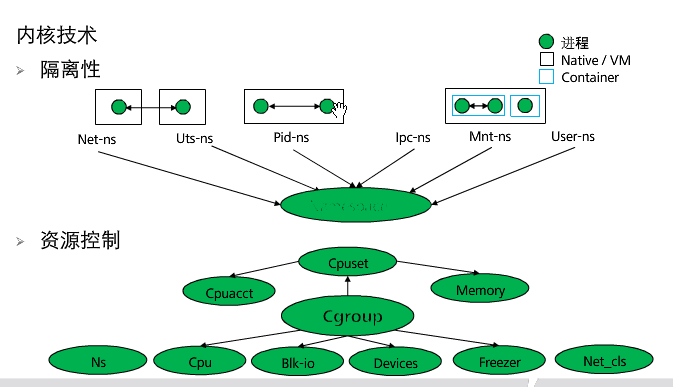
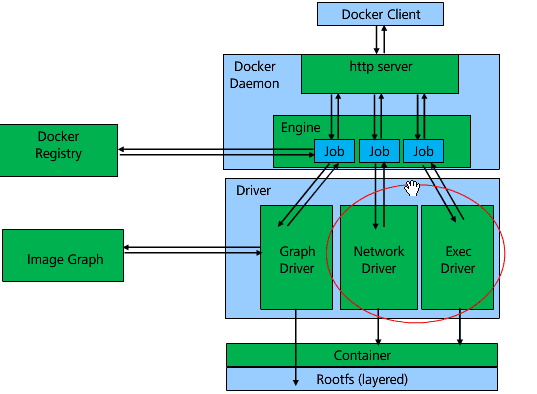
Docker基础原理

1. 背景-优势
2. 轻量化（2）资源隔离与控制（3）构建与部署效率（4）可移植性
3. 原理



1. 原理流程



1. 基础-容器

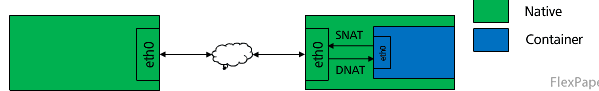
容器：独立运行一个或一组进程的运行态环境

进程：环境和资源隔离的核心对象

操作----运行：docker run –d –args image:tag $Command

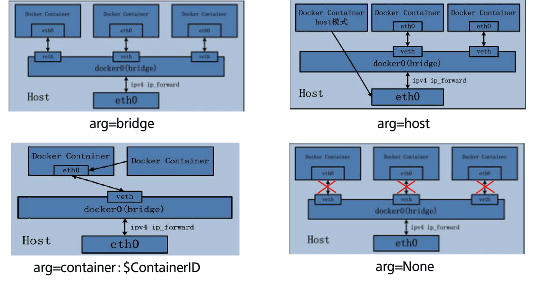
访问：docker exec –ti $ContainerID $Command

启停：docker start/stop/rm $ContainerID

通信

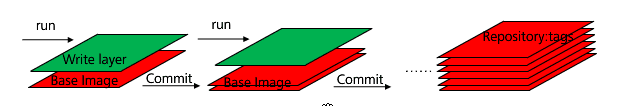
1. 网络模式

docker run –net=$arg Repository:tag $Command



1. 镜像

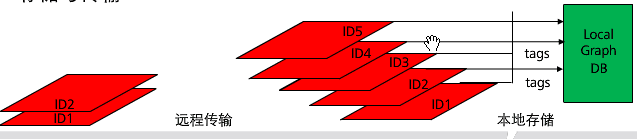
镜像：提供进程可执行的环境操作类集合



docker commit $ContainerID Repository:tag

docker bulid –t Repository:tag .(.代表Dockerfile所在目录)

存储于传输：

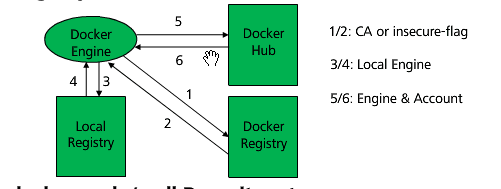


1. 仓库

Docker仓库

Hub Public/Private registry

Repository(仓库)



docker push/pull Repository:tag

docker push 10.67.164.106:5001/test/suse:latest

1. 资源控制
2. CPU配额

指定使用核

docker run –d - -（两个-）cpuset=0 10.67.164.106:5001/test/limit:v2.0 #指定运行核数

1. 指定cpu占用份额（share）

docker run –d –c=1024 10.67.164.106:5001/test/limit:v2.0 (share 2/3)

docker run –d –c=512 10.67.164.106:5001/test/limit:v2.0 (share 1/3)

1. 内存控制

Docker run –d –m 128m 10.67.164.106:5001/library/oss3.0:marnode/usr/sbin/sshd –D

1. 实践1-demo
2. 制作镜像（docker file）

FROM 10.67.164.106:5001/bsp/oss3.0:new #引援基础镜像

ADD python2.6.8/opt/ #拷贝文件到容器

WORKDIR/opt #指定后续命令工作目录

RUN[“/bin/bash”,”-c”,”/opt/install.sh”] #执行脚本或命令

ENTRYPOINT/usr/sbin/sshd –D #指定启动进程

1. 部署私有仓库

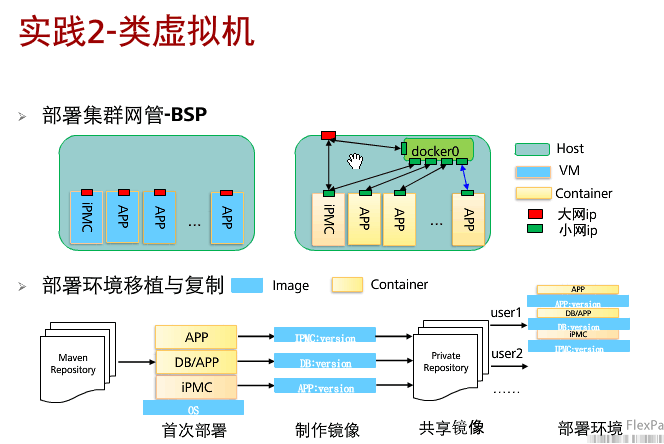
docker pull 10.67.164.106.5001/library/registry/:v3.0#拉取私有仓库服务镜像

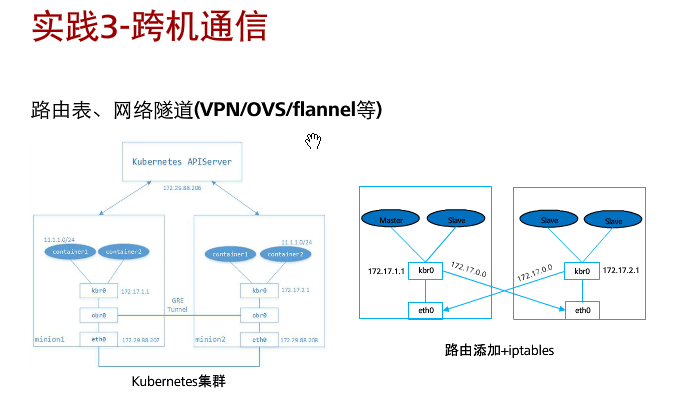
docker run –d –p $server\_port:5000 - -restart=always - -name registry

10.67.164.106.5001/library/registry/:v3.0 #运行私有仓库服务

docker run –d –p 8090:8080 –e REG1=http://$server\_port/v1/10.67.164.106.5001/docker-registry-ui

#运行私有仓库简易UI服务





1. 实践建议
2. 微服务化

单个进程运行单一进程

使用挂载或容器卷

1. Dockerfile

精简baseimage

同类型命令在相同层创建

1. 后期工作
2. 模拟HA部署场景

网络配置与Native/VM不同

NAT映射漂移问题

1. 测试环境

功能验证

统一化部署与管理

Docker发展过程

1.容器和Docker  
容器：在linux中，**容器技术是一种进程隔离的技术**，应用可以运行在每个相互隔离的容器中。  
容器与虚拟机的区别：在容器中，各个应用共用一个kernel  
Docker：Docker是一家公司，在13年之前公司名叫 dotCloud，Docker仅仅是一个容器管理的产品。在13年，将Docker开源，Docker风靡全球，公司也更名为Docker。  
2.容器技术的演进  
1979年，有了容器技术的雏形，root技术的引进开启了进程隔离技术  
2000年，FreeBSD Jails将计算机分为多个独立的小型计算系统。  
2006年，谷歌 Process Containers技术，在进程隔离的基础上，进行了计算资源的限制  
2018年，**LXC，作为第一个完整的容器管理工具**  
2013年，LMCTFY实现了linux应用程序的程序化，成为libcontainer的重要组成部分。  
2013年，Docker最初使用的是LXC，后来被替换成libcontainer   
3. Docker技术的迅猛发展  
在开源之后迎来迅猛的发展，在现在也保持着迅猛的发展势头   
4. Docker技术发展迅猛的原因总结：  
1.应用架构正在发生变革---微服务化  
在互联网时代，为了实现更快的开发迭代和更好的弹性伸缩，互联网应用不在采用传统的3层架构，而是采用微服务的方式。  
2.基础架构系统也在发生变革---虚拟化、混合云  
从硬件服务器到虚拟机到企业私有云，从本地数据中心到栽培数据中心再到公有云。  
5.他山之石，可以攻玉【容器技术的作用】  
一次封装，多次部署，随时迁移，不需要关注底层环境   
6．Docker定义的标准+服务应用  
基础设施标准化（Docker Engine）：有了Docker Engine，可将Docker容器跑起来  
应用交付的标准化（Docker Image）:提供了一套应用快速打包为轻量级Docker Image的方法。开发人员在代码完成之后，可以将其打包为镜像；  
运维管理的标准化（Docker container）:运维人员不在需要将应用准备系统、运行环境、组件和基础软件包,容器时代，应用都运行在一个个的Docker container中。标准运维将关注容器，而不是复杂的系统环境。  
分发部署标准化（Docker Registry）：指的是容器化之后不同版本的应用镜像都存储在镜像仓库中。运维人员可以将镜像仓库中特定版本的镜像一键部署。

7.增长速度

容器化应用 460K；容器镜像下载 4.1billion

容器技术精髓剖析

1. Namespace----网络、通信、文件等的隔离

namespace的六项隔离UTS---主机名与域名

IPC---信号量、消息队列和内容共享（隔离容器间的通信）

PID---进程编号

NetWork---网络设备、网络栈、端口

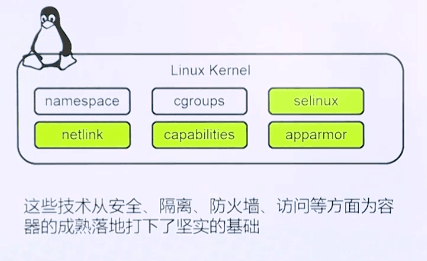
Mount---文件系统

User---用户和用户组（用户权限的隔离）

1. cgroups----为容器设置系统资源的限额和度量

包括CPU Memory IO等，当容器需要申请更多的内存时就会触发cgroup内存检测，如果用量超过cgroup的限额将会拒绝内存申请

1. 其他相关Linux Kernel技术



Selinux和 apparmor可以增强对容器的访问控制

Capabilities的实现在于将超级用户的权限分隔成了各种不同的capabilities权限

Netlink可以完成docker容器的网络环境配置和创建

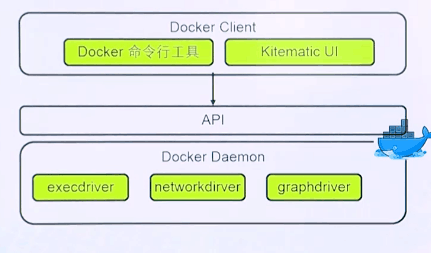
1. 容器管理

Lxc和libcontainer。Lxc是第一个容器管理技术，通过lxc可以方便的启动、创建和停止一个容器，操作容器中的应用。

2008年lxc，反向定义了一组接口标准，libcontainer并不是为了调用底层的namespace这些技术来设计的，而是定义了一些标准，为docker的实现跨平台使用提供可能

1. Docker

架构：client-server



1. Docker技术原理

Docker Daemon实际上是通过lxc、libvirt这些容器管理技术来实现容器管理操作的

Docker的三个重要组件：execdriver networkdriver和graphdriver

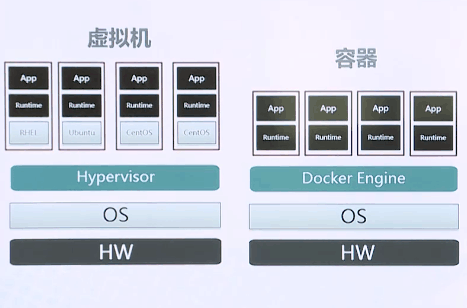
Execdriver定义了容器定义的配置信息，libcontainer拿到这些信息之后将会调用底层的namesapce和cgroups等来完成容器的创建和管理；

Networkdriver完成Docker容器的网络环境的配置，IP地址防火墙策略等；

Graphdriver主要负责对Docker容器镜像的管理；

Docker核心技术

1. VM vs Docker



虚拟机用hypervisor，容器用Docker Engine，容器的运行是不需要安装虚拟机的操作系统的。

Docker容器是共用一个kernel的，虚拟机有更好的隔离性但是



1. 容器、镜像和仓库

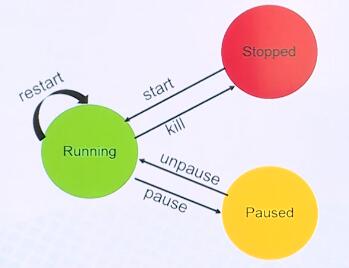
容器是承载相关应用的载体

镜像—当一个容器预先安装了特定的应用之后就可以将其打包成镜像，当需要应用的时候就可以下载镜像

仓库：存放镜像的地方

1. Docker容器的状态

和虚拟机的运行关机挂起是相似的



1. Docker镜像

当把一个镜像下载到本地之后就可以使用docker run命令启动一个基于镜像的容器，当对容器进行修改之后，也可以将容器commit回去，这就会生成一个新版本的镜像；

Docker镜像是一个层级结构的文件系统，镜像最上层往往是可写的，存储着已经运行的镜像的修改信息。

可以使用Dockerfile来build镜像，这种方式更加干净透明

1. 仓库

Dockerhub官方镜像仓库，使用PULl命令。还可以有私有仓库pull和push命令都可以产生Docker

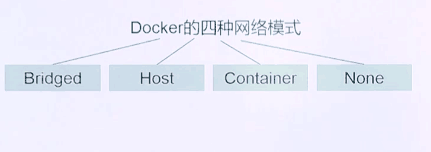
1. Bulid Ship Run

首先用Dockerfile将代码Build成一个镜像--🡪push到私有仓库，可以到任何地方连接到镜像仓库，下载了，run起来。

1. Docker数据卷

数据卷存在的意义：容器运行时可以使用数据卷中的文件；数据卷可以在多个容器间共享；存取容器运行过程中产生的数据；方便主机对容器数据的访问；

1. Docker网络



Bridged：容器可以与主机上的容器，主机，外部通信

Host；只能与主机

Container:只能与容器通信

比较常用的是Bridged模式，安装完Docker之后，主机会生成一个docker网桥，每个容器有自己的虚拟网卡，通过网桥连接到外部的物理网卡。

Docker的平台架构

1.容器编排（orchestration）

这里的编排泛指广义的编排，用于管理容器下面的主机，管理容器以及日期之间的逻辑关系，即为我们所知的应用架构。

集群管理：【关键词：配置管理、资源视图、节点增删、高可用】

容器调度：【关键词：容器部署、调度策略、互斥】

故障恢复：【关键词：主机检查、容器检查】

应用编排：应用是一个个细微的容器来组成，每一个容器之间具有一定的逻辑关系，我们需要使用简单明了的编排语句将这些容器关联起来，比如容器之间的端口访问，容器的启动顺序等，从而实现整个应用的逻辑架构。

2.编排主流的3大工具

Swarm: Docker 2014年发布 内置于Docker，和compose一起使用

Mesos: Apache 2007年发布 一般会结合marathon、Aookeeper一起使用

Kubernets: Google 2014年发布 结合Etcd一起使用

3.负载均衡和服务发现

负载均衡：请求到达负载均衡器之后，负载均衡器平均的分配到后面的容器上。

常用的负载均衡的技术：haproxy,LVS,F5,Nginx

服务发现：服务发现会自动将容器的配置信息上传至配置中心（config center），包含了容器的IP,端口、对外的域名等。

常用的服务发现的技术：Etcd,Zookeeper,Consul

负载均衡器后周期性的从配置中心获取配置信息，并且将容器加入到相关的负载均衡访问j架构中

4.日志管理

日志：包含主机、编排工具、日志、容器、容器中的应用等等相关的日志

对日志处理平台的要求：集中化、海量存储、灵活过滤、快速查询、伸缩性架构、高可用、强大的UI

日志管理软件：ELK ELK包含了3个组件:

Logstash：用于收集各种各样的日志

ElasticSearch:主要用于存储和搜索日志

Kbana:用于界面展示的管理工具

5.docker监控

包括了：主机，镜像，容器，应用几个维度进行监控

常用的监控工具：Zabbix ,Nagios,cAdvisor,Datadog,Scout

WANL原则：Watching Answer Notify Track

6.Docker平台架构

平台层：基础平台：集群管理，网络存储，镜像管理，容器调度，应用编排

能力层：集成功能：弹性架构负载均衡，性能监控日志管理，用户中心租户管理，持续支付版本控制

应用层：web UI和API

7.常用工具

编排：Swarm,Kubernetes,Mesos

集群管理：Etcd,Zookeeper,Haproxy

持续集成：Jenkins,GitLab,Registry

数据库中有：redis,Mysql,mongoDB

日志和配置管理中有：Logstash,Chef,puppet

监控里：ZABBIX,Nafios,sysdig

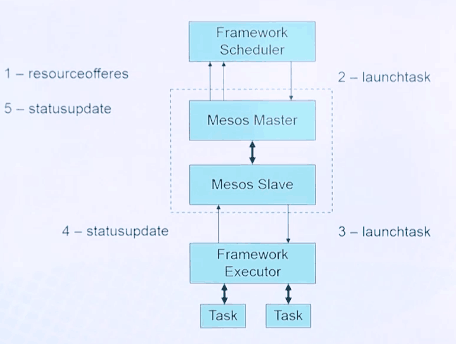
Docker平台的对比

Docker的3大编排工具：Mesos,Kubernetes,Swarm

1.Mesos介绍

两个重要角色，一个是Slave,安装集群节点上的，在一个是Master,作为集群的管理节点，slave会将节点的资源使用情况周期性的报告给我们的master节点。

Mesos配合架构的FrameWork进行资源调度，过程如下：



master会将计算机节点的资源使用情况周期性的报告给我们的Framework Scheduler

Framework Scheduler进行调配后，下发部署的任务给集群节点

节点上的framework Executor获取任务进行容器的部署（在容器的架构中，Executor就是容器的引擎）

节点会将部署结果反馈给master

Master会更新主机资源的状态给Framework Scheduler

Mesos+Marathon+Zookeeper构建Docker平台

Marathon是一个可以调用Docker引擎的framework，可以将容器按照一定的调度策略部署到合适的主机上。

Zookeeper的作用是保证Marathon和 Mesos来管理节点的高可用性，即当master节点宕机之后，可以快速的选取出新的master节点，从而不影响逻辑架构。Zookeeper本身是一个分布式的高可用的架构

2.kubernetes的介绍（Goolge的开源容器管理项目）

重点**：mesos中的最小的单元是容器，但是kubernetes的最小的单元是Pod**。**容器被封装在Pod中，一个Pod中可以存放一个或者多个容器。**

设计Pod的目的是：将需要紧密联系的Docker容器放置在一个独立的空间内（Pod）。

kubernetes整体架构：在部署 KVS（？）之前，需要先部署Etcd作为集群的管理工具

KVS 中有2个重要的角色：minion（普通节点）和master（管理节点），minion节点上的kubectl会结合etcd和？将信息汇总到master节点中。Master使用schedual会按照一定的策略会将Pod调度到各个minion上，使用replication和ctrl来控制Pod的数量和伸缩，使用service来控制Pod的Ip和处理Pod的负载均衡。

实现了平台的资源调度，部署启动，服务发现，运行监控，错误处理，扩容缩容等功能

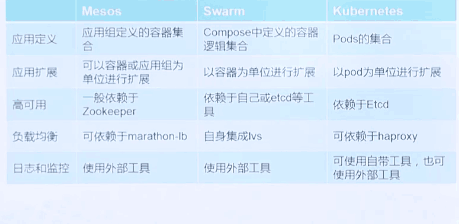
3.Swarm的介绍（Docker公司自己的容器管理工具）

1.12版本之后，Swarm已经封装进Docker引擎中，并且自带服务发现的功能

2个重要角色：manager和worker node

Swarm1.12之后，自带服务发现的功能，可以做到manager node的高可用性。Swam也内置了负载均衡的技术，使用的是LVS，但是依然支持和其他服务发现的工具 比如etcd

4.Docker平台对比



Docker 生态圈和企业应用案例

1.应用场景：快速交付与CICO

企业应用的开发上线流程一般是：代码、构建编译、测试、发布、部署

遇到的问题：可能因为环境的问题导致上线延迟，测试不通过等。

快速交付：Docker，通过Docker可以大大的提高环境交付的质量和速度，开发人员写好代码之后，交付的不在是一大堆的部署文档，而是一个个的镜像存储到镜像仓库中。在测试环境、预生产环境以及生产环境将镜像仓库中的镜像拉取出来即可。保证部署出来的所有应用都是标准的、统一的。即为实现了应用的快速交付。

CICD：持续集成和持续部署（Constant Integration Constant Deployment）当我们的代码更新时，开发人员可以构建一个新的镜像版本到镜像仓库中，运维人员可以快速的将我们的镜像应用到测试环境、预生产环境以及生产环境。甚至可以通过金KISS实现整个更新的自动化，从而实现了持续集成持续部署，实现了应用开发环境的快速迭代。

2.应用场景：云间迁移

应用容器化之后，对底层环境的要求将大大的降低，应用可以实现从本地数据中心到ALS，阿里云、公有云等迁移

3.应用场景：弹性扩展

企业应用容器化之后，应用的扩展就是拉取镜像部署更多容器的简单的过程，我们可以部署相关的监控系统，当发现应用访问慢或者是资源紧张的时候，在弹性扩展的策略下，应用会自动增加相应的容器实例，从而减轻应用访问的压力。当集群中的主机资源不足的时候，还可以使用IaaS接口，自动的增加主机的数量，以便于创建更多的Docker容器。

4.应用案例：平安Padis平台（2014）京东618（基于openstack和Docker）、天猫双十一

开发、测试部门对运维的交付效率提出了越来越高的要求，运营部门对应用的故障恢复，动态伸缩提出了越来越高的要求。

5.Docker巨大生态势能

从基础架构领域，操作系统领域、网络、存储、安全、安全、监控、日志等方面，越来越多的公司卷入到Dockers的发展潮流当中。

6.基于Docker的产品：包括容器公有云服务，容器私有云服务。

红帽openshift、阿里云容器服务、Azure容器服务、网易蜂巢、道客云、有容云、希云、时速云等

