Docker容器操作命令

小清单

Eleven

2020

目录

[第一章、Docker 安装 4](#_Toc43494531)

[第一步 卸载旧的安装包 4](#_Toc43494532)

[第二步 安装需要的安装包 4](#_Toc43494533)

[第三步 设置镜像的仓库 4](#_Toc43494534)

[第四步# 安装docker相关的 4](#_Toc43494535)

[第五步启动docker 服务 5](#_Toc43494536)

[第六步 hello world 5](#_Toc43494537)

[第二章、Docker 卸载 5](#_Toc43494538)

[# 卸载依赖 5](#_Toc43494539)

[# 删除资源 5](#_Toc43494540)

[第三章、Docker镜像相关指令 5](#_Toc43494541)

[1 docker镜像基本命令： 5](#_Toc43494542)

[查看所有镜像 5](#_Toc43494543)

[docker images 5](#_Toc43494544)

[获取新的镜像： 6](#_Toc43494545)

[查找镜像 6](#_Toc43494546)

[删除镜像(会提示先停止使用中的容器) 6](#_Toc43494547)

[创建镜像 6](#_Toc43494548)

[更新镜像 6](#_Toc43494549)

[设置镜像标签 7](#_Toc43494550)

[第四章、 Docker容器基本命令 7](#_Toc43494551)

[查看全部命令 7](#_Toc43494552)

[容器实例基本操作 7](#_Toc43494553)

[容器导入导出 9](#_Toc43494554)

[第五章、Docker 仓库管理 9](#_Toc43494555)

[1 注册-登录 9](#_Toc43494556)

[2 DockerHub基本操作 9](#_Toc43494557)

[第六章、Dockerfile指令 10](#_Toc43494558)

[FROM：指定基础镜像，必须为第一个命令 10](#_Toc43494559)

[MAINTAINER: 维护者信息 10](#_Toc43494560)

[RUN：构建镜像时执行的命令 10](#_Toc43494561)

[ADD命令 11](#_Toc43494562)

[COPY： 11](#_Toc43494563)

[CMD： 11](#_Toc43494564)

[ENTRYPOINT： 12](#_Toc43494565)

[LABEL： 13](#_Toc43494566)

[ENV： 13](#_Toc43494567)

[EXPOSE： 13](#_Toc43494568)

[VOLUME： 14](#_Toc43494569)

[WORKDIR： 14](#_Toc43494570)

[USER: 15](#_Toc43494571)

[ARG： 15](#_Toc43494572)

[ONBUILD： 16](#_Toc43494573)

[第七章、Docker Compose 16](#_Toc43494574)

[Compose 简介 16](#_Toc43494575)

[Compose 安装 16](#_Toc43494576)

[yml 配置指令参考 17](#_Toc43494577)

[version 17](#_Toc43494578)

[build 17](#_Toc43494579)

[cap\_add，cap\_drop 18](#_Toc43494580)

[cgroup\_parent 18](#_Toc43494581)

[command 18](#_Toc43494582)

[container\_name 18](#_Toc43494583)

[depends\_on 19](#_Toc43494584)

[deploy 19](#_Toc43494585)

[devices 22](#_Toc43494586)

[dns 22](#_Toc43494587)

[dns\_search 22](#_Toc43494588)

[entrypoint 23](#_Toc43494589)

[env\_file 23](#_Toc43494590)

[environment 23](#_Toc43494591)

[expose 24](#_Toc43494592)

[extra\_hosts 24](#_Toc43494593)

[healthcheck 24](#_Toc43494594)

[image 24](#_Toc43494595)

[logging 25](#_Toc43494596)

[network\_mode 25](#_Toc43494597)

[restart 26](#_Toc43494598)

[secrets 26](#_Toc43494599)

[security\_opt 27](#_Toc43494600)

[stop\_grace\_period 27](#_Toc43494601)

[stop\_signal 27](#_Toc43494602)

[sysctls 27](#_Toc43494603)

[tmpfs 28](#_Toc43494604)

[ulimits 28](#_Toc43494605)

[volumes 28](#_Toc43494606)

[Swarm 集群管理 29](#_Toc43494607)

[简介 29](#_Toc43494608)

[原理 29](#_Toc43494609)

[使用 30](#_Toc43494610)

[1、创建 swarm 集群管理节点（manager） 30](#_Toc43494611)

[2、创建 swarm 集群工作节点（worker） 31](#_Toc43494612)

[3、查看集群信息 32](#_Toc43494613)

[4、部署服务到集群中 33](#_Toc43494614)

[5、查看服务部署情况 33](#_Toc43494615)

[6、扩展集群服务 34](#_Toc43494616)

[7、删除服务 34](#_Toc43494617)

[8、滚动升级服务 34](#_Toc43494618)

[9、停止某个节点接收新的任务 35](#_Toc43494619)

# 第一章、Docker 安装

#在root 用户下

## 第一步 卸载旧的安装包

yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

## 第二步 安装需要的安装包

yum install -y yum-utils

## 第三步 设置镜像的仓库

yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo --默认是国外的

#如果没有vpn 建议安装阿里云的

yum-config-manager \

--add-repo \

http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

#更新yum 索引安装包

yum makecache fast

## 第四步# 安装docker相关的

yum install docker-ce docker-ce-cli containerd.io (可能会出错，多试几次)

## 第五步启动docker 服务

systemctl start docker

#查看docker 是否安装完成

docker --version

## 第六步 hello world

docker run hello-world

#查看所有的docker 镜像

docker images

docker search aspnet

docker pull mcr.microsoft.com/dotnet/core/aspnet:latest

# 第二章、Docker 卸载

## # 卸载依赖

yum remove docker-ce docker-ce-cli containerd.io

## # 删除资源

rm -rf /var/lib/docker

# 第三章、Docker镜像相关指令

## 1 docker镜像基本命令：

### 查看所有镜像

### docker images

* **REPOSITORY：**表示镜像的仓库源
* **TAG：**镜像的标签
* **IMAGE ID：**镜像ID
* **CREATED：**镜像创建时间
* **SIZE：**镜像大小

### 获取新的镜像：

docker pull 名称

### 查找镜像

docker search httpd

**NAME:**镜像仓库源的名称

**DESCRIPTION:**镜像的描述

**OFFICIAL:**是否 docker 官方发布

**stars:**类似 Github 里面的 star，表示点赞、喜欢的意思。

**AUTOMATED:**自动构建。

### 删除镜像(会提示先停止使用中的容器)

docker rmi  镜像name/镜像id

### 创建镜像

使用 Dockerfile 指令来创建一个新的镜像

**docker build** ， 从零开始来创建一个新的镜像。为此，我们需要创建一个 Dockerfile 文件，其中包含一组指令来告诉 Docker 如何构建我们的镜像。

### 更新镜像

1 运行的容器

2 docker exec -it fd2c868cadlc /bin/bash 进入容器操作

3 **apt-get update**

**4** exit

5 docker commit 来提交容器副本

docker commit -m="nginx has update0618" -a="eleven" 79323dxds323 nginx-8081-8082:vCustom2

* **-m:** 提交的描述信息
* **-a:** 指定镜像作者
* 79323dxds323**：**容器 ID
* nginx-8081-8082:vCustom2**:** 指定要创建的目标镜像名-tag

(要点时间)

### 设置镜像标签

docker tag 命令，为镜像添加一个新的标签。

docker tag 860c279d2fec runoob/centos:dev

docker tag 镜像ID，这里是 860c279d2fec ,用户名称、镜像源名(repository name)和新的标签名(tag)。

# 第四章、 Docker容器基本命令

## 查看全部命令

docker

具体命令详情 docker ps –help

启动容器 docker run -it nginx /bin/bash

* **-i**: 交互式操作。
* **-t**: 终端。
* **-d 后台运行**
* nginx: nginx镜像。
* **/bin/bash**：放在镜像名后的是命令，这里我们希望有个交互式 Shell，因此用的是 /bin/bash。(exit 退出终端)

## 容器实例基本操作

查看所有容器 docker ps -a

查看容器运行日志 docker logs 容器名称/容器id

停止容器运行 docker stop 容器name/容器id

终止容器后运行 docker start 容器name/容器id

容器重启 docker restart 容器name/容器id

删除容器 docker rm -f 容器name/容器id

#删除镜像

docker rmi -f 镜像id (可以根据 docker images 查询)

docker rmi -f $(docker images) --删除所有镜像

#查询docker 的详细信息

docker stats dockerid

查看 Docker 的底层信息

**docker inspect** 来查看 Docker 的底层信息

二、停止一个正在运行的容器

1、docker stop 此方式常常被翻译为优雅的停止容器

docker stop 容器ID或容器名

参数 -t：关闭容器的限时，如果超时未能关闭则用kill强制关闭，默认值10s，这个时间用于容器的自己保存状态

docker stop -t=60 容器ID或容器名

2、docker kill

docker kill 容器ID或容器名 :直接关闭容器

由此可见stop和kill的主要区别:stop给与一定的关闭时间交由容器自己保存状态，kill直接关闭容器

想更进一步了解处理机制的可以看下面这篇文章，比较详细但是需要其他方面的

1.停用全部运行中的容器:

docker stop $(docker ps -q)

2.删除全部容器：

docker rm $(docker ps -aq)

3.一条命令实现停用并删除容器：

docker stop $(docker ps -q) & docker rm $(docker ps -aq)

## 容器导入导出

**导出容器**

如果要导出本地某个容器，可以使用 **docker export** 命令。

docker export 1e560fca3906 > nginx.tar

**导入容器快照**

可以使用 docker import 从容器快照文件中再导入为镜像

$ cat docker/ubuntu.tar | docker import - test/ubuntu:v1

也可以通过指定 URL 或者某个目录来导入

$ docker import http://example.com/exampleimage.tgz example/imagerepo

# 第五章、Docker 仓库管理

## 1 注册-登录

[https://hub.docker.com](https://hub.docker.com/)

eleven202001

eleven202001

xuyang@ZhaoxiEdu.Net

## 2 DockerHub基本操作

docker login

登录

docker logout

退出

搜索镜像：

docker search nginx

拉取镜像

docker pull nginx

推送镜像

docker push nginx-8081-8082:vCustom2（权限没成功）

# 第六章、Dockerfile指令

## FROM：指定基础镜像，必须为第一个命令

[复制代码](javascript:void(0);)

格式：

　　FROM <image>

　　FROM <image>:<tag>

　　FROM <image>@<digest>  
示例：  
　　FROM mysql:5.6  
注：  
　　tag或digest是可选的，如果不使用这两个值时，会使用latest版本的基础镜像

[复制代码](javascript:void(0);)

## MAINTAINER: 维护者信息

格式：

MAINTAINER <name>

示例：

MAINTAINER Jasper Xu

MAINTAINER sorex@163.com

MAINTAINER Jasper Xu <sorex@163.com>

## RUN：构建镜像时执行的命令

[复制代码](javascript:void(0);)

RUN用于在镜像容器中执行命令，其有以下两种命令执行方式：

**shell执行**

格式：

RUN <command>

**exec执行**

格式：

RUN ["executable", "param1", "param2"]

示例：

RUN ["executable", "param1", "param2"]

RUN apk update

RUN ["/etc/execfile", "arg1", "arg1"]

注：  
　　RUN指令创建的中间镜像会被缓存，并会在下次构建中使用。如果不想使用这些缓存镜像，可以在构建时指定--no-cache参数，如：docker build --no-cache

[复制代码](javascript:void(0);)

## ADD命令

**：将本地文件添加到容器中，tar类型文件会自动解压(网络压缩资源不会被解压)，可以访问网络资源，类似wget**

[复制代码](javascript:void(0);)

格式：

ADD <src>... <dest>

ADD ["<src>",... "<dest>"] 用于支持包含空格的路径

示例：

ADD hom\* /mydir/ # 添加所有以"hom"开头的文件

ADD hom?.txt /mydir/ # ? 替代一个单字符,例如："home.txt"

ADD test relativeDir/ # 添加 "test" 到 `WORKDIR`/relativeDir/

ADD test /absoluteDir/ # 添加 "test" 到 /absoluteDir/

[复制代码](javascript:void(0);)

## COPY：

**功能类似ADD，但是是不会自动解压文件，也不能访问网络资源**

## CMD：

**构建容器后调用，也就是在容器启动时才进行调用。**

[复制代码](javascript:void(0);)

格式：

CMD ["executable","param1","param2"] (执行可执行文件，优先)

CMD ["param1","param2"] (设置了ENTRYPOINT，则直接调用ENTRYPOINT添加参数)

CMD command param1 param2 (执行shell内部命令)

示例：

CMD echo "This is a test." | wc -

CMD ["/usr/bin/wc","--help"]  
注：  
 　　CMD不同于RUN，CMD用于指定在容器启动时所要执行的命令，而RUN用于指定镜像构建时所要执行的命令。

[复制代码](javascript:void(0);)

## ENTRYPOINT：

**配置容器，使其可执行化。配合CMD可省去"application"，只使用参数。**

[复制代码](javascript:void(0);)

格式：

ENTRYPOINT ["executable", "param1", "param2"] (可执行文件, 优先)

ENTRYPOINT command param1 param2 (shell内部命令)

示例：

FROM ubuntu

ENTRYPOINT ["top", "-b"]

CMD ["-c"]  
注：  
　　　ENTRYPOINT与CMD非常类似，不同的是通过docker run执行的命令不会覆盖ENTRYPOINT，而docker run命令中指定的任何参数，都会被当做参数再次传递给ENTRYPOINT。Dockerfile中只允许有一个ENTRYPOINT命令，多指定时会覆盖前面的设置，而只执行最后的ENTRYPOINT指令。

[复制代码](javascript:void(0);)

## LABEL：

**用于为镜像添加元数据**

格式：

LABEL <key>=<value> <key>=<value> <key>=<value> ...

示例：

　　LABEL version="1.0" description="这是一个Web服务器" by="IT笔录"

注：

　　使用LABEL指定元数据时，一条LABEL指定可以指定一或多条元数据，指定多条元数据时不同元数据之间通过空格分隔。推荐将所有的元数据通过一条LABEL指令指定，以免生成过多的中间镜像。

## ENV：

**设置环境变量**

[复制代码](javascript:void(0);)

格式：

ENV <key> <value> #<key>之后的所有内容均会被视为其<value>的组成部分，因此，一次只能设置一个变量

ENV <key>=<value> ... #可以设置多个变量，每个变量为一个"<key>=<value>"的键值对，如果<key>中包含空格，可以使用\来进行转义，也可以通过""来进行标示；另外，反斜线也可以用于续行

示例：

ENV myName John Doe

ENV myDog Rex The Dog

ENV myCat=fluffy

[复制代码](javascript:void(0);)

## EXPOSE：

**指定于外界交互的端口**

[复制代码](javascript:void(0);)

格式：

EXPOSE <port> [<port>...]

示例：

EXPOSE 80 443

EXPOSE 8080  
 EXPOSE 11211/tcp 11211/udp  
注：  
　　EXPOSE并不会让容器的端口访问到主机。要使其可访问，需要在docker run运行容器时通过-p来发布这些端口，或通过-P参数来发布EXPOSE导出的所有端口

[复制代码](javascript:void(0);)

## VOLUME：

**用于指定持久化目录**

[复制代码](javascript:void(0);)

格式：

VOLUME ["/path/to/dir"]

示例：

VOLUME ["/data"]

VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"  
注：  
　　一个卷可以存在于一个或多个容器的指定目录，该目录可以绕过联合文件系统，并具有以下功能：

1 卷可以容器间共享和重用

2 容器并不一定要和其它容器共享卷

3 修改卷后会立即生效

4 对卷的修改不会对镜像产生影响

5 卷会一直存在，直到没有任何容器在使用它

[复制代码](javascript:void(0);)

## WORKDIR：

**工作目录，类似于cd命令**

[复制代码](javascript:void(0);)

格式：

WORKDIR /path/to/workdir

示例：

WORKDIR /a (这时工作目录为/a)

WORKDIR b (这时工作目录为/a/b)

WORKDIR c (这时工作目录为/a/b/c)  
注：  
　　通过WORKDIR设置工作目录后，Dockerfile中其后的命令RUN、CMD、ENTRYPOINT、ADD、COPY等命令都会在该目录下执行。在使用docker run运行容器时，可以通过-w参数覆盖构建时所设置的工作目录。

[复制代码](javascript:void(0);)

## USER:

**指定运行容器时的用户名或 UID，后续的 RUN 也会使用指定用户。使用USER指定用户时，可以使用用户名、UID或GID，或是两者的组合。当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户**

[复制代码](javascript:void(0);)

 格式:  
　　USER user  
　　USER user:group  
　　USER uid  
　　USER uid:gid  
　　USER user:gid  
　　USER uid:group

 示例：  
　　USER www

 注：

　　使用USER指定用户后，Dockerfile中其后的命令RUN、CMD、ENTRYPOINT都将使用该用户。镜像构建完成后，通过docker run运行容器时，可以通过-u参数来覆盖所指定的用户。

[复制代码](javascript:void(0);)

## ARG：

**用于指定传递给构建运行时的变量**

格式：

ARG <name>[=<default value>]

示例：

ARG site

ARG build\_user=www

## ONBUILD：

**用于设置镜像触发器**

[复制代码](javascript:void(0);)

格式：  
　　ONBUILD [INSTRUCTION]

示例：

　　ONBUILD ADD . /app/src

　　ONBUILD RUN /usr/local/bin/python-build --dir /app/src

注：  
　　当所构建的镜像被用做其它镜像的基础镜像，该镜像中的触发器将会被钥触发

[复制代码](javascript:void(0);)

# 第七章、Docker Compose

## Compose 简介

Compose 是用于定义和运行多容器 Docker 应用程序的工具。通过 Compose，您可以使用 YML 文件来配置应用程序需要的所有服务。然后，使用一个命令，就可以从 YML 文件配置中创建并启动所有服务。

YAML教程 <https://www.runoob.com/w3cnote/yaml-intro.html>

Compose 使用的三个步骤：

* 使用 Dockerfile 定义应用程序的环境。
* 使用 docker-compose.yml 定义构成应用程序的服务，这样它们可以在隔离环境中一起运行。
* 最后，执行 docker-compose up 命令来启动并运行整个应用程序。

## Compose 安装

下载 Docker Compose 的当前稳定版本：

$ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

要安装其他版本的 Compose，请替换 1.24.1。

将可执行权限应用于二进制文件：

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

创建软链：

$ sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

测试是否安装成功：

$ docker-compose --version

cker-compose version 1.24.1, build 4667896b

执行以下命令来启动应用程序：

docker-compose up -d

## yml 配置指令参考

### version

指定本 yml 依从的 compose 哪个版本制定的。

### build

指定为构建镜像上下文路径：

例如 webapp 服务，指定为从上下文路径 ./dir/Dockerfile 所构建的镜像：

version: "3.7"

services:

webapp:

build: ./dir

或者，作为具有在上下文指定的路径的对象，以及可选的 Dockerfile 和 args：

version: "3.7"

services:

webapp:

build:

context: ./dir

dockerfile: Dockerfile-alternate

args:

buildno: 1

labels:

- "com.example.description=Accounting webapp"

- "com.example.department=Finance"

- "com.example.label-with-empty-value"

target: prod

* context：上下文路径。
* dockerfile：指定构建镜像的 Dockerfile 文件名。
* args：添加构建参数，这是只能在构建过程中访问的环境变量。
* labels：设置构建镜像的标签。
* target：多层构建，可以指定构建哪一层。

### cap\_add，cap\_drop

添加或删除容器拥有的宿主机的内核功能。

cap\_add:

- ALL # 开启全部权限

cap\_drop:

- SYS\_PTRACE # 关闭 ptrace权限

### cgroup\_parent

为容器指定父 cgroup 组，意味着将继承该组的资源限制。

cgroup\_parent: m-executor-abcd

### command

覆盖容器启动的默认命令。

command: ["bundle", "exec", "thin", "-p", "3000"]

### container\_name

指定自定义容器名称，而不是生成的默认名称。

container\_name: my-web-container

### depends\_on

设置依赖关系。

* docker-compose up ：以依赖性顺序启动服务。在以下示例中，先启动 db 和 redis ，才会启动 web。
* docker-compose up SERVICE ：自动包含 SERVICE 的依赖项。在以下示例中，docker-compose up web 还将创建并启动 db 和 redis。
* docker-compose stop ：按依赖关系顺序停止服务。在以下示例中，web 在 db 和 redis 之前停止。

version: "3.7"

services:

web:

build: .

depends\_on:

- db

- redis

redis:

image: redis

db:

image: postgres

注意：web 服务不会等待 redis db 完全启动 之后才启动。

### deploy

指定与服务的部署和运行有关的配置。只在 swarm 模式下才会有用。

version: "3.7"

services:

redis:

image: redis:alpine

deploy:

mode：replicated

replicas: 6

endpoint\_mode: dnsrr

labels:

description: "This redis service label"

resources:

limits:

cpus: '0.50'

memory: 50M

reservations:

cpus: '0.25'

memory: 20M

restart\_policy:

condition: on-failure

delay: 5s

max\_attempts: 3

window: 120s

可以选参数：

**endpoint\_mode**：访问集群服务的方式。

endpoint\_mode: vip

# Docker 集群服务一个对外的虚拟 ip。所有的请求都会通过这个虚拟 ip 到达集群服务内部的机器。

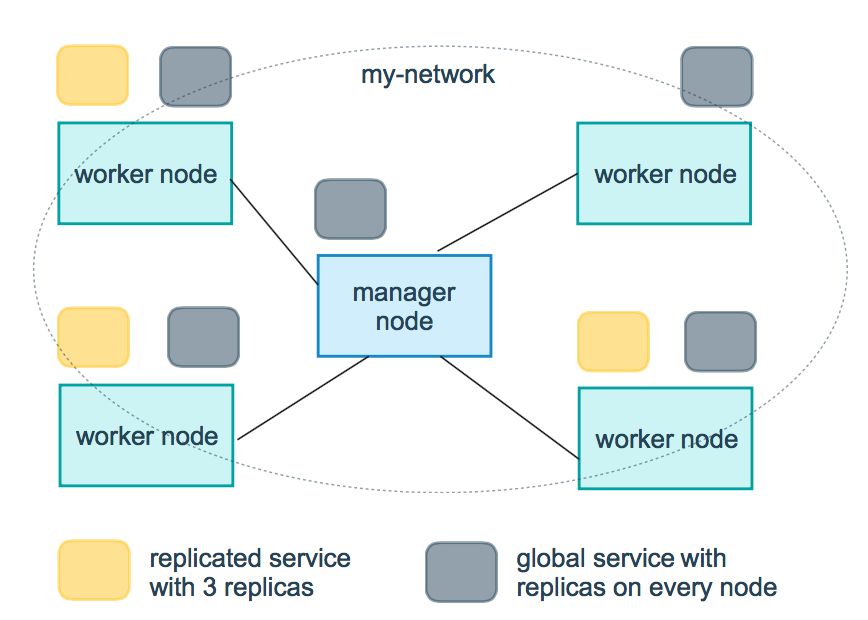
endpoint\_mode: dnsrr

# DNS 轮询（DNSRR）。所有的请求会自动轮询获取到集群 ip 列表中的一个 ip 地址。

**labels**：在服务上设置标签。可以用容器上的 labels（跟 deploy 同级的配置） 覆盖 deploy 下的 labels。

**mode**：指定服务提供的模式。

* **replicated**：复制服务，复制指定服务到集群的机器上。
* **global**：全局服务，服务将部署至集群的每个节点。
* 图解：下图中黄色的方块是 replicated 模式的运行情况，灰色方块是 global 模式的运行情况。



**replicas：mode** 为 replicated 时，需要使用此参数配置具体运行的节点数量。

**resources**：配置服务器资源使用的限制，例如上例子，配置 redis 集群运行需要的 cpu 的百分比 和 内存的占用。避免占用资源过高出现异常。

**restart\_policy**：配置如何在退出容器时重新启动容器。

* condition：可选 none，on-failure 或者 any（默认值：any）。
* delay：设置多久之后重启（默认值：0）。
* max\_attempts：尝试重新启动容器的次数，超出次数，则不再尝试（默认值：一直重试）。
* window：设置容器重启超时时间（默认值：0）。

**rollback\_config**：配置在更新失败的情况下应如何回滚服务。

* parallelism：一次要回滚的容器数。如果设置为0，则所有容器将同时回滚。
* delay：每个容器组回滚之间等待的时间（默认为0s）。
* failure\_action：如果回滚失败，该怎么办。其中一个 continue 或者 pause（默认pause）。
* monitor：每个容器更新后，持续观察是否失败了的时间 (ns|us|ms|s|m|h)（默认为0s）。
* max\_failure\_ratio：在回滚期间可以容忍的故障率（默认为0）。
* order：回滚期间的操作顺序。其中一个 stop-first（串行回滚），或者 start-first（并行回滚）（默认 stop-first ）。

**update\_config**：配置应如何更新服务，对于配置滚动更新很有用。

* parallelism：一次更新的容器数。
* delay：在更新一组容器之间等待的时间。
* failure\_action：如果更新失败，该怎么办。其中一个 continue，rollback 或者pause （默认：pause）。
* monitor：每个容器更新后，持续观察是否失败了的时间 (ns|us|ms|s|m|h)（默认为0s）。
* max\_failure\_ratio：在更新过程中可以容忍的故障率。
* order：回滚期间的操作顺序。其中一个 stop-first（串行回滚），或者 start-first（并行回滚）（默认stop-first）。

**注**：仅支持 V3.4 及更高版本。

### devices

指定设备映射列表。

devices:

- "/dev/ttyUSB0:/dev/ttyUSB0"

### dns

自定义 DNS 服务器，可以是单个值或列表的多个值。

dns: 8.8.8.8

dns:

- 8.8.8.8

- 9.9.9.9

### dns\_search

自定义 DNS 搜索域。可以是单个值或列表。

dns\_search: example.com

dns\_search:

- dc1.example.com

- dc2.example.com

### entrypoint

覆盖容器默认的 entrypoint。

entrypoint: /code/entrypoint.sh

也可以是以下格式：

entrypoint:

- php

- -d

- zend\_extension=/usr/local/lib/php/extensions/no-debug-non-zts-20100525/xdebug.so

- -d

- memory\_limit=-1

- vendor/bin/phpunit

### env\_file

从文件添加环境变量。可以是单个值或列表的多个值。

env\_file: .env

也可以是列表格式：

env\_file:

- ./common.env

- ./apps/web.env

- /opt/secrets.env

### environment

添加环境变量。您可以使用数组或字典、任何布尔值，布尔值需要用引号引起来，以确保 YML 解析器不会将其转换为 True 或 False。

environment:

RACK\_ENV: development

SHOW: 'true'

### expose

暴露端口，但不映射到宿主机，只被连接的服务访问。

仅可以指定内部端口为参数：

expose:

- "3000"

- "8000"

### extra\_hosts

添加主机名映射。类似 docker client --add-host。

extra\_hosts:

- "somehost:162.242.195.82"

- "otherhost:50.31.209.229"

以上会在此服务的内部容器中 /etc/hosts 创建一个具有 ip 地址和主机名的映射关系：

162.242.195.82 somehost

50.31.209.229 otherhost

### healthcheck

用于检测 docker 服务是否健康运行。

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost"] # 设置检测程序

interval: 1m30s # 设置检测间隔

timeout: 10s # 设置检测超时时间

retries: 3 # 设置重试次数

start\_period: 40s # 启动后，多少秒开始启动检测程序

### image

指定容器运行的镜像。以下格式都可以：

image: redis

image: ubuntu:14.04

image: tutum/influxdb

image: example-registry.com:4000/postgresql

image: a4bc65fd # 镜像id

### logging

服务的日志记录配置。

driver：指定服务容器的日志记录驱动程序，默认值为json-file。有以下三个选项

driver: "json-file"

driver: "syslog"

driver: "none"

仅在 json-file 驱动程序下，可以使用以下参数，限制日志得数量和大小。

logging:

driver: json-file

options:

max-size: "200k" # 单个文件大小为200k

max-file: "10" # 最多10个文件

当达到文件限制上限，会自动删除旧得文件。

syslog 驱动程序下，可以使用 syslog-address 指定日志接收地址。

logging:

driver: syslog

options:

syslog-address: "tcp://192.168.0.42:123"

### network\_mode

设置网络模式。

network\_mode: "bridge"

network\_mode: "host"

network\_mode: "none"

network\_mode: "service:[service name]"

network\_mode: "container:[container name/id]"

networks

配置容器连接的网络，引用顶级 networks 下的条目 。

services:

some-service:

networks:

some-network:

aliases:

- alias1

other-network:

aliases:

- alias2

networks:

some-network:

# Use a custom driver

driver: custom-driver-1

other-network:

# Use a custom driver which takes special options

driver: custom-driver-2

**aliases** ：同一网络上的其他容器可以使用服务名称或此别名来连接到对应容器的服务。

### restart

* no：是默认的重启策略，在任何情况下都不会重启容器。
* always：容器总是重新启动。
* on-failure：在容器非正常退出时（退出状态非0），才会重启容器。
* unless-stopped：在容器退出时总是重启容器，但是不考虑在Docker守护进程启动时就已经停止了的容器

restart: "no"

restart: always

restart: on-failure

restart: unless-stopped

注：swarm 集群模式，请改用 restart\_policy。

### secrets

存储敏感数据，例如密码：

version: "3.1"

services:

mysql:

image: mysql

environment:

MYSQL\_ROOT\_PASSWORD\_FILE: /run/secrets/my\_secret

secrets:

- my\_secret

secrets:

my\_secret:

file: ./my\_secret.txt

### security\_opt

修改容器默认的 schema 标签。

security-opt：

- label:user:USER # 设置容器的用户标签

- label:role:ROLE # 设置容器的角色标签

- label:type:TYPE # 设置容器的安全策略标签

- label:level:LEVEL # 设置容器的安全等级标签

### stop\_grace\_period

指定在容器无法处理 SIGTERM (或者任何 stop\_signal 的信号)，等待多久后发送 SIGKILL 信号关闭容器。

stop\_grace\_period: 1s # 等待 1 秒

stop\_grace\_period: 1m30s # 等待 1 分 30 秒

默认的等待时间是 10 秒。

### stop\_signal

设置停止容器的替代信号。默认情况下使用 SIGTERM 。

以下示例，使用 SIGUSR1 替代信号 SIGTERM 来停止容器。

stop\_signal: SIGUSR1

### sysctls

设置容器中的内核参数，可以使用数组或字典格式。

sysctls:

net.core.somaxconn: 1024

net.ipv4.tcp\_syncookies: 0

sysctls:

- net.core.somaxconn=1024

- net.ipv4.tcp\_syncookies=0

### tmpfs

在容器内安装一个临时文件系统。可以是单个值或列表的多个值。

tmpfs: /run

tmpfs:

- /run

- /tmp

### ulimits

覆盖容器默认的 ulimit。

ulimits:

nproc: 65535

nofile:

soft: 20000

hard: 40000

### volumes

将主机的数据卷或着文件挂载到容器里。

version: "3.7"

services:

db:

image: postgres:latest

volumes:

- "/localhost/postgres.sock:/var/run/postgres/postgres.sock"

- "/localhost/data:/var/lib/postgresql/data"

# Swarm 集群管理

## 简介

Docker Swarm 是 Docker 的集群管理工具。它将 Docker 主机池转变为单个虚拟 Docker 主机。 Docker Swarm 提供了标准的 Docker API，所有任何已经与 Docker 守护程序通信的工具都可以使用 Swarm 轻松地扩展到多个主机。

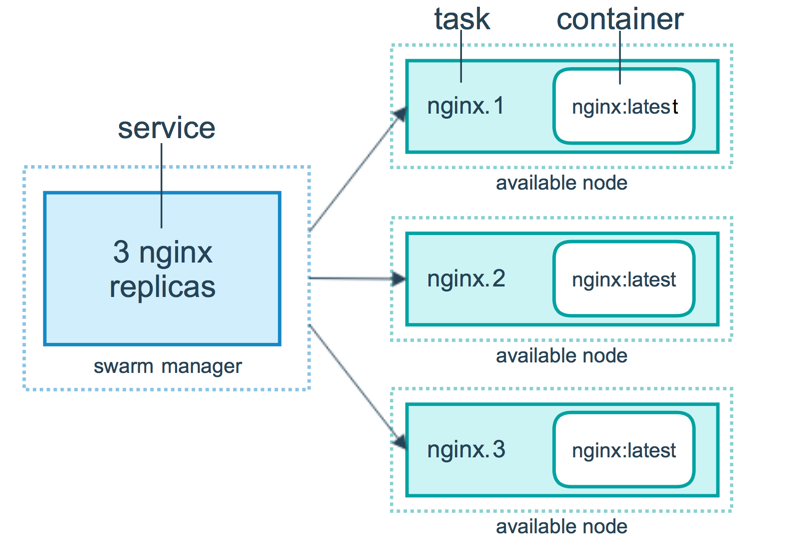
支持的工具包括但不限于以下各项：

* Dokku
* Docker Compose
* Docker Machine
* Jenkins

## 原理

如下图所示，swarm 集群由管理节点（manager）和工作节点（work node）构成。

* **swarm mananger**：负责整个集群的管理工作包括集群配置、服务管理等所有跟集群有关的工作。
* **work node**：即图中的 available node，主要负责运行相应的服务来执行任务（task）。

[](https://www.runoob.com/wp-content/uploads/2019/11/services-diagram.png)

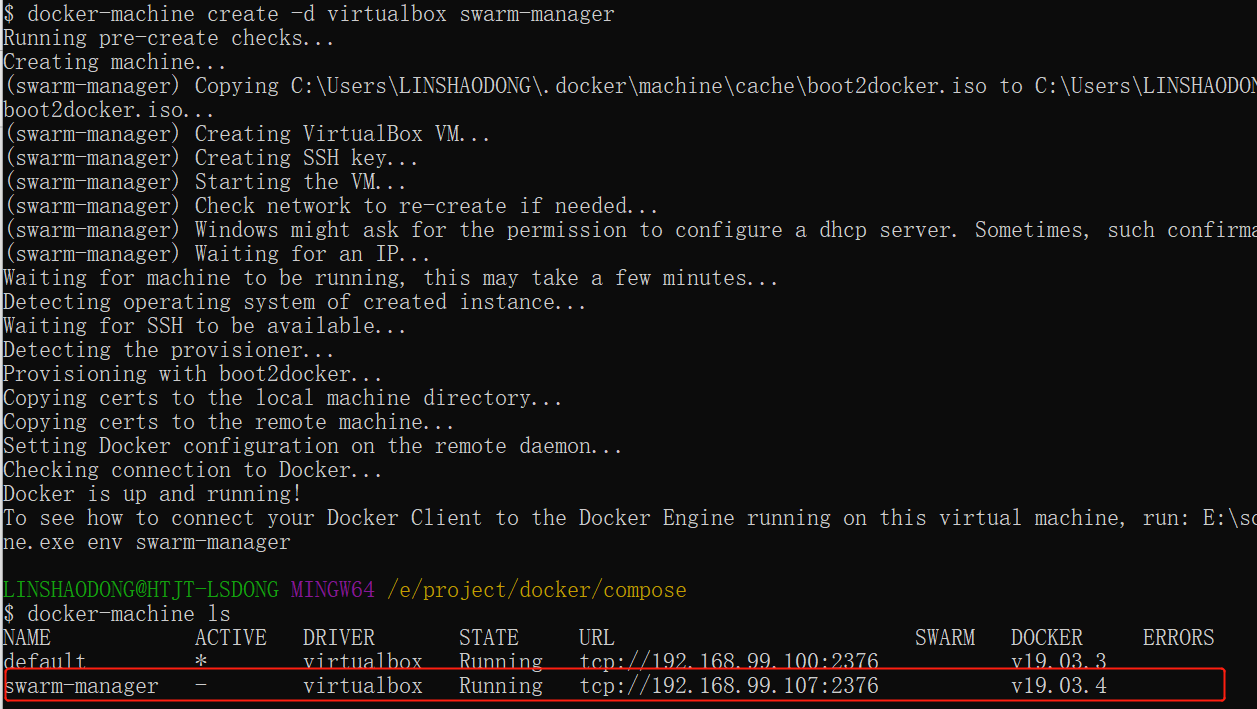
## 使用

以下示例，均以 Docker Machine 和 virtualbox 进行介绍，确保你的主机已安装 virtualbox。

### 1、创建 swarm 集群管理节点（manager）

创建 docker 机器：

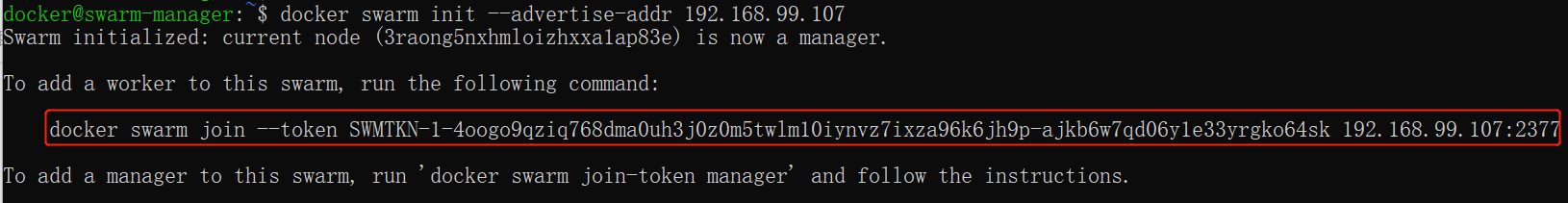
$ docker-machine create -d virtualbox swarm-manager

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm1.png)

初始化 swarm 集群，进行初始化的这台机器，就是集群的管理节点。

$ docker-machine ssh swarm-manager

$ docker swarm init --advertise-addr 192.168.99.107 #这里的 IP 为创建机器时分配的 ip。

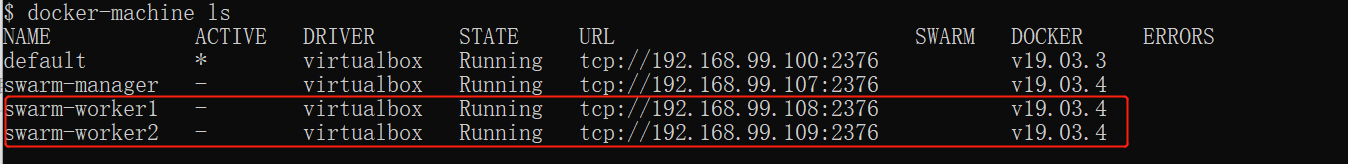
[](https://www.runoob.com/wp-content/uploads/2019/11/swarm2.png)

以上输出，证明已经初始化成功。需要把以下这行复制出来，在增加工作节点时会用到：

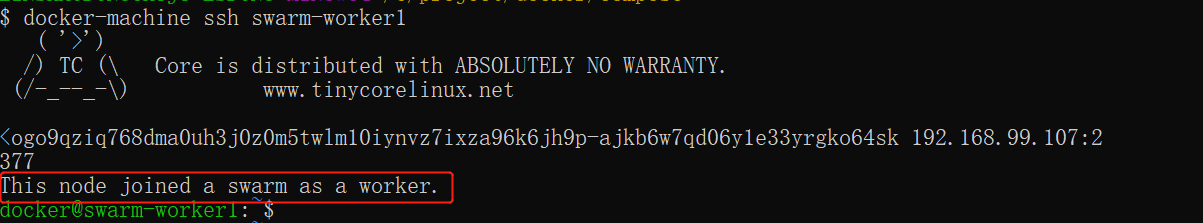
docker swarm join --token SWMTKN-1-4oogo9qziq768dma0uh3j0z0m5twlm10iynvz7ixza96k6jh9p-ajkb6w7qd06y1e33yrgko64sk 192.168.99.107:2377

### 2、创建 swarm 集群工作节点（worker）

这里直接创建好俩台机器，swarm-worker1 和 swarm-worker2 。

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm3.png)

分别进入两个机器里，指定添加至上一步中创建的集群，这里会用到上一步复制的内容。

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm4.png)

以上数据输出说明已经添加成功。

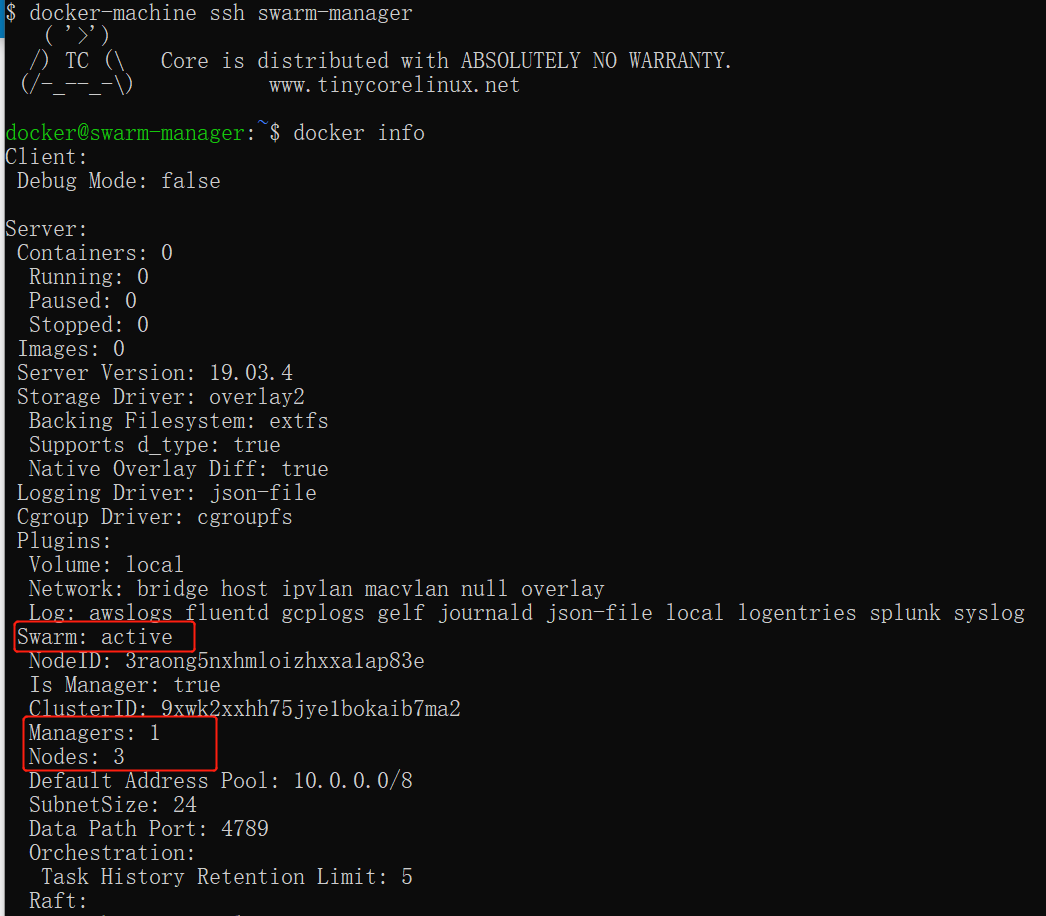
上图中，由于上一步复制的内容比较长，会被自动截断，实际上在图运行的命令如下：

docker@swarm-worker1:~$ docker swarm join --token SWMTKN-1-4oogo9qziq768dma0uh3j0z0m5twlm10iynvz7ixza96k6jh9p-ajkb6w7qd06y1e33yrgko64sk 192.168.99.107:2377

### 3、查看集群信息

进入管理节点，执行：docker info 可以查看当前集群的信息。

$ docker info

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm5.png)

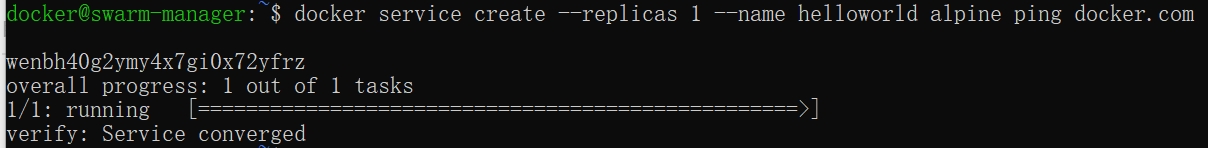
通过画红圈的地方，可以知道当前运行的集群中，有三个节点，其中有一个是管理节点。

### 4、部署服务到集群中

**注意**：跟集群管理有关的任何操作，都是在管理节点上操作的。

以下例子，在一个工作节点上创建一个名为 helloworld 的服务，这里是随机指派给一个工作节点：

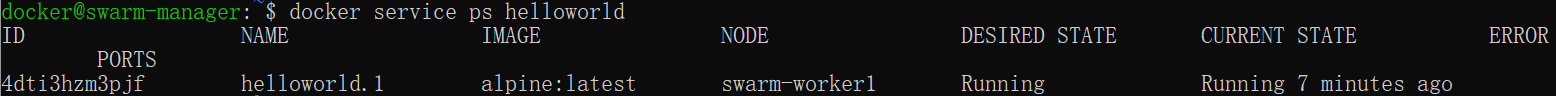
docker@swarm-manager:~$ docker service create --replicas 1 --name helloworld alpine ping docker.com

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm6.png)

### 5、查看服务部署情况

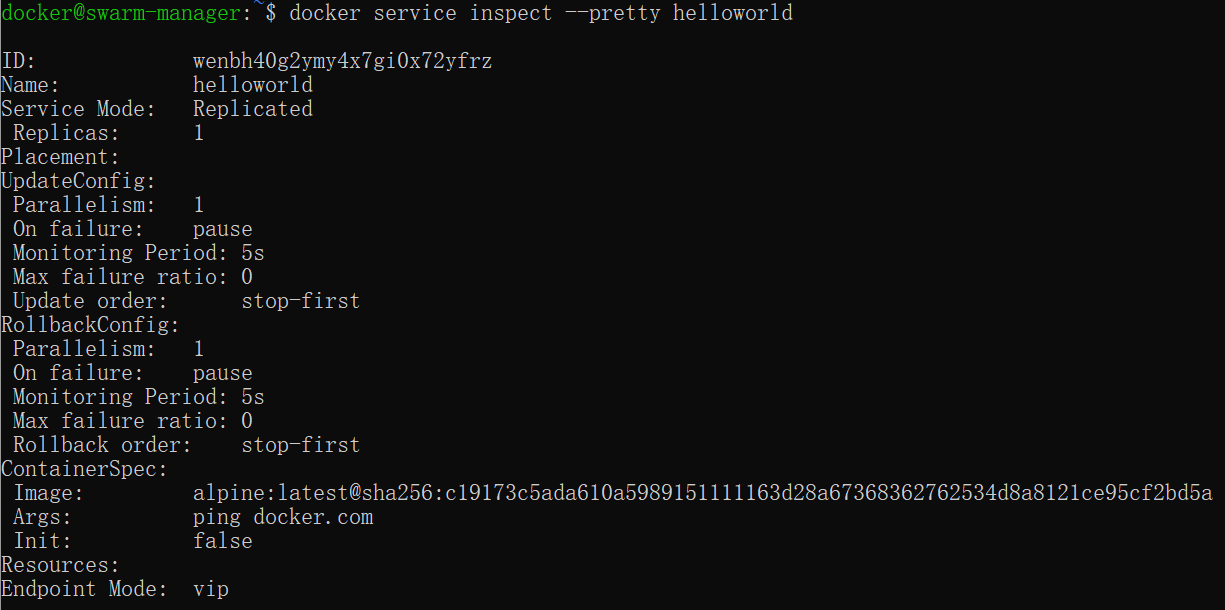
查看 helloworld 服务运行在哪个节点上，可以看到目前是在 swarm-worker1 节点：

docker@swarm-manager:~$ docker service ps helloworld

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm7.png)

查看 helloworld 部署的具体信息：

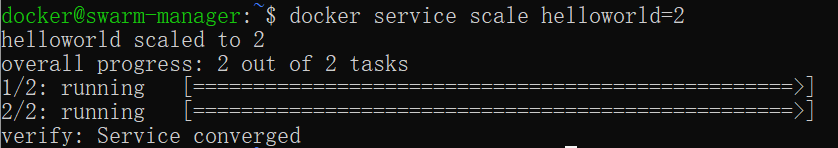
docker@swarm-manager:~$ docker service inspect --pretty helloworld

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm8.png)

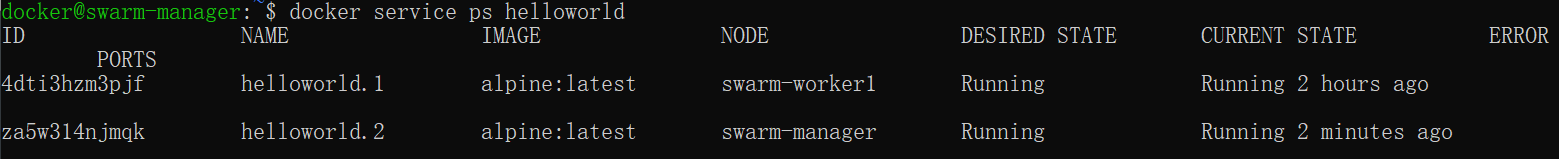
### 6、扩展集群服务

我们将上述的 helloworld 服务扩展到俩个节点。

docker@swarm-manager:~$ docker service scale helloworld=2

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm9.png)

可以看到已经从一个节点，扩展到两个节点。

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm10.png)

### 7、删除服务

docker@swarm-manager:~$ docker service rm helloworld

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm11.png)

查看是否已删除：

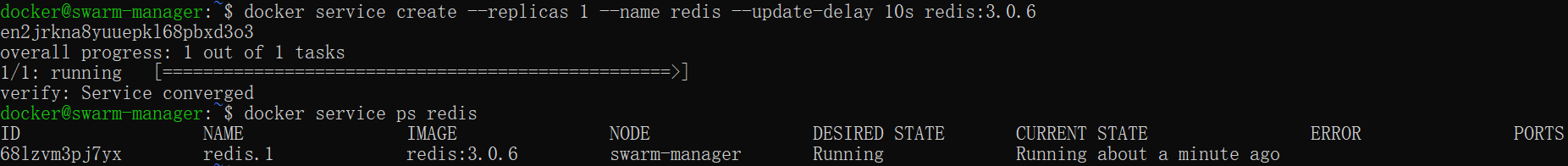
[](https://www.runoob.com/wp-content/uploads/2019/11/swarm12.png)

### 8、滚动升级服务

以下实例，我们将介绍 redis 版本如何滚动升级至更高版本。

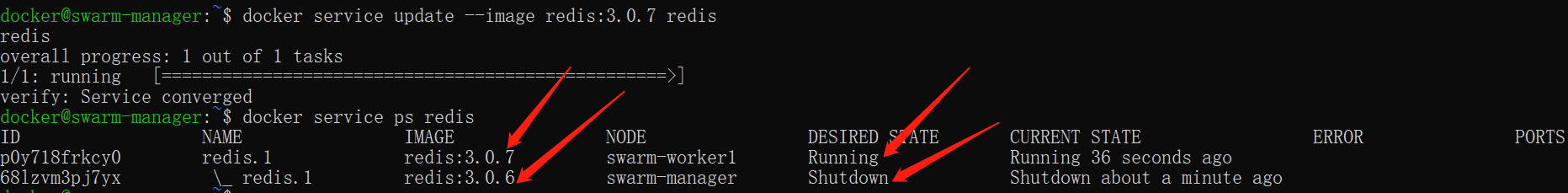
创建一个 3.0.6 版本的 redis。

docker@swarm-manager:~$ docker service create --replicas 1 --name redis --update-delay 10s redis:3.0.6

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm13.png)

滚动升级 redis 。

docker@swarm-manager:~$ docker service update --image redis:3.0.7 redis

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm14.png)

看图可以知道 redis 的版本已经从 3.0.6 升级到了 3.0.7，说明服务已经升级成功。

### 9、停止某个节点接收新的任务

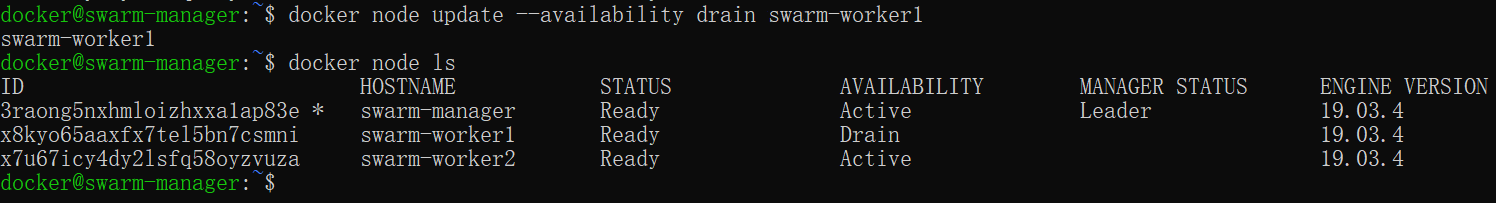
查看所有的节点：

docker@swarm-manager:~$ docker node ls

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm16.png)

可以看到目前所有的节点都是 Active, 可以接收新的任务分配。

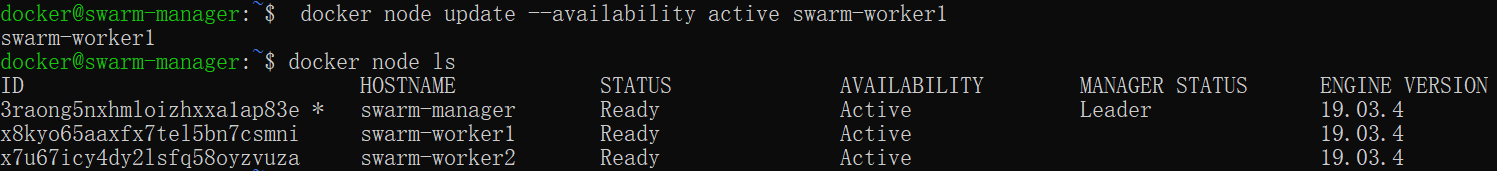
停止节点 swarm-worker1：

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm17.png)

**注意**：swarm-worker1 状态变为 Drain。不会影响到集群的服务，只是 swarm-worker1 节点不再接收新的任务，集群的负载能力有所下降。

可以通过以下命令重新激活节点：

docker@swarm-manager:~$ docker node update --availability active swarm-worker1

[](https://www.runoob.com/wp-content/uploads/2019/11/swarm19.png)