目录

[1技术类型 1](#_Toc529350559)

[2 Docker 1](#_Toc529350560)

# 1技术类型

1、Iaas(Infrastructure as a Service) 基础设施即服务，通过Internet向用户提供存储、数据库、cpu、网络、GPU等资源。

2、Saas(Software as a Service) 软件即服务，通过internet(云)向用户提供应用程序的服务。

3、PAAS平台即(Platform-as-a-Service：平台即服务)，把应用服务的运行和开发环境作为一种服务提供的商业模式

IaaS主要提供了虚拟计算、存储、数据库等基础设施服务，SaaS为用户提供了基于云的应用，PaaS则为开发人员提供了构建应用程序的环境。

# 2 Docker

## 1 基本概念

特性:

更高效的利用资源(不需要额外的虚拟化管理软件)

更简单的更新管理

更方便的迁移部署

更快速的启动

更容易扩展

传统虚拟化方式:

虚拟出所需的硬件资源之后，在其上运行一套完整的操作系统，然后在该系统上运行所需的应用程序。



Docker: 容器虚拟化

不需要虚拟出硬件资源，容器的应用程序直接运行于宿主内核，容器内没自己的内核，更轻便。



|  |  |  |
| --- | --- | --- |
| 特性 | 容器 | 虚拟机 |
| 启动 | 秒级 | 分钟级 |
| 硬盘使用 | MB | GB |
| 性能 | 接近原生 | 若于原生 |
| 系统支持量 | 几千个容器 | 几十个虚拟机 |

## 2 三个概念

镜像images： 镜像包含操作系统完整的root文件系统。

分层存储框架。镜像不像ISO打包文件那样，镜像是一组文件。

容器

仓库

## 3.基本指令

拉取镜像: docker pull image\_name

查看已有的镜像:docker images

查看帮助:docker help

查看镜像列表:docker search nginx

创建并启动一个容器: docker run –name=container\_name images

启动一个容器:docker start container\_name/container\_id（已经创建的容器）

进入一个容器:docker attach container\_name

停止容器:docker stop container\_id

退出:ctrl+p

导出镜像: docker save –o image\_name.tar image\_name

删除镜像:docker rmi image\_name

基于创建好的容器自定义镜像:docker commit –m “con\_name” –a”author” con\_id image\_name

创建一个容器的同时进入这个容器 docker run –it –name=con\_name images

放入后台运行: docker run –d –p image\_name

启动一个容器:基于一个镜像新建并启动；启动已存在的终止的容器

载入镜像:sudo docker load --input ubuntu14.04.tar

查看docker 容器的日志:docker logs container\_name/container\_id

-f 跟踪实时日志 –t –since= –until

--tail=xxx 从日志末尾显示xxx行的日志

查看docker 的网络: docker network ls

创建一个docker网络: docker network create –d bridge

--subnet=192.168.0.0/24

--gateway=192.168.0.120

-- ip-range=192.168.0.0/24 my-docker

运行dockerfile并给dockerfile创建的镜像建立名字: docker build –t mysql:3.5.34 ‘pwd’

docker build –t second:v1.0 .注意最后有个点，代表使用当前路径的Dockerfile进行构件

-t 指定镜像名称 -f 指定Dockerfile的路径

Dockerfile 是一个文本格式的配置文件。

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指令

FROM 基础镜像信息 可以使用多个FROM，如果在同一个Dockerfile中创建多个镜像，可以使用多个FROM（每个镜像一次）

FORM<image> FROM Ubuntu:trusty

#MAINTAINER 指定维护者信息

#MAINTAINER username<xxx.com>

#ENV指定一个环境变量，会被后续RUN指令使用，并在容器运行时保持。

#ENV <key> <value>

ENV DEBIAN\_FROMEND noninteractive（设置环境变量）

#RUN 镜像的操作指令 image支持的命令

#ADD ADD [source directory or URL] [destination directory]

它的基本作用是从源系统的文件系统上复制文件到目标容器的文件系统。可以用“\”换行

复制指定的<src>到容器的<dest>中，<src>可以是Dockerfile所在的目录的一个相对路径；可以是URL，也可以是tar.gz（自动解压）

#COPY

复制本地主机的 <src> （ 为 Dockerfile 所在目录的相对路径）到容器中的 <dest> （当使用本地目录为源目录时，推荐使用 COPY）

#EXPOSE

格式：EXPOSE <port>  [ <port> ...]

告诉Docker服务端暴露端口，在容器启动时需要通过 -p 做端口映射

#WORKDIR

格式：WORKDIR /path/to/workdir

为后续的 RUN 、 CMD 、 ENTRYPOINT 指令配置工作目录。（可以使用多个 WORKDIR 指令，后续命令如果参数是相对路径， 则会基于之前命令指定的路径）

#CMD 设置容器启动时执行的操作 只能存在一条(多个CMD命令存在的话只执行最后一个CMD 命令，因此只需要一个CMD命令)，调用方式有三种

# 方式一，运行一个可执行文件，并提供参数(like an exec, this is the preferred form)

CMD ["executable","param1","param2"]

# 方式二，利用”/bin/sh -c”去执行， (as a shell)

CMD command param1 param2

# 方式三，作为ENTRYPOINT的默认参数

CMD ["param1", "param2"]

在使用docker run imagename command新建并启动容器的时候，command会替换dockerfile里的CMD命令，如上面我们创建的docker镜像，如果后面输入了hello docker，则不会输出hello world了，本来dockerfile里面指定了输出hello world。

$ docker run hello\_docker echo "hello docker"

hello docker

$ docker run hello\_docker

hello world

该操作可以是执行自定义脚本，也可以是执行系统命令。

#ENTRYPOINT 功能和CMD 一样，区别在于ENTRYPOINT后面的携带的参数不会被docker run 提供的参数覆盖，而CMD 会被覆盖

CMD指令指定的容器启动时命令可以被docker run指定的命令覆盖，而ENTRYPOINT指令指定的命令不能被覆盖，而是将docker run指定的参数当做ENTRYPOINT指定命令的参数。

docker run –d –P training/webapp python app.py

-d 后台运行

-P 将容器内部使用的网络端口映射到使用的主机上

-p 5000:5000

docker run –d –p 127.0.0.1:5000:5000 training/webapp python app.py

网络端口的快捷方式

docker port 可以查看某个容器的确定端口映射到主机上的端口号

Dockerfile 创建ubuntu的新用户，并以新用户登录。

RUN useradd --create-home --no-log-init --shell /bin/bash mynewuser

RUN adduser mynewuser sudoRUN echo 'mynewuser:mynewpassword' | chpasswd

USER mynewuser

WORKDIR /home/mynewuser

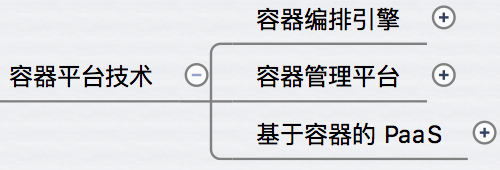
## 4.docker 理论

### 4.1Docker的 技术组成

容器生态系统:容器核心技术、容器平台技术、容器支持技术。

容器核心技术:让container在host上运行起来

容器平台技术:让容器作为集群在分布式环境中运行



容器编排引擎 kubernetes docker swarm

基于容器的应用一般会采用微服务架构。在这种架构下，应用被划分为不同的组件，并以服务的形式运行在各自的容器中，通过 API 对外提供服务。为了保证应用的高可用，每个组件都可能会运行多个相同的容器。这些容器会组成集群，集群中的容器会根据业务需要被动态地创建、迁移和销毁。

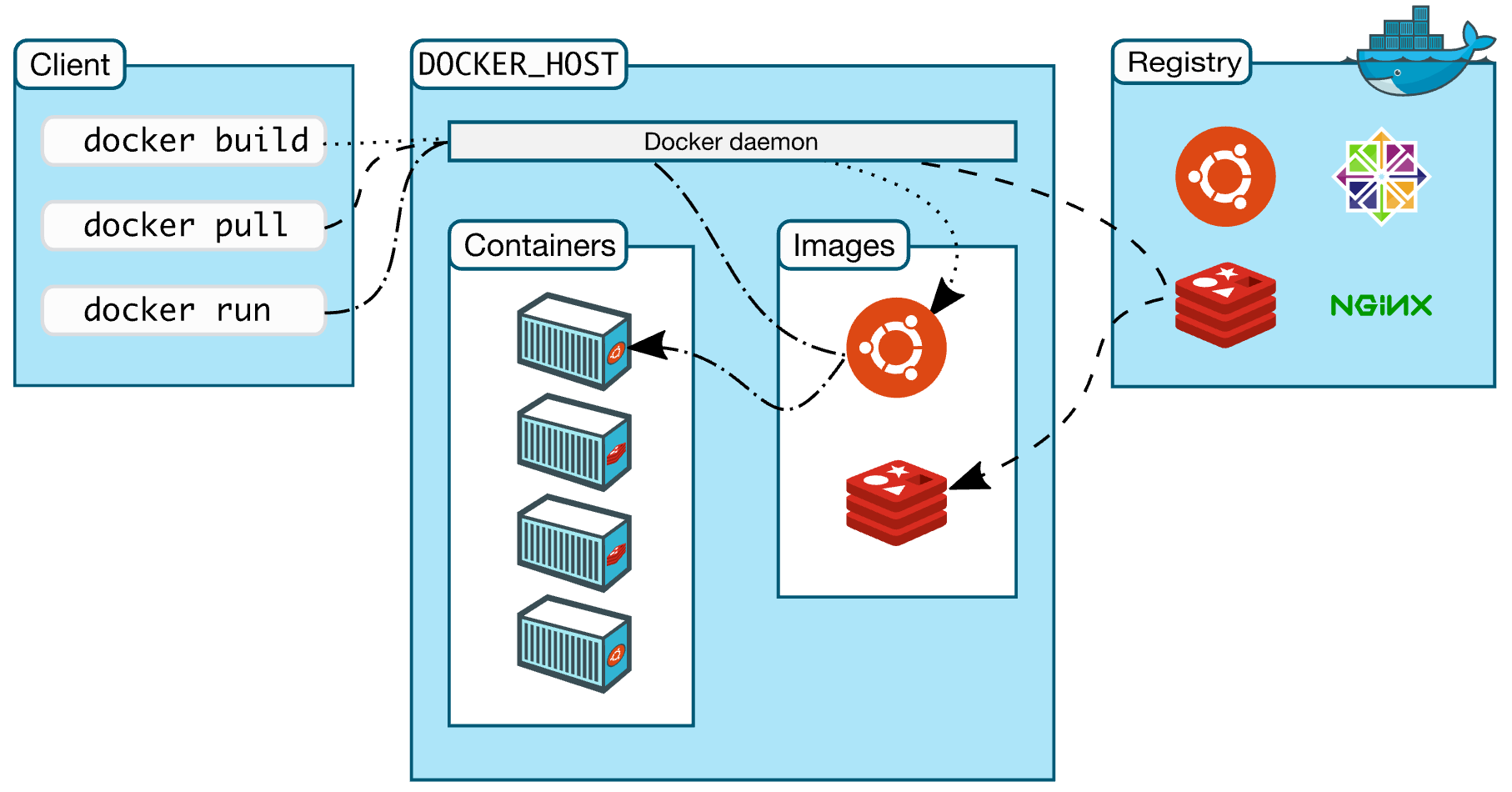
大家可以看到，这样一个基于微服务架构的应用系统实际上是一个动态的可伸缩的系统。这对我们的部署环境提出了新的要求，我们需要有一种高效的方法来管理容器集群。而这，就是容器编排引擎要干的工作。

所谓编排（orchestration），通常包括容器管理、调度、集群定义和服务发现等。通过容器编排引擎，容器被有机的组合成微服务应用，实现业务需求。

Deis、Flynn 和 Dokku 都是开源容器 PaaS 的代表。

容器支持的技术:网络、数据、服务状态等

### 4.2 Docker的基本框架图



5 docker数据管理

5.1 数据卷 Data Volumes 容器内的数据直接映射到localhost

数据卷可以在容器之间共享和重用，容器间的传递数据将会变得更高效和方便

数据卷会一直存在，直到没有容器使用后可以安全的删除

数据卷更新不影响镜像

数据卷的修改会立马生效，在容器内和本地

创建数据卷: docker volume create –d local test

绑定数据卷

Sudo docker run –d –P –name web –v /webapp:/opt/webapp training/webapp python app.py

5.2 数据卷容器 Data Volumes Containers

# 3 openStack

## 3.1 OpenStack 基本介绍

组件：

1 Horizon 控制台 web展示界面操作平台

2 Nove 计算 负责创建，调度，销毁云主机

3 Neutron 网络 负责实现SDN

4 Swift 对象存储 结构化存储数据

5 Cinder 块存储 提供持久化块存储，为云主机提供附加云盘

6 Glance 镜像 提供镜像服务 装机使用

7 Keystone 认证

8 Ceilometer 计费

KVM(Kernel-based Virtual Machine) 开放虚拟化技术

是一个开源的系统虚拟化模块，需要硬件支持。

# 附录

## Dockerfile 模板

FROM centos

MAINTAINER yaolin

COPY jdk1.7.0\_79 jdk1.7.0\_79

ADD websocket.jar app.jar

ENV JAVA\_HOME=/jdk1.7.0\_79

ENV PATH=$JAVA\_HOME/bin:$PATH

ENV CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

EXPOSE 8080

ENTRYPOINT ["java","-jar","/app.jar"]