**docker技术与实践指南**

****

**2018-2-18  
何锡捷**

**目录**

[1. 文档主题 3](#_Toc22676840)

[2. docker简介 3](#_Toc1025044361)

[3. 实战docker 3](#_Toc806759093)

[3.1. 安装虚拟机(略) 3](#_Toc2135812540)

[3.2. 安装docker 3](#_Toc1412200175)

[3.2.1. Ubuntu 安装 Docker CE 3](#_Toc859074581)

[3.2.2. 准备工作 4](#_Toc933924086)

[3.2.3. 使用 APT 安装 6](#_Toc504137479)

[3.2.4. 使用脚本自动安装 8](#_Toc1215622138)

[3.2.5. 启动 Docker CE 8](#_Toc1949339455)

[3.2.6. 建立 docker 用户组 8](#_Toc537701553)

[3.2.7. 测试 Docker 是否安装正确 9](#_Toc538814695)

[3.2.8. 镜像加速 10](#_Toc2067523113)

[3.2.9. 参考文档 12](#_Toc428068084)

[3.3. 创建docker网络 12](#_Toc470070338)

[3.4. 安装mysql 12](#_Toc2027317100)

[3.5. 创建数据库 13](#_Toc1142956398)

[3.6. 安装redis 13](#_Toc426958771)

[3.7. 创建应用 13](#_Toc1153199570)

[3.8. 部署应用 13](#_Toc785258815)

[3.9. 应用测试 14](#_Toc1557892890)

[4. why docker? docker原理 14](#_Toc1385178006)

[4.1. 为什么使用docker 14](#_Toc1964013362)

[4.1.1. docker与虚拟机对比 14](#_Toc201437097)

[4.1.2. 持续集成/交付 15](#_Toc1119061607)

[4.2. docker原理 15](#_Toc406648423)

[4.2.1. Docker简介 15](#_Toc1247080607)

[4.2.2. docker架构 16](#_Toc243367129)

[4.2.3. docker对象 16](#_Toc1462473215)

[4.2.4. 例子 17](#_Toc1836984590)

[4.2.5. 底层技术 17](#_Toc1975094858)

[4.2.6. Docker network 19](#_Toc1764546727)

[4.3. docker常用命令详解 20](#_Toc2135159266)

[4.3.1. 容器管理 20](#_Toc1170042292)

[4.3.2. 镜像管理 21](#_Toc393046065)

[5. Dockerfile 22](#_Toc265516283)

[6. Docker compose 25](#_Toc61149915)

[7. Docker Swarm 26](#_Toc1249438139)

[7.1. Stack 26](#_Toc1211701807)

[7.2. Service 27](#_Toc484845748)

[7.3. Config 27](#_Toc1249529918)

[7.4. Secret 28](#_Toc606747813)

[8. GluseterFS 31](#_Toc1358137135)

[8.1. 安装 31](#_Toc607143982)

[8.2. 使用 33](#_Toc1574098577)

[9. Docker集群环境搭建案例 35](#_Toc1023736246)

[10. 常见问题 36](#_Toc296106758)

# 文档主题

本文旨在介绍docker基础与实践，可作为docker入门培训教程。

# docker简介

* Docker 是一个开源的应用容器引擎，基于 [Go 语言](http://www.runoob.com/go/go-tutorial.html" \t "/Users/hexijie\\x/_blank) 并遵从Apache2.0协议开源。
* Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。
* 容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。

# 实战docker

设想这样一个场景。我们有一个基于java的应用，需要依赖mysql，redis，现在我们需要将应用部署到测试环境。针对这个场景，我们来一起做一个实验。

## 安装虚拟机(略)

首先安装ubuntu16.04版本的虚拟机，确保可以远程连接到该虚拟机。

## 安装docker

### Ubuntu 安装 Docker CE

>警告：切勿在没有配置 Docker APT 源的情况下直接使用 apt 命令安装 Docker.

### 准备工作

#### 系统要求

Docker CE 支持以下版本的 [Ubuntu](https://www.ubuntu.com/server) 操作系统：

\* Bionic 18.04 (LTS)

\* Xenial 16.04 (LTS)

\* Trusty 14.04 (LTS) (Docker CE v18.06 及以下版本)

Docker CE 可以安装在 64 位的 x86 平台或 ARM 平台上。Ubuntu 发行版中，LTS（Long-Term-Support）长期支持版本，会获得 5 年的升级维护支持，这样的版本会更稳定，因此在生产环境中推荐使用 LTS 版本。

#### ubuntu配置国内源

如果你的源不是国内的，最好配置下，下载速度快！

编辑/etc/apt/sources.list，替换成如下内容

# deb cdrom:[Ubuntu 16.04 LTS \_Xenial Xerus\_ - Release amd64 (20160420.1)]/ xenial main restricted

deb-src http://archive.ubuntu.com/ubuntu xenial main restricted #Added by software-properties

deb http://mirrors.aliyun.com/ubuntu/ xenial main restricted

deb-src http://mirrors.aliyun.com/ubuntu/ xenial main restricted multiverse universe #Added by software-properties

deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main restricted

deb-src http://mirrors.aliyun.com/ubuntu/ xenial-updates main restricted multiverse universe #Added by software-properties

deb http://mirrors.aliyun.com/ubuntu/ xenial universe

deb http://mirrors.aliyun.com/ubuntu/ xenial-updates universe

deb http://mirrors.aliyun.com/ubuntu/ xenial multiverse

deb http://mirrors.aliyun.com/ubuntu/ xenial-updates multiverse

deb http://mirrors.aliyun.com/ubuntu/ xenial-backports main restricted universe multiverse

deb-src http://mirrors.aliyun.com/ubuntu/ xenial-backports main restricted universe multiverse #Added by software-properties

deb http://archive.canonical.com/ubuntu xenial partner

deb-src http://archive.canonical.com/ubuntu xenial partner

deb http://mirrors.aliyun.com/ubuntu/ xenial-security main restricted

deb-src http://mirrors.aliyun.com/ubuntu/ xenial-security main restricted multiverse universe #Added by software-properties

deb http://mirrors.aliyun.com/ubuntu/ xenial-security universe

deb http://mirrors.aliyun.com/ubuntu/ xenial-security multiverse

完成后执行`

sudo apt-get update

#### 卸载旧版本

旧版本的 Docker 称为 `docker` 或者 `docker-engine`，使用以下命令卸载旧版本：

$ sudo apt-get remove docker \

docker-engine \

docker.io

#### Ubuntu 14.04 可选内核模块

从 Ubuntu 14.04 开始，一部分内核模块移到了可选内核模块包 (`linux-image-extra-\*`) ，以减少内核软件包的体积。正常安装的系统应该会包含可选内核模块包，而一些裁剪后的系统可能会将其精简掉。`AUFS` 内核驱动属于可选内核模块的一部分，作为推荐的 Docker 存储层驱动，一般建议安装可选内核模块包以使用 `AUFS`。

如果系统没有安装可选内核模块的话，可以执行下面的命令来安装可选内核模块包：

$ sudo apt-get update

$ sudo apt-get install \

linux-image-extra-$(uname -r) \

linux-image-extra-virtual

#### Ubuntu 16.04 +

Ubuntu 16.04 + 上的 Docker CE 默认使用 `overlay2` 存储层驱动,无需手动配置。

### 使用 APT 安装

由于 `apt` 源使用 HTTPS 以确保软件下载过程中不被篡改。因此，我们首先需要添加使用 HTTPS 传输的软件包以及 CA 证书。

$ sudo apt-get update

$ sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

software-properties-common

鉴于国内网络问题，强烈建议使用国内源，官方源请在注释中查看。

为了确认所下载软件包的合法性，需要添加软件源的 `GPG` 密钥。

$ curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add -

# 官方源

# $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

然后，我们需要向 `source.list` 中添加 Docker 软件源

$ sudo add-apt-repository \

"deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu \

$(lsb\_release -cs) \

stable"

# 官方源

# $ sudo add-apt-repository \

# "deb [arch=amd64] https://download.docker.com/linux/ubuntu \

# $(lsb\_release -cs) \

# stable"

>以上命令会添加稳定版本的 Docker CE APT 镜像源，如果需要测试或每日构建版本的 Docker CE 请将 stable 改为 test 或者 nightly。

#### 安装 Docker CE

更新 apt 软件包缓存，并安装 `docker-ce`：

$ sudo apt-get update

$ sudo apt-get install docker-ce

### 使用脚本自动安装

在测试或开发环境中 Docker 官方为了简化安装流程，提供了一套便捷的安装脚本，Ubuntu 系统上可以使用这套脚本安装：

$ curl -fsSL get.docker.com -o get-docker.sh

$ sudo sh get-docker.sh --mirror Aliyun

执行这个命令后，脚本就会自动的将一切准备工作做好，并且把 Docker CE 的 Edge 版本安装在系统中。

### 启动 Docker CE

$ sudo systemctl enable docker

$ sudo systemctl start docker

Ubuntu 14.04 请使用以下命令启动：

$ sudo service docker start

### 建立 docker 用户组

默认情况下，`docker` 命令会使用 [Unix socket](https://en.wikipedia.org/wiki/Unix\_domain\_socket) 与 Docker 引擎通讯。而只有 `root` 用户和 `docker` 组的用户才可以访问 Docker 引擎的 Unix socket。出于安全考虑，一般 Linux 系统上不会直接使用 `root` 用户。因此，更好地做法是将需要使用 `docker` 的用户加入 `docker` 用户组。

建立 `docker` 组：

$ sudo groupadd docker

将当前用户加入 `docker` 组：

$ sudo usermod -aG docker $USER

退出当前终端并重新登录，进行如下测试。

### 测试 Docker 是否安装正确

$ docker run hello-world

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

d1725b59e92d: Pull complete

Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

(amd64)

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/

若能正常输出以上信息，则说明安装成功。

### 镜像加速

国内从 Docker Hub 拉取镜像有时会遇到困难，此时可以配置镜像加速器。Docker 官方和国内很多云服务商都提供了国内加速器服务，例如：

* [Docker官方提供的中国 ](https://docs.docker.com/registry/recipes/mirror/#use-case-the-china-registry-mirror)
* [阿里云加速器(需登录账号获取)](https://cr.console.aliyun.com/cn-hangzhou/mirrors)
* [七牛云加速器](https://kirk-enterprise.github.io/hub-docs/#/user-guide/mirror)

> 当配置某一个加速器地址之后，若发现拉取不到镜像，请切换到另一个加速器地址。

> 国内各大云服务商均提供了 Docker 镜像加速服务，建议根据运行 Docker 的云平台选择对应的镜像加速服务。

我们以 Docker 官方加速器 `https://registry.docker-cn.com` 为例进行介绍。

#### Ubuntu 14.04、Debian 7 Wheezy

对于使用 [upstart](http://upstart.ubuntu.com/) 的系统而言，编辑 `/etc/default/docker` 文件，在其中的 `DOCKER\_OPTS` 中配置加速器地址：

DOCKER\_OPTS="--registry-mirror=https://registry.docker-cn.com"

重新启动服务。

$ sudo service docker restart

#### Ubuntu 16.04+、Debian 8+、CentOS 7

对于使用 [systemd](https://www.freedesktop.org/wiki/Software/systemd/) 的系统，请在 `/etc/docker/daemon.json` 中写入如下内容（如果文件不存在请新建该文件）

{

"registry-mirrors": [

"https://registry.docker-cn.com"

]

*}*

> 注意，一定要保证该文件符合 json 规范，否则 Docker 将不能启动。

之后重新启动服务。

$ sudo systemctl daemon-reload

$ sudo systemctl restart docker

>注意：如果您之前查看旧教程，修改了 `docker.service` 文件内容，请去掉您添加的内容（`--registry-mirror=https://registry.docker-cn.com`），这里不再赘述。

#### 检查加速器是否生效

命令行执行 `docker info`，如果从结果中看到了如下内容，说明配置成功。

Registry Mirrors:

https://registry.docker-cn.com/

### 参考文档

\* [Docker 官方 Ubuntu 安装文档](https://docs.docker.com/install/linux/docker-ce/ubuntu/)

## 创建docker网络

docker network create nw-example

## 安装mysql

docker container run --name mysql01 --network nw-example -e MYSQL\_ROOT\_PASSWORD=123456 -p3306:3306 -d mysql

* --name给新创建的容器命名，此处命名为mysql01
* -e：环境变量，此处配置mysql的root用户的登陆密码
* -p：端口映射，此处映射主机3306端口到容器mysql01的3306端口
* -d：容器在后台运行.
* 最后一个mysql指的是mysql镜像名字

注：在执行container run命令的时候，如果本地没有镜像，则先从服务器下载镜像后再启动容易。

完成指令后使用命令连接mysql

mysql -h127.0.0.1 -uroot  -p

## 创建数据库

mysql -h127.0.0.1 -uroot  -p

> CREATE DATABASE IF NOT EXISTS springboot\_docker DEFAULT CHARSET utf8 COLLATE utf8\_general\_ci;

## 安装redis

docker container run --name redis01 --network nw-example -p6379:6379 -d redis

为了连接容器里的redis，宿主机需要安装redis客户端，使用如下命令安装:

$ sudo apt-get install redis-tools

安装完成后使用如下命令连接redis进行测试

redis-cli

## 创建应用

sudo docker build -t hnykx/springboot-docker-example:1.0 https://github.com/cj19881112/springboot-docker-example.git

## 部署应用

docker container run -it --name app --network nw-example -e "SPRING\_PROFILES\_ACTIVE=test" -p 8081:8080 hnykx/springboot-docker-example:1.0-SNAPSHOT

## 应用测试

1. 连接到mysql容器，添加如下两条数据

insert into user values(1, 'male', 'Jhon');

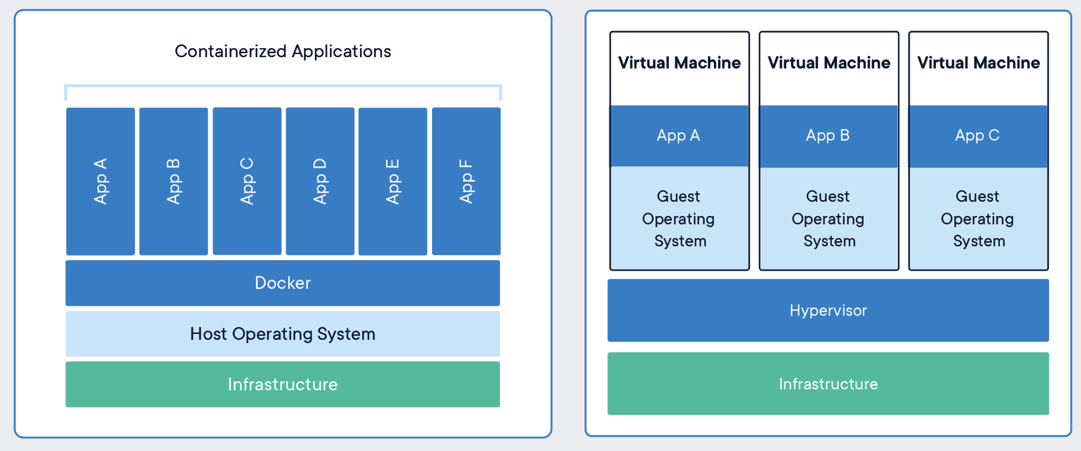
insert into user values(2, 'male', 'Lucy');

1. 访问192.168.1.151:8081

# why docker? docker原理

## 为什么使用docker

### docker与虚拟机对比



### 持续集成/交付

## docker原理

### Docker简介

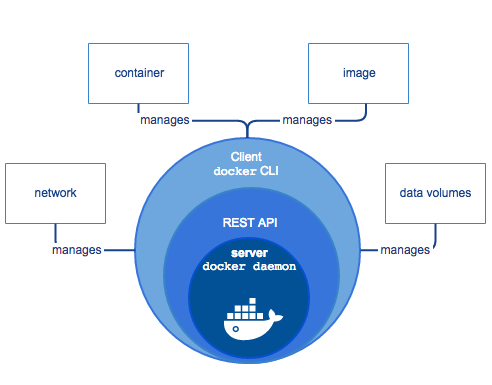
**Docker Engine**使用BS架构，包含如下组件:

* server组件，作为守护进程在后台运行（dockerd命令），可以使用一下命令查看帮助

man dockerd # 查看man手册

ps -ef|grep dockerd # 查看进程

* 一组REST api，可以通过http调用
* CLI组件(docker命令)



### docker架构

IMG_256

* **Docker deamon**

Docker deamon监听网络为外部提供api接口，docker deamon还可以与其他的docker daemon通信。

* **Docker Client**

平时主要使用cli来进行docker维护，比如docker run等命令。

* **Docker registries**

存储docker镜像。当使用docker pull或者docker run的时候会从docker registry下载镜像。默认使用的是Docker Hub(https://hub.docker.com/)

### docker对象

#### 镜像(image)

镜像是创建容器的只读模板。通常一个镜像会基于其他镜像定制。你可以使用Dockerfile定制自己的镜像。

#### 容器(container)

容器是可以执行的镜像。你可以启动、停止、创建、删除容器。可以将容器想象成轻量级的虚拟机。

### 例子

docker run -i -t ubuntu /bin/bash

可以使用上述命令创建一个基于ubuntu镜像的容器。当执行这个命令的时候，首先会从远程服务器下载ubuntu镜像，然后创建对应的容器并执行。上述命令的参数介绍如下：

-i：交互式运行

-t: 虚拟终端

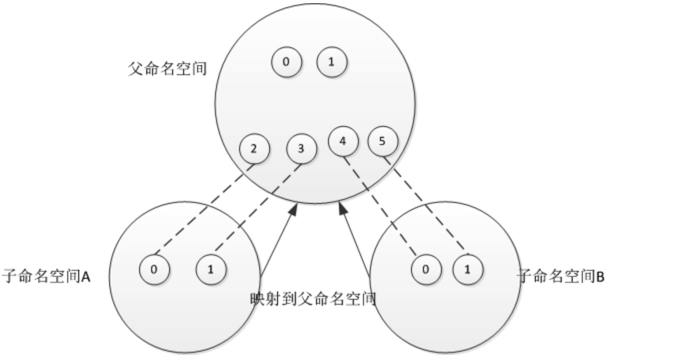
ubuntu: 镜像名称

/bin/bash：要执行的命令

### 底层技术

#### Namespaces

在Linux系统中，可以同时存在多用户多进程，那么对他们的运行协调管理，通过进程调度和进度管理可以解决，但是，整体资源是有限的，怎么把有限的资源（进程号、通信资源、网络资源等等）合理分配给各个用户所在的进程？Linux中提出了namespace机制，这是一种轻量级的虚拟化形式。



命名空间建立系统的不同视图， 对于每一个命名空间，从用户看起来，应该像一台单独的Linux计算机一样，有自己的init进程(PID为0)，其他进程的PID依次递增，A和B空间都有PID为0的init进程，子容器的进程映射到父容器的进程上，父容器可以知道每一个子容器的运行状态，而子容器与子容器之间是隔离的。

#### Cgroups

Cgroups是control groups的缩写，是Linux内核提供的一种可以限制、记录、隔离进程组（process groups）所使用的物理资源（如：cpu,memory,IO等等）的机制。最初由google的工程师提出，后来被整合进Linux内核。Cgroups也是LXC为实现虚拟化所使用的资源管理手段，可以说没有cgroups就没有LXC。

Cgroups可以做什么？

cgroups最初的目标是为资源管理提供的一个统一的框架，既整合现有的cpuset等子系统，也为未来开发新的子系统提供接口。现在的cgroups适用于多种应用场景，从单个进程的资源控制，到实现操作系统层次的虚拟化（OS Level Virtualization）。

cgroups提供了一下功能：

1.限制进程组可以使用的资源数量（Resource limiting ）。比如：memory子系统可以为进程组设定一个memory使用上限，一旦进程组使用的内存达到限额再申请内存，就会出发OOM（out of memory）。

2.进程组的优先级控制（Prioritization ）。比如：可以使用cpu子系统为某个进程组分配特定cpu share。

3.记录进程组使用的资源数量（Accounting ）。比如：可以使用cpuacct子系统记录某个进程组使用的cpu时间

4.进程组隔离（Isolation）。比如：使用ns子系统可以使不同的进程组使用不同的namespace，以达到隔离的目的，不同的进程组有各自的进程、网络、文件系统挂载空间。

5.进程组控制（Control）。比如：使用freezer子系统可以将进程组挂起和恢复。

#### Overlay

OverlayFS将单个Linux主机上的两个目录合并成一个目录。这些目录被称为层，统一过程被称为联合挂载。OverlayFS底层目录称为lowerdir， 高层目录称为upperdir。合并统一视图称为merged。当需要修改一个文件时，使用CoW(copy on write - 写时复制)将文件从只读的Lower复制到可写的Upper进行修改，结果也保存在Upper层。在Docker中，底下的只读层就是image，可写层就是Container。

例子：

mkdir lower upper work merged

echo "lower.aaaa" > lower/aaaa

echo "lower.bbbb" > lower/bbbb

echo "upper.bbbb" > upper/bbbb

echo "upper.cccc" > upper/cccc

mount -t overlay overlay -olowerdir=lower,upperdir=upper,workdir=work merged

可以看到，lower uppper目录整合了，且upper目录的bbbb文件覆盖了lower中的文件。

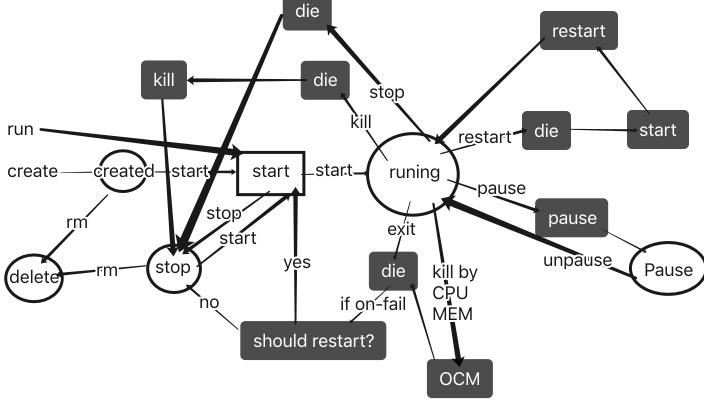
### Docker network

其他文档展开

## docker常用命令详解

### 容器管理

#### 容器状态变迁



#### 基本命令

**docker container**

Usage: docker container COMMAND

Manage containers

Commands:

  attach    Attach local standard input, output, and error streams to a running container

  commit   Create a new image from a container's changes

  cp         Copy files/folders between a container and the local filesystem

  create     Create a new container

  diff        Inspect changes to files or directories on a container's filesystem

  exec      Run a command in a running container

  export    Export a container's filesystem as a tar archive

  inspect   Display detailed information on one or more containers

  kill        Kill one or more running containers

  logs       Fetch the logs of a container

  ls          List containers

  pause     Pause all processes within one or more containers

  port       List port mappings or a specific mapping for the container

  prune     Remove all stopped containers

  rename    Rename a container

  restart     Restart one or more containers

  rm          Remove one or more containers

  run         Run a command in a new container

  start       Start one or more stopped containers

  stats       Display a live stream of container(s) resource usage statistics

  stop        Stop one or more running containers

  top         Display the running processes of a container

  unpause   Unpause all processes within one or more containers

  update     Update configuration of one or more containers

  wait        Block until one or more containers stop, then print their exit codes

### 镜像管理

Usage: docker image COMMAND

Manage images

Commands:

  build     Build an image from a Dockerfile

  history   Show the history of an image

  import   Import the contents from a tarball to create a filesystem image

  inspect  Display detailed information on one or more images

  load   Load an image from a tar archive or STDIN

  ls         List images

  prune   Remove unused images

  pull      Pull an image or a repository from a registry

  push    Push an image or a repository to a registry

  rm        Remove one or more images

  save    Save one or more images to a tar archive (streamed to STDOUT by default)

  tag       Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

Run 'docker image COMMAND --help' for more information on a command.

# Dockerfile

docker使用基于层的文件系统来构建镜像。通常，我们dockerfile来构建docker镜像。下面。

在第四章的例子中，我们使用到了dockerfile，我们来看看dockerfile的内容：

FROM lw96/docker-maven-aliyun:jdk-8 as builder

COPY src /app/src/

COPY pom.xml /app/

RUN cd /app \

&& mvn -Dmaven.test.skip=true clean package

FROM openjdk:8-jdk-alpine as spring-docker-example

VOLUME /tmp

COPY --from=builder /app/target/\*.jar /app.jar

RUN echo 'hosts: files mdns4\_minimal [NOTFOUND=return] dns mdns4' >> /etc/nsswitch.conf

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

dockerfile的每一行是一条指令，上面这个文件用到了这些指令：FROM、COPY、RUN、VOLUME、RUN、ENTRYPOINT。

FROM lw96/docker-maven-aliyun:jdk-8 as builder

FROM：执行基础镜像。通常，每个docker镜像都会基于某个镜像，上面这个文件的第一行表示使用lw96/docker-maven-aliyun:jdk-8作为基础镜像，其中lw96表示用户的名称，docker-maven-aliyun表示镜像的名称，jdk-8表示镜像的tag。行末的as builder说明这个文件分阶段编译，下个FROM出现前的命令组成阶段builder，后续的阶段可以使用builder来引用编译出来的镜像。

COPY src /app/src/

COPY pom.xml /app/

将上下文中的src目录与pom.xml拷贝到镜像的/app/目录下

RUN cd /app \

&& mvn -Dmaven.test.skip=true clean package

在容器中执行mvn编译操作，编译结果在容器的/app/target下

FROM openjdk:8-jdk-alpine as spring-docker-example

使用openjdk:8-jdk-alpine作为基础镜像

VOLUME /tmp

创建匿名卷，挂在到/tmp目录下(spring boot运行的时候在/tmp目录创建容器相关文件)

COPY --from=builder /app/target/\*.jar /app.jar

从builder阶段容器，拷贝/app/target/\*.jar到新容器的/app.jar

RUN echo 'hosts: files dns' >> /etc/nsswitch.conf

写入/etc/nsswitch.conf文件dns内容，表示查询主机名的时候县查询文件再查询dns。主要用于解决无法通过域名访问mysql的问题。

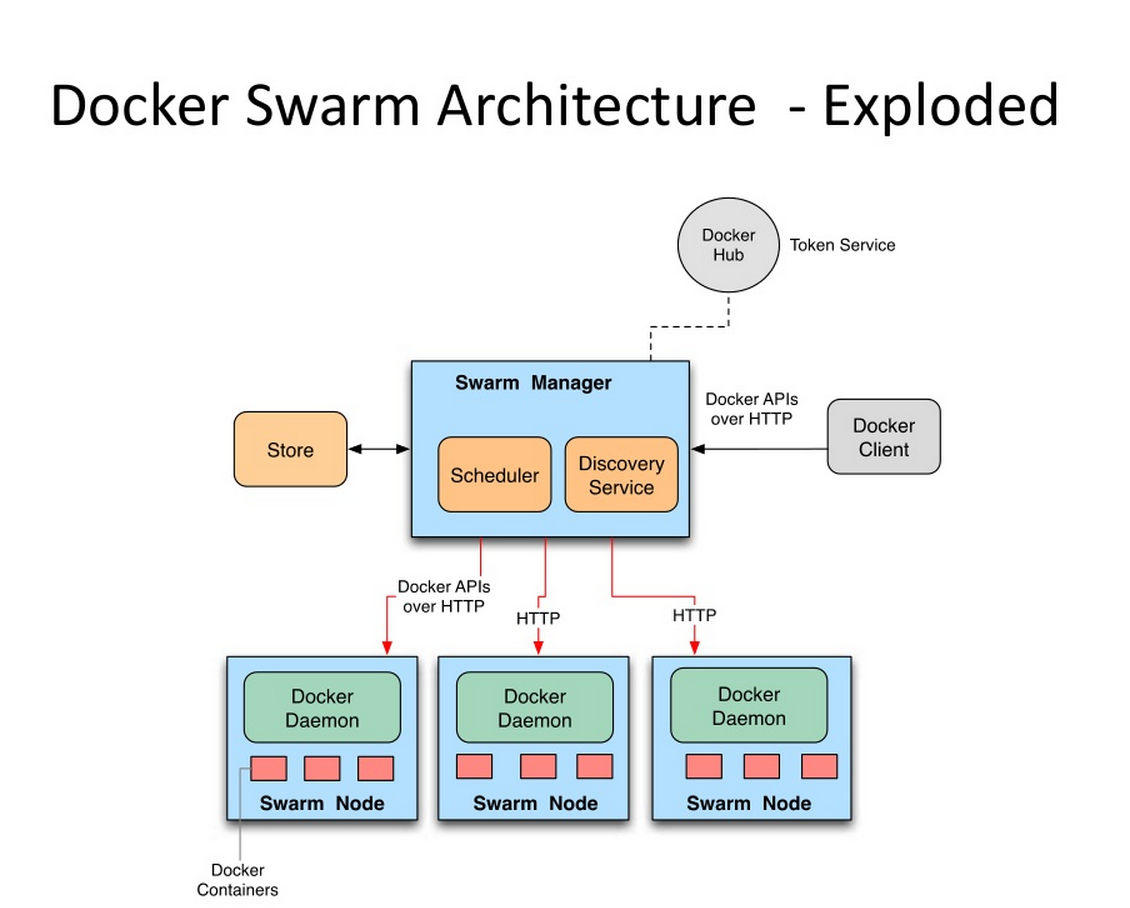
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

指明容器的入口。除了使用ENTRYPOINT 还可以使用CMD的形式来执行入口，两者的区别在于, cmd能被命令行覆盖，而entrypoint不能。举个例子，如果把上面的entrypoint替换成cmd，那么docker container run -it --rm app:1.0 /bin/bash能够替换掉java -jar的执行，容器启动后展示出来的是一个bash。而使用上面的entrypoint格式执行docker container run -it --rm app:1.0 /bin/bash，最终再容器里面执行的是java -Djava.security.egd=file:/dev/./urandom -jar /app.jar /bin/bash。

# Docker compose

*Todo：由于网络原因始终无法下载compose，后续补充*

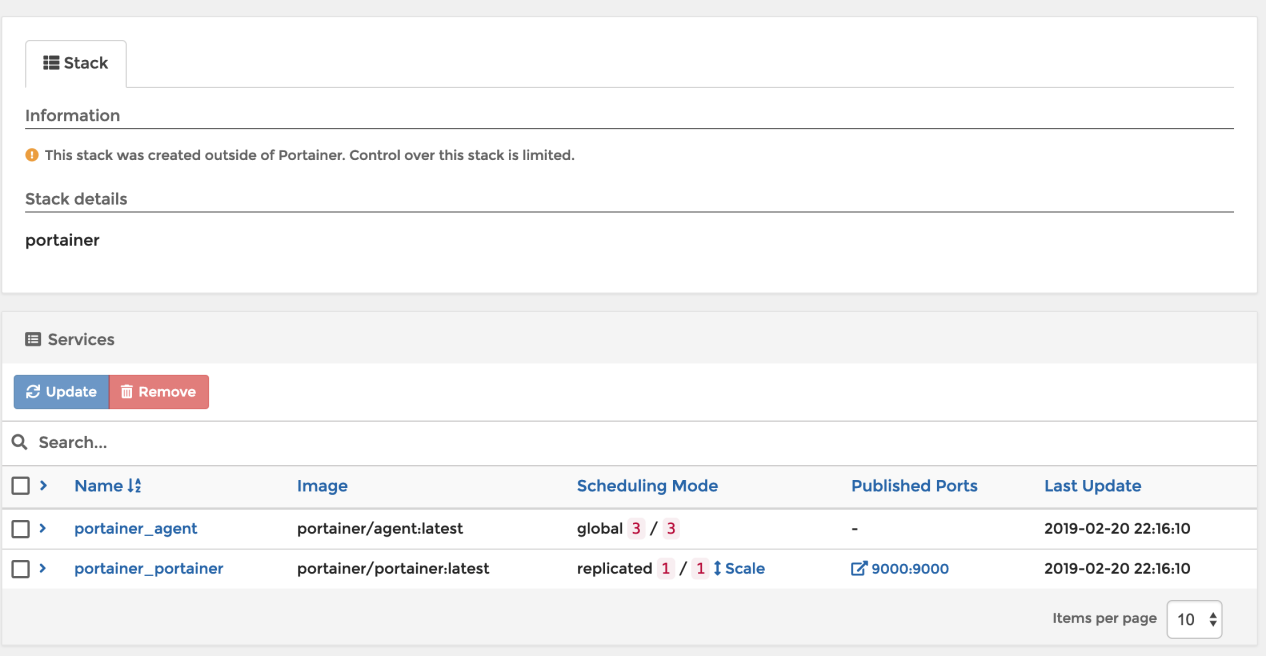
# Docker Swarm



Swarm是Docker官方提供的一款集群管理工具，其主要作用是把若干台Docker主机抽象为一个整体，并且通过一个入口统一管理这些Docker主机上的各种Docker资源。Swarm和Kubernetes比较类似，但是更加轻，具有的功能也较kubernetes更少一些。

## Stack

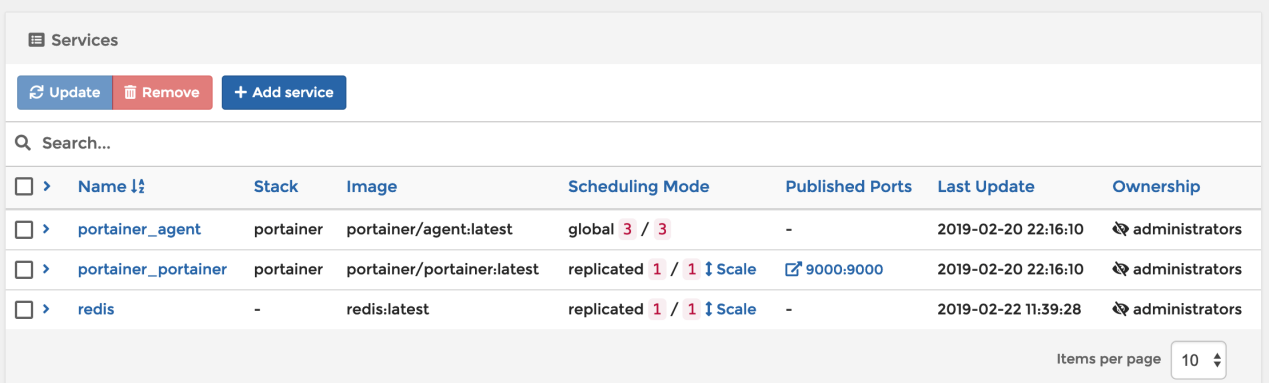
stack 是构成特定环境中的 service 集合, 它是自动部署多个相互关联的服务的简便方法，而无需单独定义每个服务。比如，我们现有一个微服务项目，包含10个服务，每个服务为一个service，所有的service组成了一个stack。



如上图，是portianer这个应用的stack，包含了两个服务，一个是portianer\_agent,类型是global，表示每个node上运行一个实例，用于收集node的数据。另外一个服务是portianer\_portianer，是portianer页面服务，类型是replicated，表示可以在集群里面运行多个实例，当前运行了一个实例。

## Service

service可以理解成集群版的container，一个container只能运行在一个docker节点，而service可以在集群中运行。通过scale执行可以调整一个service在集群中的副本数，例如：我们发现眼里上来后，集群中某个service的副本不够导致运行缓慢，那么可以实时的添加该service的副本数量；当压力过去后再降低service的副本个数。



## Config

Docker config主要用于存放应用的配置文件。我们知道，为了构建的镜像最大灵活度，我们通常会把配置信息放在容器外（配置变更无需重新构建镜像），通过环境变量或者挂载卷的形式透传到容器中。在单机情况下没有问题，但是在集群环境下，一个service可能会在多个节点间漂移。比如某个service运行在node1，此时node1当机，service会在node2启动，这个时候如果配置文件使用的是挂载的形式，那么容器将无法运行。解决方案有两个：

1. 每个节点上存放相同的配置，缺点：配置麻烦，管理麻烦
2. 使用docker config

这个时候docker config出场了，他将配置文件存放在raft log中，保证这个集群的配置的可访问与一致性。

一个简单的使用例子如下：

1、创建配置

echo "This is a config" | docker config create my-config -

2、创建 redis 服务，使用 my-config 配置

docker service create --name redis --config my-config redis

3、查看容器所在的节点

docker service ps redis

4、查看配置内容

docker exec $(docker ps --filter name=redis -q) cat /my-config

## Secret

在 Docker Swarm 服务中, Secret 是一种 BLOB（二进制大对象） 数据, 就像密码、SSH 私钥、 SSL 证书或那些不应该未加密就直接存储在 Dockerfile 或应用程序代码中的数据。在 Docker 1.13 及更高版本中，可以使用 Docker Secrets 集中管理这些数据，并将其安全地传输给需要访问的容器。 一个给定的 Secret 只能被那些已被授予明确访问权限的服务使用，并且只能在这些服务任务正在运行的情况下。

不想在镜像或代码中管理的任何敏感数据，都可以使用 Secret 来管理，比如：

用户名和密码

* TLS certificates and keys
* SSH keys
* 其他重要数据：比如数据库名、内部服务器信息
* 通用的字符串或二进制内容 (最大可达 500 Kb)

举个例子：

1、创建文件 password.txt，里面存入 Mysql 的 root 密码

echo "myPassword" > password.txt

2、创建文件 wordpress.yml，用于启动 mysql 和 wordpress 服务，内容：

version: '3.3'

services:

db:

image: mysql:5.6

environment:

MYSQL\_ROOT\_PASSWORD\_FILE: /run/secrets/db\_password

MYSQL\_DATABASE: wordpress

secrets:

- db\_password

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: root

WORDPRESS\_DB\_PASSWORD\_FILE: /run/secrets/db\_password

secrets:

- db\_password

secrets:

db\_password:

file: password.txt

3、启动 wordpress 服务

docker stack deploy -c wordpress.yml wordpress

1. 删除密钥文件

rm password.txt

# GluseterFS

docker集群适合运行无状态的容器，这些容器在集群中启动，停止，迁移不会导致问题。当容器有状态的时候会遇到新的问题。例如，在集群中运行了一个jenkins应用，容器跑在node1上，此时node1当机，容器迁移到node2，所有数据丢失。

为了将容器无状态，我们引入分布式文件系统glusterfs。通过对比其他方案，例如ceph,nfs后发现，glusterfs具有安装简单，容错等优点，适合用于测试环境。

下面我们通过例子来学习下glusterFs怎么跟docker结合。

## 安装

1. 准备磁盘（我们使用文件作为虚拟磁盘）(三台都要执行)

mkdir /data/disk/ -p

cd /data/disk

dd if=/dev/zero of=gluster.disk bs=1G count=0 seek=30

mkfs.xfs -i size=512 /data/disk/gluster.disk

mkdir -p /data/brick1

echo '/data/disk/gluster.disk /data/brick1 xfs defaults 1 2' >> /etc/fstab

mount -a && mount

1. 安装glusterfs(三台都要执行)

apt-get install glusterfs-server -y

1. 启动服务(三台都要执行)

systemctl enable glusterfs-server.service

systemctl start glusterfs-server.service

1. 配置集群(在第一台执行)

gluster peer probe swarm-manager-02

gluster peer probe swarm-manager-03

1. 配置集群2(在第二台执行，由于使用host来配置，这步不能省略)

gluster peer probe swarm-manager-01

1. 查看集群状态

gluster peer status

1. 创建挂载点（三台均要执行）

mkdir /sharefs/

1. 创建卷并启动卷

gluster volume create share-vol replica 3 swarm-manager-01:/data/brick1/gv1 swarm-manager-02:/data/brick1/gv1 swarm-manager-03:/data/brick1/gv1

gluster volume start share-vol

1. 挂载点，并且配置开机自动挂载（三台均要执行，替换红色部分hostname ）

mkdir /sharefs

mount -t glusterfs swarm-manager-01:/share-vol /sharefs

echo "swarm-manager-01:/share-vol /sharefs glusterfs defaults,\_netdev 0 0" >> /etc/fstab

## 使用

1. 在swarm-manager-01上执行

mkdir /sharefs/mysql/data -p

1. 创建mysql.yml，内容如下：

version: "3"

services:

mysql:

image: "mysql:5.6"

environment:

MYSQL\_ROOT\_PASSWORD: "123456"

restart: always

volumes:

- "mysql-data:/var/lib/mysql"

ports:

- "3306:3306"

volumes:

mysql-data:

driver: local

driver\_opts:

type: "none"

o: "bind"

device: "/sharefs/mysql/data"

1. 执行docker stack deploy -c mysql.yml mysql
2. 测试
   1. 在01上执行

mysql -h swarm-manager-01 -uroot -p

> create database hello;

> use hello;

> create table world(hello varchar(15));

* 1. 在01上执行

systemctl stop docker.service

* 1. 在02上执行

mysql -h swarm-manager-02 -uroot -p

> show databases;

> use hello;

> show tables;

可以看到，数据还在

# Docker集群环境搭建案例

1. 准备三台虚拟机，本实验使用virtualbox创建虚拟机

实验环境：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | Ip | role | hostname |
| docker-master01 | 192.168.56.7 | manager | swarm-manager-01 |
| docker-worker01 | 192.168.56.8 | manager | swarm-manager-02 |
| docker-worker02 | 192.168.56.9 | manager | swarm-manager-03 |

1. 修改host

> 修改/etc/hosts成下面的内容，红色部分改成所在主机的hostname

127.0.0.1       localhost

127.0.1.1       swarm-manager-01

192.168.56.7    swarm-manager-01

192.168.56.8    swarm-manager-02

192.168.56.9    swarm-manager-03

> 修改/etc/hostname，内容改成所在主机的hostname

> 执行命令(红色部分改成所在主机的hostname)

Hostname swarm-manager-01

1. 安装docker

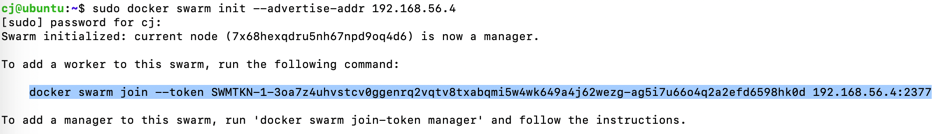
参考上文，安装docker

1. 集群搭建

在manager01上执行

docker swarm init --advertise-addr 192.168.56.7

命令完成后将输出如下信息



在manager02/manager03上执行上述图片中的命令，集群搭建完成

1. 安装可视化工具portainer

$ curl -L https://downloads.portainer.io/portainer-agent-stack.yml -o portainer-agent-stack.yml

$ mkdir /sharefs/portainer

修改portainer-agent-stack.yml,

volumes:

  portainer\_data:

    driver: local

    driver\_opts:

      type: "none"

      o: "bind"

      device: "/sharefs/portainer\_data"

部署

$ docker stack deploy --compose-file=portainer-agent-stack.yml portainer

1. 访问控制台

使用浏览器访问http://192.168.56.7:9000

# 常见问题

1. 怎删除没有用的镜像?
2. 怎么删除停止的容器?
3. 怎么拷贝某个镜像里面的文件到宿主机?
4. 假设有个maven镜像，我们需要修改他的源地址为aliyun，怎么操作
5. 一个应用发布了新版本，怎么更新docker swarm中所有的该应用
6. maven镜像每次编译都回去远端下载，有没有什么好的办法可以解决呢？
7. Error response from daemon: Get https://registry-1.docker.io/v2/rexray/rbd/manifests/latest

dig @114.114.114.114 registry-1.docker.io

使用上述命令将显示域名对应的ip地址，可以在hosts中添加对应的ip映射即可