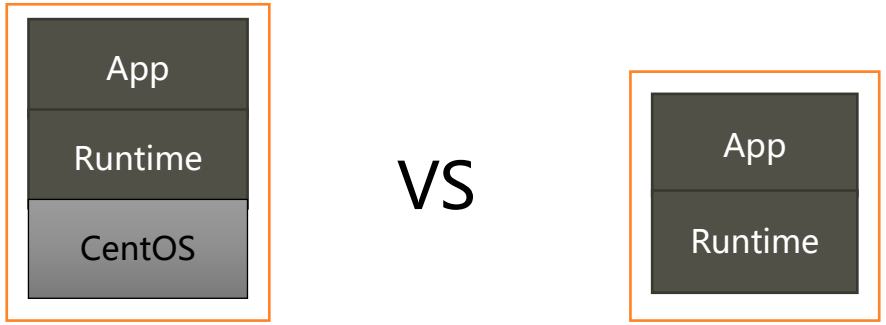


本章目录

2.2 容器 vs 虚拟化


虚拟机
App
Runtime
RHEL
App
Runtime
Ubuntu
App
Runtime
CentOS
App
Runtime
CentOS
Hypervisor
OS
HW
容器
App
Runtime
App
Runtime
App
Runtime
App
Runtime
Docker Engine
OS
HW

Docker的优势



对比项	VM	Docker
隔离性	强	较弱
计算资源开销	大	小
镜像大小	几百MB至几GB	可小至几MB
启动速度	数秒至数分钟	秒级
快速扩展能力	一般	强
跨平台迁移能力	一般	强
对微服务架构的支持	一般	强
对Devops的支持	一般	强

本章目录

2.3 容器常用术语

Docker Engine

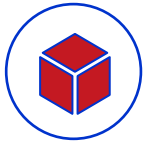
基础设施标准化

使得底层OS透明化

Docker Image

应用交付标准化

代替了以往代码+一堆部署文档的交付方式



Docker Container

运维管理标准化

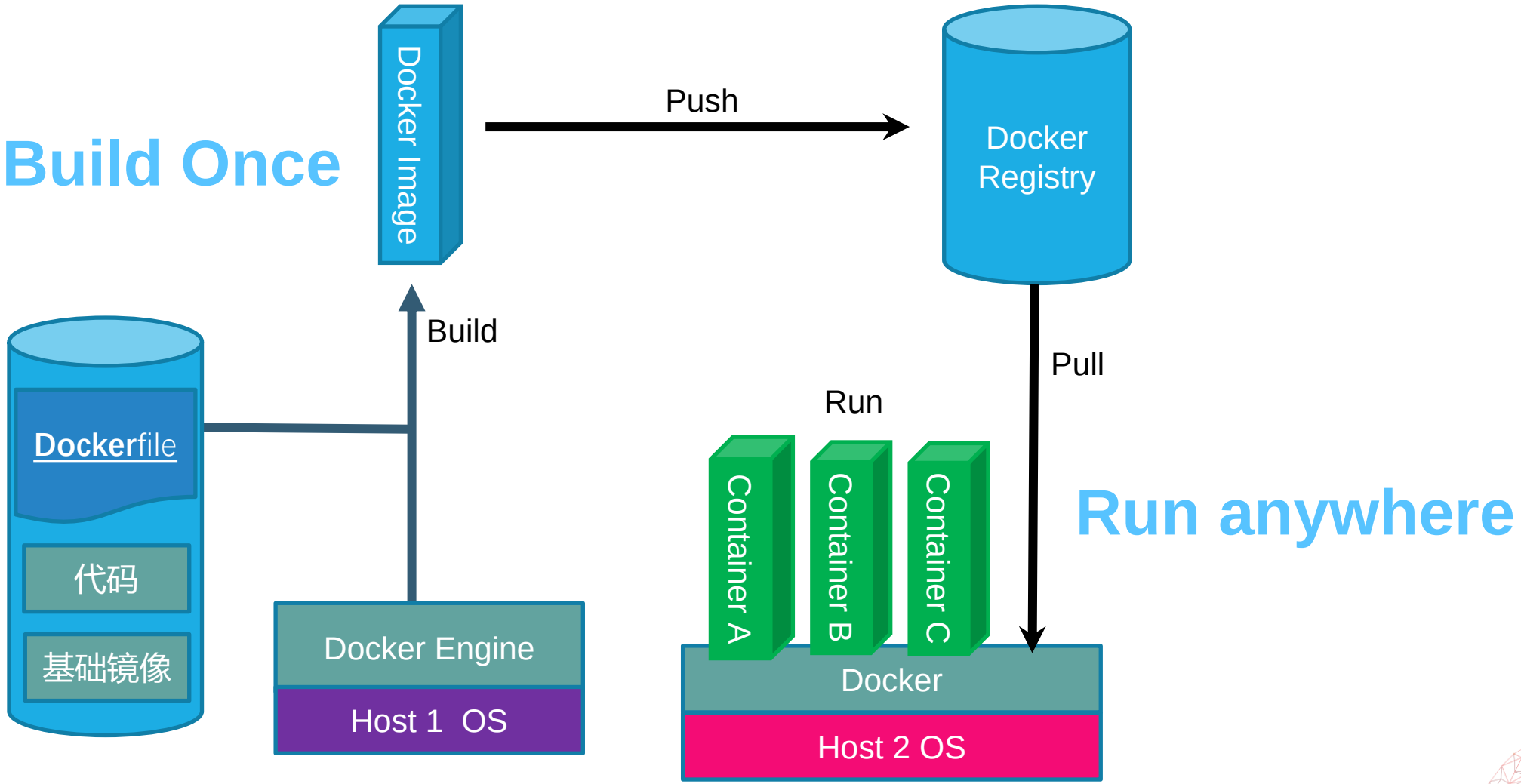
种类应用都跑在一个个标准化的容器中

Docker Registry

分发部署标准化

一次构建，随处部署

Docker核心概念


Build Once
Docker Image
Push
Docker Registry
Build
Pull
Run
Dockerfile
代码
基础镜像
Docker Engine
Host 1 OS
Container A
Container B
Container C
Docker
Host 2 OS
Run anywhere

第三章

Docker安装

本章目录

3.1 Windows Docker安装

- Windows10以前版本：

◘安装 Docker Toolbox，同时还附加安装：

- Windows10版本：

◘安装Docker Desktop for windows

https://docs.docker.com/docker-for-windows/install/

本章目录

3.2 Linux Docker安装

- 安装前准备：
  - cat /etc/centos-release
  - uname –r　　#内核版本在3.10以上
- 配置yum源：

  cd /etc/yum.repos.d/

  wget http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
- 安装 Docker 软件
  - yum install docker-ce -y
- 启动 Docker 服务
  - systemctl start docker
  - systemctl enable docker
- 验证
  - docker info
  - docker --version

第四章

Docker基本操作

本章目录

4.1 启动docker容器

- 拉取镜像
  - docker search centos
  - docker pull centos
- 查看镜像
  - docker images
- 创建一个后台容器
  - docker run –itd --name c1 centos /bin/bash
- 查看容器
  - docker ps [-a]
  - docker inspect c1

启动docker容器

- 容器生命周期管理
  - docker stop c1
  - docker start c1
  - docker restart c1
  - docker rm c1
- 打印容器的控制台输出
  - docker logs c1
- 进入后台容器
  - docker exec -it c1 /bin/bash

本章目录

4.2 镜像下载加速器

- 加速器：要运行容器，首先需要下载一个镜像，例如 mysql、wordpress，然而由于网络原因，从公共仓库中下载一个官方镜像可能会需要很长的时间，甚至下载失败，可以通过配置镜像下载加速器的方式提升镜像下载速度。
- mkdir -p /etc/docker
- tee /etc/docker/daemon.json <<-'EOF'

```
{
  "registry-mirrors": ["https://m4x67mmr.mirror.aliyuncs.com"]
}
EOF
```

- systemctl daemon-reload
- systemctl restart docker

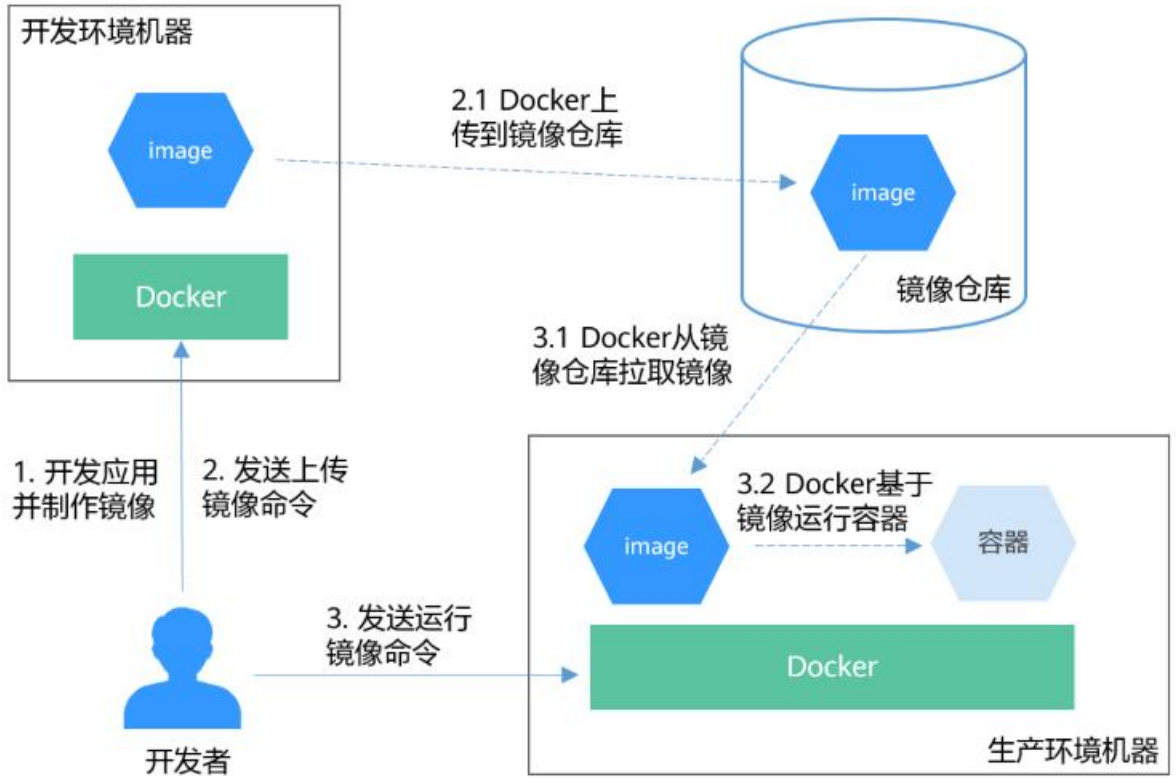
第五章

Docker自定义镜像

本章目录

5.1 自定义镜像概述

镜像使用流程：



1、首先开发者在开发环境机器上开发应用并制作镜像。Docker执行命令，构建镜像并存储在机器上；

2、开发者发送上传镜像命令，Docker收到命令后，将本地镜像上传到镜像仓库。

3、开发者向生产环境机器发送运行镜像命令，生产环境机器收到命令后，Docker会从镜像仓库拉取镜像到机器上，然后基于镜像运行容器。

镜像分层特征示例


MyApp
LAMP
Core HTTPD
RHEL Base
Customization
Liferay
Jboss EAP
OpenJDK
RHEL Base
Customization
Liferay
Jboss EAP
OpenJDK
RHEL Base
LAMP
Core HTTPD
RHEL Base
Jboss BPM
Jboss EAP
OpenJDK
RHEL Base
Liferay
Jboss EAP
OpenJDK
RHEL Base
Jboss EAP
OpenJDK
RHEL Base
Core HTTPD
RHEL Base
OpenJDK
RHEL Base
RHEL Base

本章目录

5.2 基于容器构建自定义镜像

- Docker commit构建自定义镜像的方法类似于虚拟化中虚拟机模板的制作；
- 具体步骤：
  - docker run -itd --name con1 centos /bin/bash
  - docker exec -it con1 /bin/bash
  - ……容器内进行配置操作……
  - docker stop 容器ID
  - docker commit –m “my new centos” –a “harry” con1 my/centos:v2
  - docker images

本章目录

5.3 Dockerfile构建自定义镜像

Dockerfile基本语法：

- FROM
  - 基于哪个镜像
- MAINTAINER
  - 镜像创建者
- RUN
  - 安装软件用
- ADD/COPY
  - 将宿主机文件拷贝到容器中
- CMD
  - container启动时执行的命令，但是一个Dockerfile中只能有一条CMD命令，多条则只执行最后一条CMD.
- ENTRYPOINT
  - 配置容器启动后执行的命令，并且不可被 docker run 提供的参数覆盖。

Dockerfile构建自定义镜像

示例：

◘编写Dockerfile文件

- FROM nginx:1.13.12
- MAINTAINER Aaron
- ADD index.html /usr/share/nginx/html/index.html

◘ 自定义镜像

- docker build -t my/nginxv1 .
- docker images

◘ 验证

- docker run -itd --name con1 my/nginx:v1
- curl http://ip --验证Nginx服务

本章目录

5.4 镜像导出和导入

制作好镜像后，可以对镜像执行打包导出的动作，拷贝到目标服务器，再通过镜像导入的方式实现镜像跨主机的迁移；

- **镜像导出：**

```
docker save -o centos_bk.tar.gz centos
```

- **镜像导入**

```
docker load  -i centos_bk.tar.gz
```

第六章

Docker镜像仓库

本章目录

6.1 镜像仓库概述

- 镜像仓库：

集中存放镜像的地方，容易混淆的是注册服务器，它是管理镜像仓库的服务器，每个服务器可以有多个仓库，每个仓库可以有多个镜像，因此仓库可以认为是一个具体的目录，

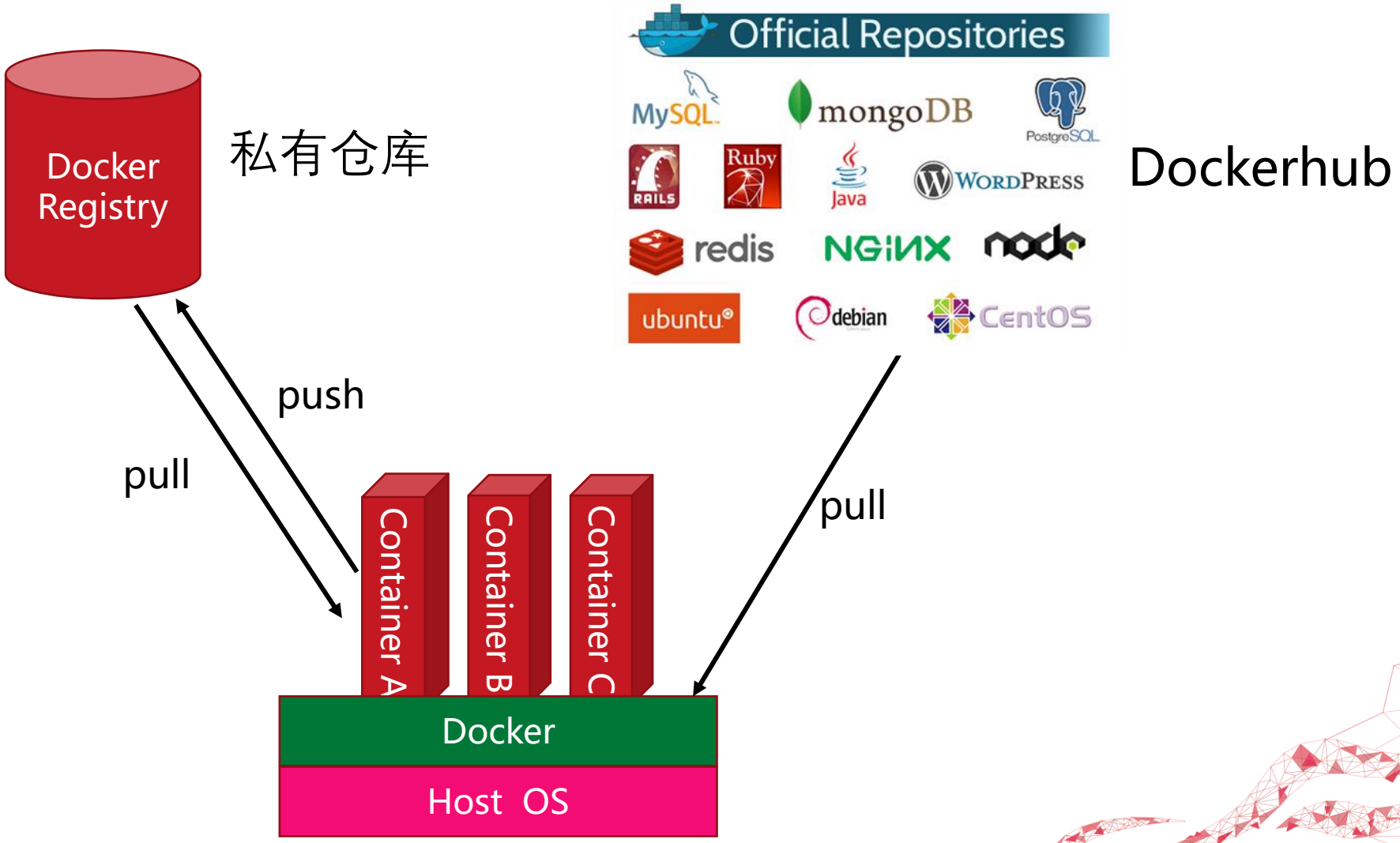
例如：dl.dockerpool.com/repo/centos：tag

dl.dockerpool.com是注册服务器，repo是镜像仓库的名称，centos是镜像的名称；

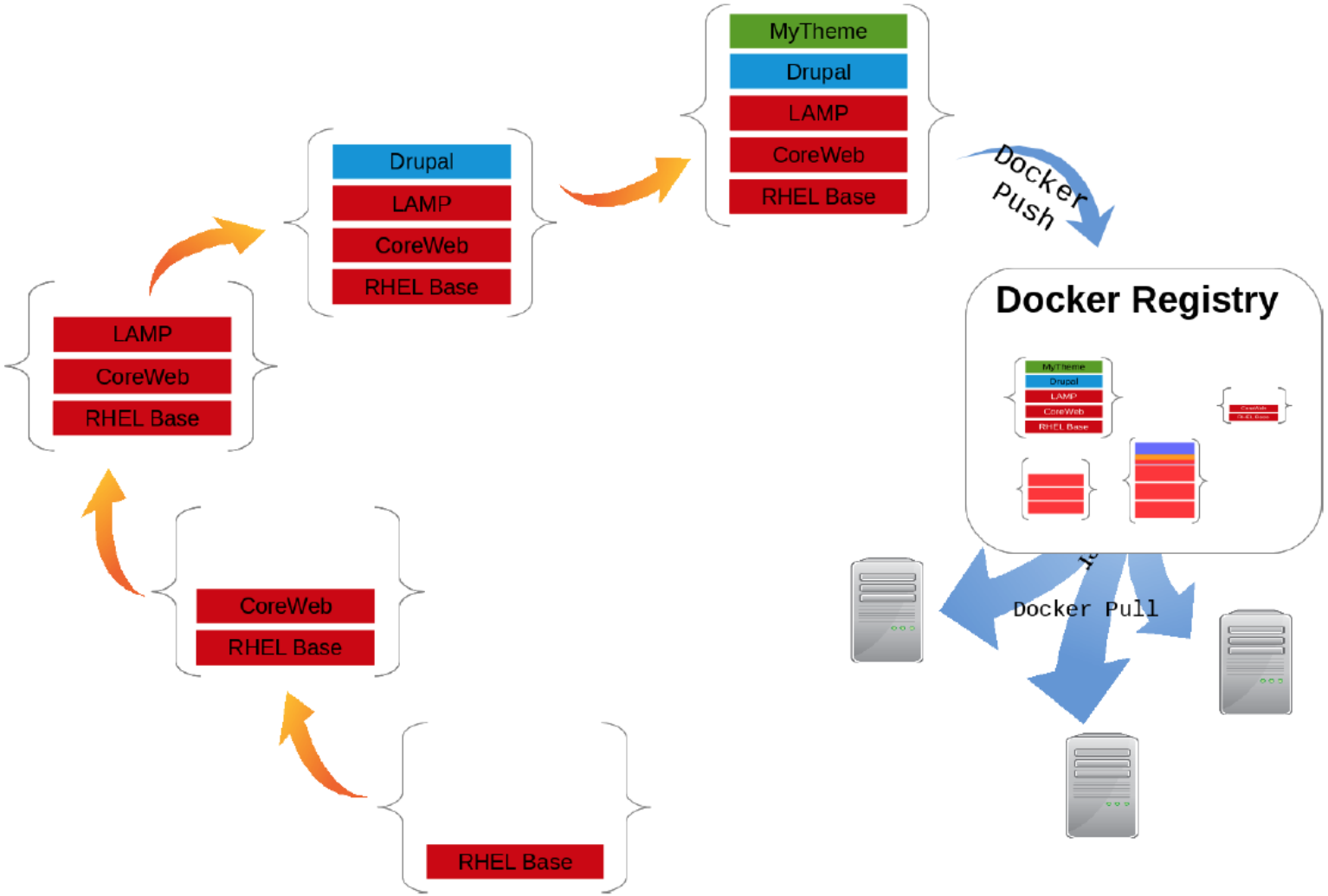
- 镜像仓库分类：

- 公共仓库：docker官方维护的公共库，https://hub.docker.com/ ，可注册账号后构建自己的私有存储空间；
- 私有仓库：在公司内部为了提高分享的速度，需要在公司内部自己搭建一个本地的仓库，供私人使用；

镜像仓库


Docker Registry
私有仓库
push
pull
pull
Container A
Container B
Container C
Docker
Host OS
Official Repositories
MySQL
mongoDB
PostgreSQL
RAILS
Ruby
Java
WordPress
redis
NGINX
node
ubuntu
debian
CentOS
Dockerhub

Docker仓库功能示例



本章目录

6.2 私有仓库构建实践（一）

基于开源的registry镜像构建私有仓库:

1）下载registry镜像

docker pull registry

2）启动镜像注册容器服务：https://docs.docker.com/registry/deploying/

docker run -d -p 5000:5000 --restart=always --name registry registry

3）对下载好的镜像打标签

docker tag

4）上传镜像：

docker push

注：如果提示 ……response to HTTPS…… 可参考https://docs.docker.com/registry/insecure/进行调试

本章目录

6.3 私有仓库构建实践（二）

基于开源的Harbor项目构建私有仓库：

Harbor项目介绍：

Harbor是由VMware公司开源的企业级的Docker Registry管理项目，它包括权限管理(RBAC)、LDAP、日志审核、管理界面、自我注册、镜像复制和中文支持等功能。

实验环境准备：基于Centos7.2版本虚拟机完成如下操作：

1、安装docker：yum install –y docker-ce

2、安装docker-compose：

1）下载docker-compose软件：

curl -L https://github.com/docker/compose/releases/download/1.13.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

2）添加执行权限

chmod +x /usr/local/bin/docker-compose

3）验证docker-compose安装成功

docker-compose --version

6.3 私有仓库构建实践（二）

3、Harbor服务搭建：

1）到github harbor官网下载指定版本的安装包：https://github.com/vmware/harbor

A:下载离线安装包

wget https://github.com/vmware/harbor/releases/download/v1.1.2/harbor-offline-installer-v1.1.2.tgz

$ tar xvf harbor-offline-installer-v1.1.2.tgz

B:下载在线安装包

wget https://github.com/vmware/harbor/releases/download/v1.1.2/harbor-online-installer-v1.1.2.tgz

tar xvf harbor-online-installer-v1.1.2.tgz

2）配置Harbor

解压缩之后，目录下生成harbor.cfg文件，即Harbor的配置文件

vim　harbor.cfg

hostname = 192.168.56.10　　# hostname设置访问地址，比如ip或域名

harbor_admin_password = Harbor12345　#admin账号登录密码

6.3 私有仓库构建实践（二）

3）启动harbor服务

./install.sh

harbor服务就会根据当前目录下docker-compose.yml文件中定义的依赖的镜像，下载并启动各个微服务容器；

4、访问harbor私有仓库：

打开浏览器，输入http://192.168.56.10

账号：admin　密码：Harbor12345

5、命令行方式login镜像仓库并push镜像进行验证

6.3 私有仓库构建实践（二）

Q：命令行下执行docker login时，提示：Error response from daemon: Get https://192.168.56.10/v1/users/: dial tcp 192.168.56.10:443: getsockopt: connection refused 怎么办?

A：修改建议：

vim /etc/docker/daemon.json

OPTIONS选项增加： “insecure-registries”：[“192.168.56.10”]

systemctl daemon-reload

systemctl restart docker

docker-compose start

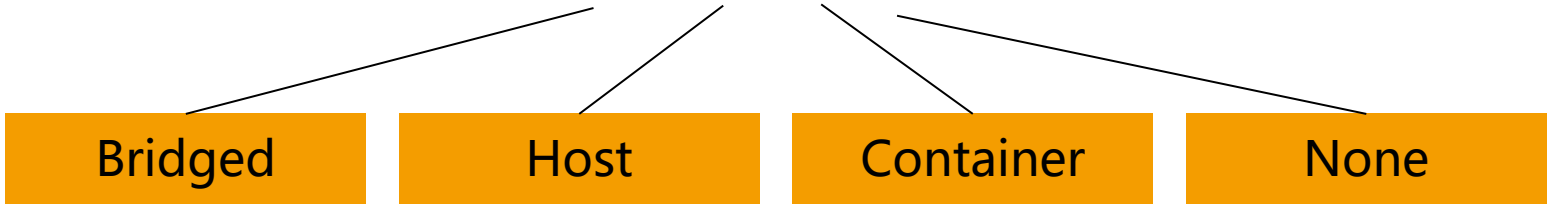
第七章

Docker网络管理

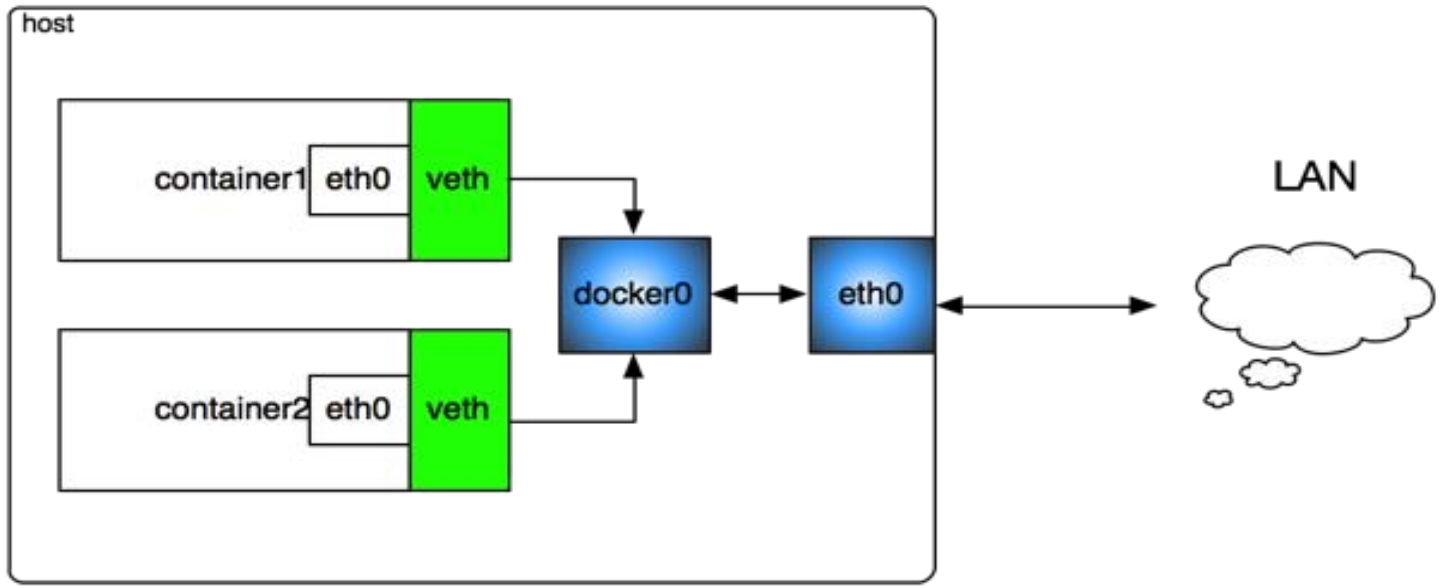
本章目录

7.1 容器网络概述

Docker的四种网络模式



Bridged模式



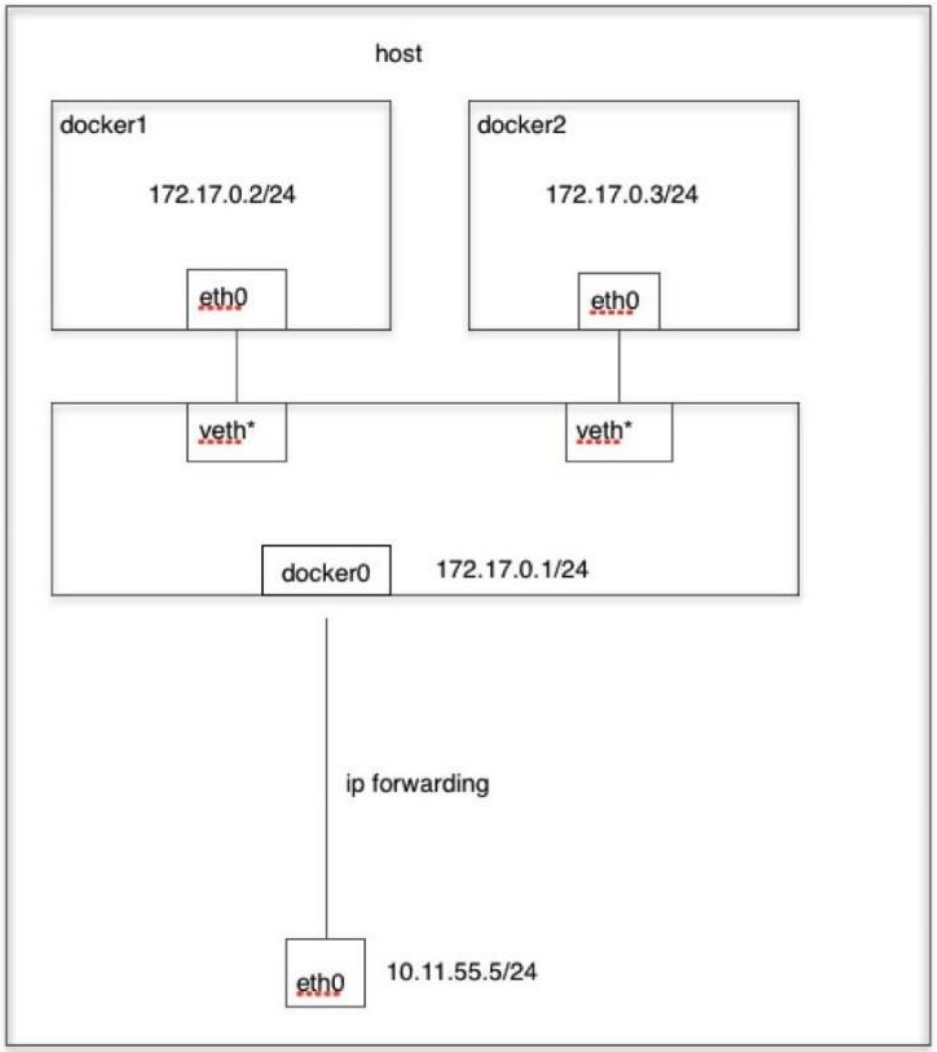
本章目录

7.2 bridge网络模式

桥接模式：

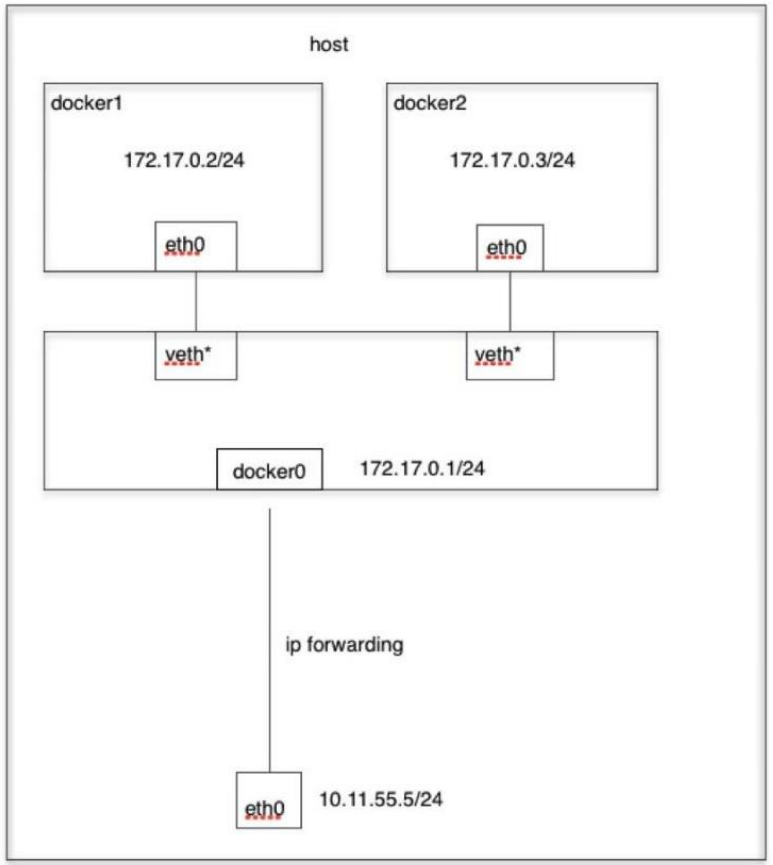
当Docker进程启动时，会在主机上创建一个名为docker0的虚拟网桥，此主机上启动的Docker容器会连接到这个虚拟网桥上。虚拟网桥的工作方式和物理交换机类似，这样主机上的所有容器就通过交换机连在了一个二层网络中。

从docker0子网中分配一个IP给容器使用，并设置docker0的IP地址为容器的默认网关。在主机上创建一对虚拟网卡veth pair设备，Docker将veth pair设备的一端放在新创建的容器中，并命名为eth0（容器的网卡），另一端放在主机中，以vethxxx这样类似的名字命名，并将这个网络设备加入到docker0网桥中。可以通过brctl show命令查看，可以在容器和host上使用ip link show 的方法定位容器和veth的对应关系。

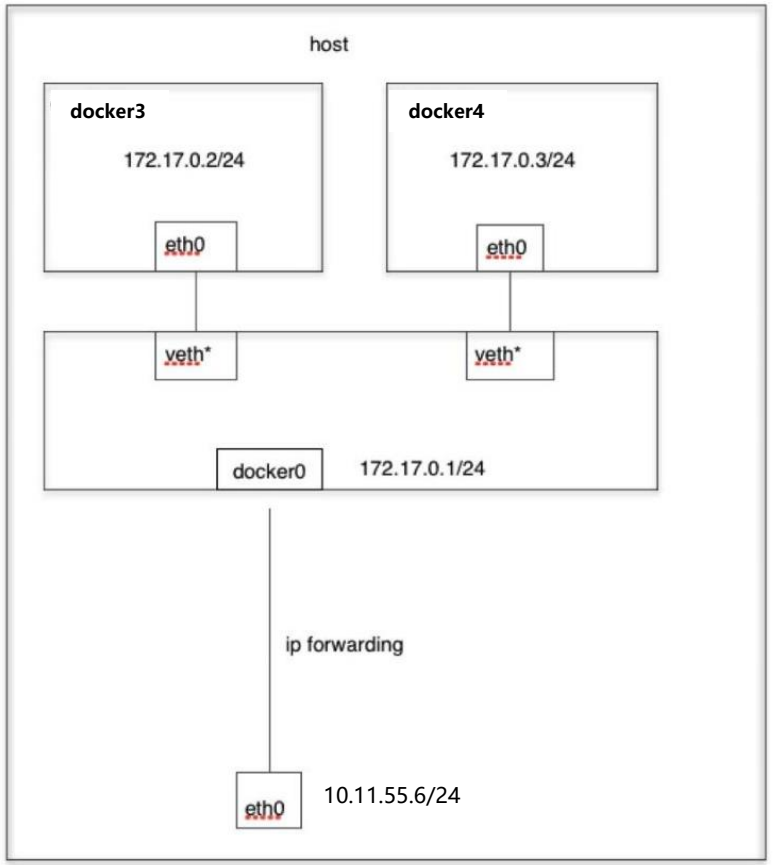


思考题

- 当应用容器集群跨宿主机部署时，IP冲突问题，如何解决?



Host1



Host2

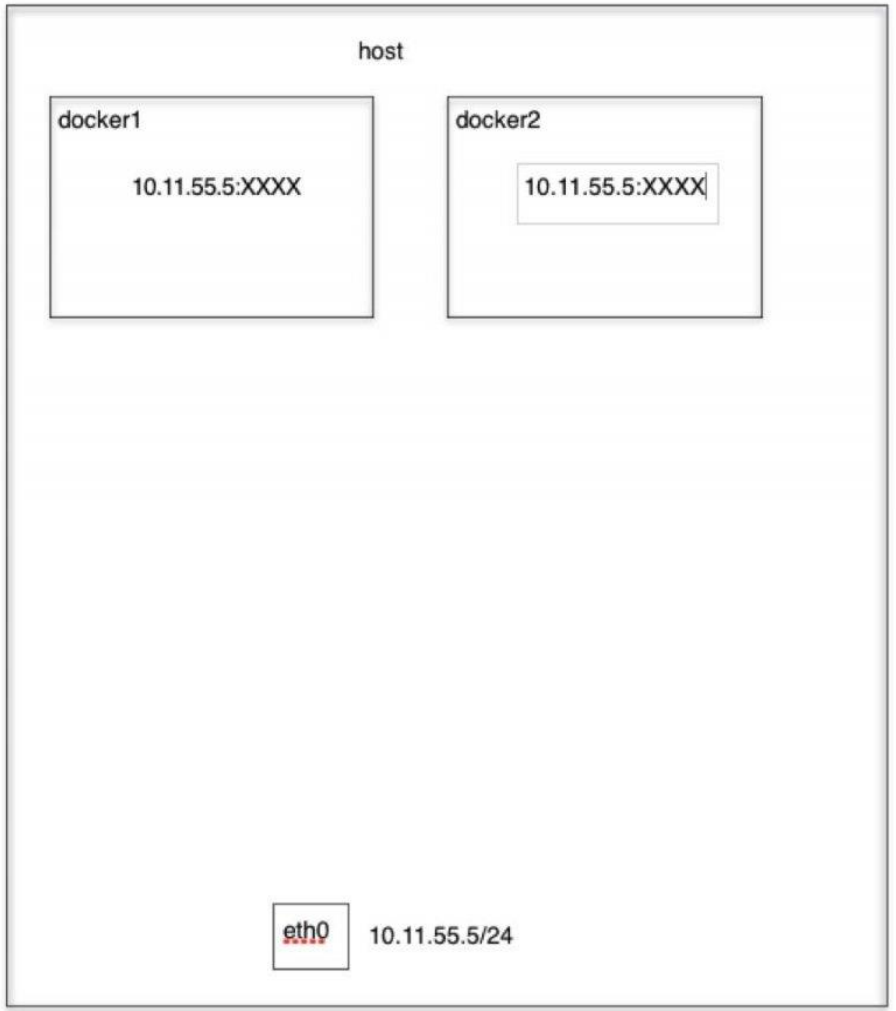
本章目录

Docker四种网络模式

Host模式：

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。但是，容器的其他方面，如文件系统、进程列表等还是和宿主机隔离的。



示例：

#docker run -tid --net=host --name host1 centos /bin/bash

#docker run -tid --net=host --name host2 centos /bin/bash

docker exec 去验证

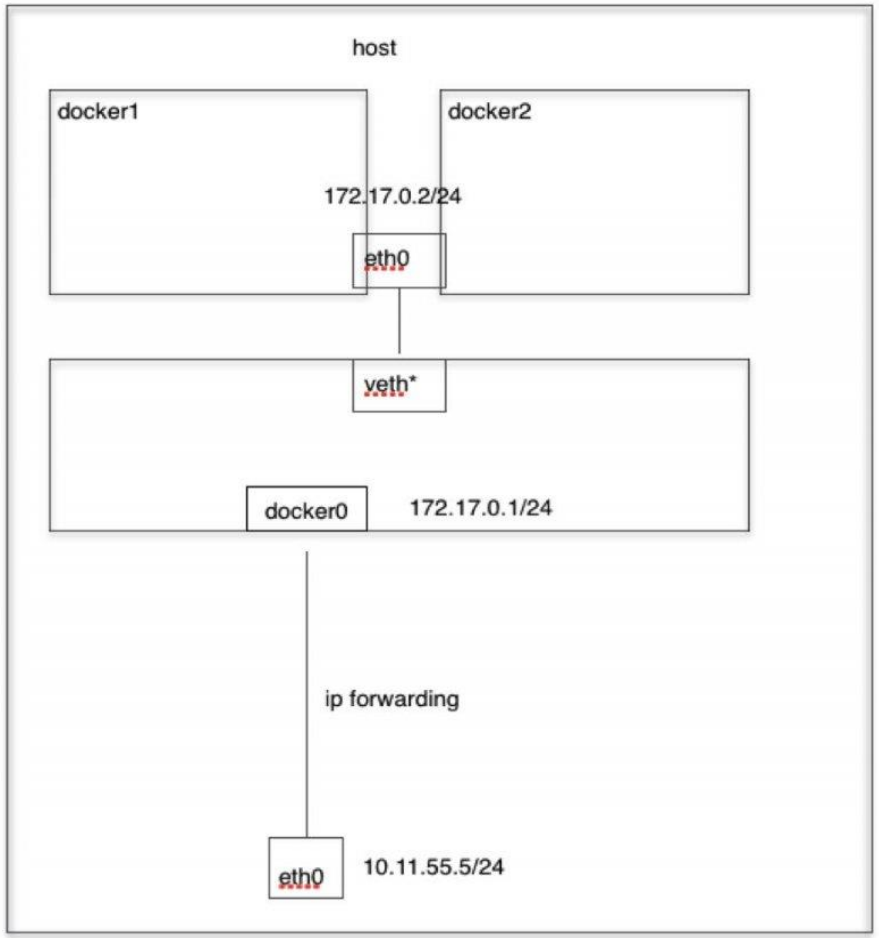
本章目录

Docker四种网络模式

Container模式：

该模式指定新创建容器和已存在的容器共享一个 Network Namespace，而不是和宿主机共享。

新创建的容器不会创建自己的网卡，配置自己的 IP，而是和一个指定的容器共享 IP、端口范围等。同样，两个容器除了网络方面，其他的如文件系统、进程列表等还是隔离的。两个容器的进程可以通过 lo 网卡设备通信。



示例：

#docker run -tid --name=container0 centos /bin/bash

#docker run -tid --net=container:container0 --name container1 centos /bin/bash

docker exec 去验证

本章目录

Docker四种网络模式

None模式：

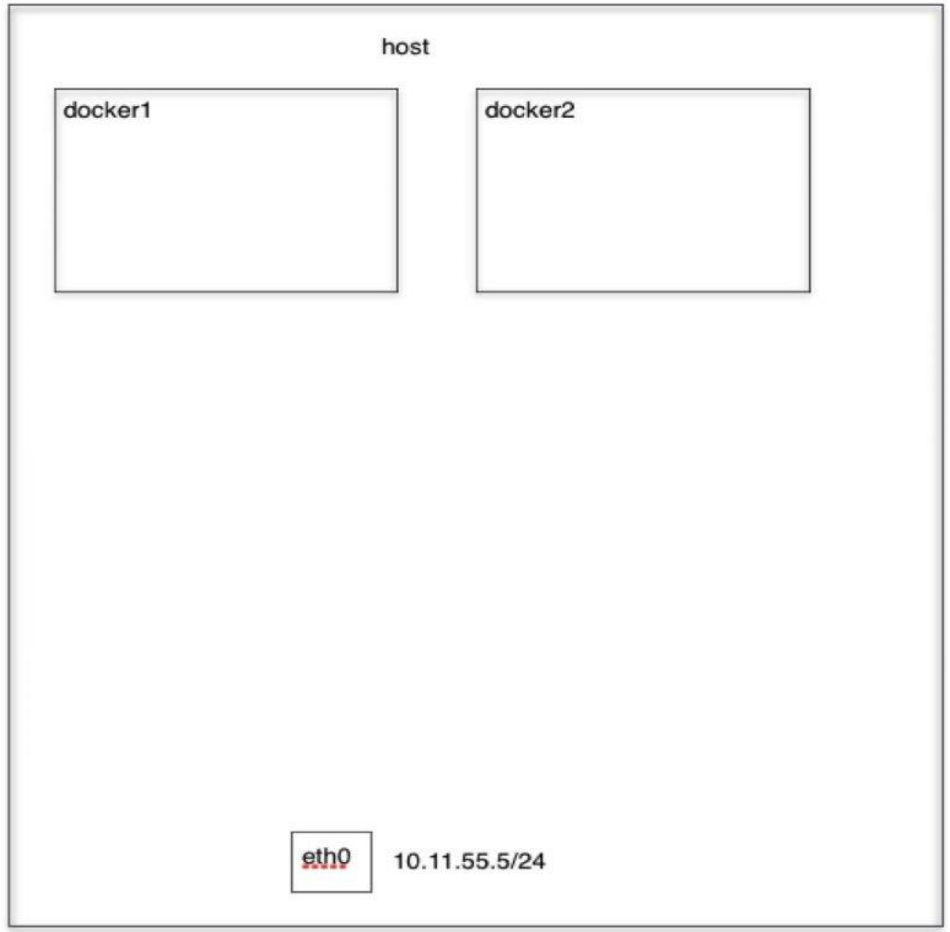
使用none模式，Docker容器拥有自己的Network Namespace，但是，并不为Docker容器进行任何网络配置。

也就是说，Docker容器没有网卡、IP、路由等信息。需要我们自己为Docker容器添加网卡、配置IP等。

示例：

#docker run -itd --net=none --name none1 centos /bin/bash

docker exec 去验证



第八章

容器数据持久化

本章目录

8.1 数据卷

Docker内部及容器间管理数据的方式：

- 数据卷
- 数据卷容器

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，可以提供很多有用的特性：

- 实现容器数据的持久化存储
- 对数据卷的修改会立马生效
- 对数据卷的更新，不会影响镜像
- 卷会一直存在，直到没有容器使用 *数据卷的使用，类似于 Linux 下对目录或文件进行 mount。

示例

- 创建数据卷：

```
docker run  -itd --name con1  centos -v  /data0  /bin/bash
docker run  -itd --name con2  centos -v  /local_data0:/data0  /bin/bash
```

- 验证数据卷映射：

```
docker inspect con1/con2
docker stop con2
docker rm con2
```

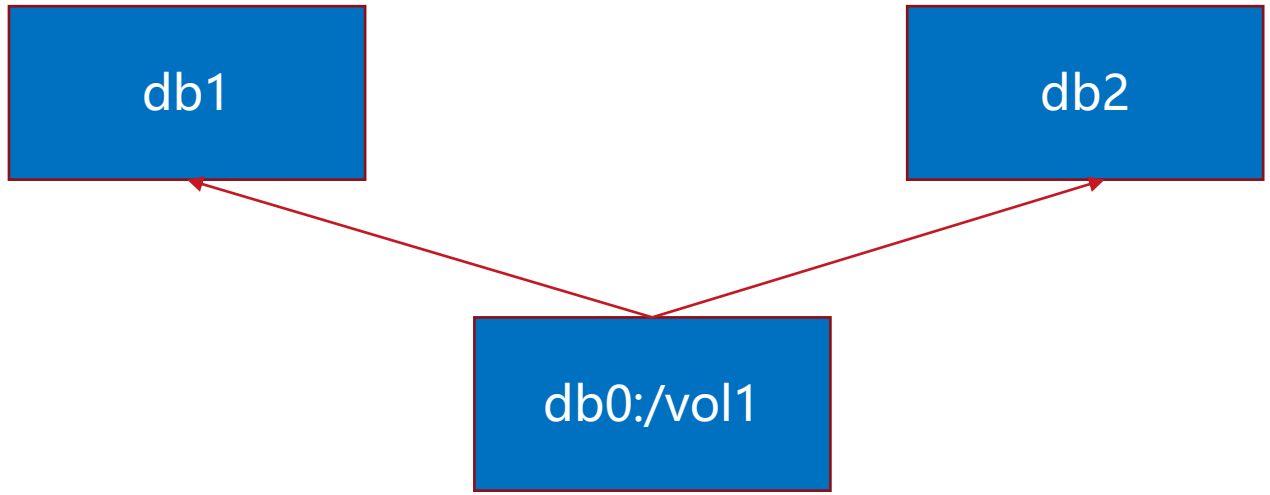
本章目录

8.2 数据卷容器

数据卷容器：

如果你有一些持续更新的数据需要在容器之间共享，最好创建数据卷容器。

数据卷容器，其实就是一个正常的容器，专门用来提供数据卷供其它容器挂载的。



示例

- 创建数据卷容器：

docker run -itd --name db0 centos -v /vol1 -v /vol2 /bin/bash

(默认存放在/var/lib/docker/)

docker run -itd --name db1 centos --volumes-from db0 /bin/bash

docker run -itd --name db2 centos --volumes-from db0 /bin/bash

- 验证数据卷映射：

docker stop db0

docker rm db0

嘉为教育

谢谢聆听

Thanks