

Docker ABC

gudaoxuri

修订 1.0.0

Nov 6, 2015

贡献者

表 1. 贡献者名单

贡献者	所属	内容
gudaoxuri	杭州阿思拓电子商务有限公司	建立文档

目录

- 序言 xi
- 1. 什么是Docker 3
 - 1.1. 权威解释 3
 - 1.2. 通俗地说... 3
- 2. 核心概念 5
- 3. 基础 7
 - 3.1. 获取MySQL镜像 7
 - 3.2. 查看本机镜像 8
 - 3.3. 运行MySQL镜像 8
 - 3.4. 查看本机容器 8
 - 3.5. 删除MySQL容器 9
 - 3.6. 停止/运行/重启MySQL容器 10
 - 3.7. 完成 10
- 4. 进阶 11
 - 4.1. 获取Redis镜像并运行 11
 - 4.2. 获取MySQL镜像并运行 11
 - 4.3. 创建相应的数据库及权限 11
 - 4.4. 获取Gitlab镜像 12
 - 4.5. 运行Gitlab镜像 12
 - 4.6. 查看运行日志 14
 - 4.7. 完成 15
- 5. 高级 17
 - 5.1. 构建镜像的两种方法 17
 - 5.2. Dockerfile介绍 17
 - 5.3. 构建规划 17
 - 5.4. SSH镜像 17
 - 5.5. 编译SSH镜像 19
 - 5.6. Java镜像 19
 - 5.7. Scala镜像 20
 - 5.8. 编译Java/Scala镜像 20
 - 5.9. 发布镜像 20
 - 5.9.1. 使用github发布镜像 21
- 6. 网络 25
- 7. 生态系统 27
 - 7.1. Docker Toolbox 27
 - 7.2. Kitematic 27

7.3. pipework	27
7.4. Docker Swarm	28
8. 热点问题	29
8.1. 容器如何使用静态IP	29
8.2. 如何为运行中的容器增加访问端口	29
8.3. 让Docker容器使用静态独立的外部IP（便于集群组建）	29
9. 附	33
索引	35

插图清单

2.1. 关系	5
---------------	---

表格清单

1. 贡献者名单 iii

序言



Docker可谓是当下炙手可热的工具，如同Java当然提出的“Write once, run anywhere”类似 Docker所能实现的是“Building once run anywhere”，这是一个伟大的创举，当然Java的一个理念却被NodeJS打脸，这是后话。

这段时间要组织公司的Docker的培训，正好做个知识沉淀，故写此 快餐书，与市面上众多的Docker书籍相比，本书希望做到的是从速入门、上手、实战，关注怎么使用，对于背后的原理不会做这多的涉及。

gudaoxuri @ asto

2015-11-06

版权说明

Copyleft !

您可以自由使用、散布、改作，但也请保持开源共享。

第 1 章 什么是Docker

1.1. 权威解释

Docker is an open platform for building, shipping and running distributed applications. It gives programmers, development teams and operations engineers the common toolbox they need to take advantage of the distributed and networked nature of modern applications.

— docker.com

这是来自Docker官网的说明，其中有两层意思：

1.1.1. docker可以干什么？	3
1.1.2. 谁用docker？	3

1.1.1. docker可以干什么？

它可以用于编译（ building ）、分发（ shipping ）、运行（ running ）分布式应用。

1.1.2. 谁用docker？

程序员（ programmers ）、开发团队（ development teams ）、运维工程师（ operations engineers ）。

1.2. 通俗地说...



这货就是精简版的VM虚拟机，在牺牲了一定的资源隔离性的情况下实现了极小的资源占用、极快地启动/停止，它可以方便地共享/获取公共应用（ 镜像 ），通过 **Dockfile** 更可以方便地构建自己的应用（ 镜像 ）。

延伸阅读：[Docker与VM的区别](http://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-normal-virtual-machine)¹

¹ <http://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-normal-virtual-machine>

第 2 章 核心概念

在正式使用Docker前我们先简述几个核心概念。

Image (镜像)

与我们熟知的 Windows 镜像 (ISO或安装光盘) 类似，可以理解为一份特定应用的Copy，它是只读的。

Container (容器)

镜像是不能直接运行的，只有安装后才能使用，容器就是这个运行镜像后的实例，我们可以修改实例。

Dockerfile

用于制作镜像的元文件，有点类似 Maven 的 pom.xml 文件，可以通过它构建一个镜像。

hub.docker.com

镜像发布共享的地方，与 search.maven.org 类似。

它们之间的关系如下：

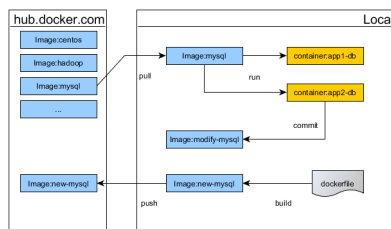


图 2.1. 关系

第 3 章 基础

本节我们以运行 MySQL 实例为例，介绍基础使用。



使用 `docker --help` 查看支持的所有命令

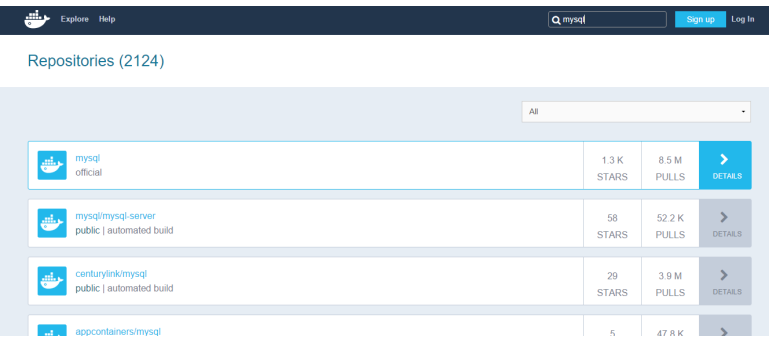
3.1. 获取MySQL镜像

- 首先我们从 `hub.docker.com` 上获取 MySQL 镜像

这里有两种方式，一是使用 `search` 命令：

```
root@ubuntu:~# docker search mysql
NAME                DESCRIPTION                STARS   OFFICIAL  AUTOMATED
mysql               MySQL is a widely used, open-source relational database management system. 1253    [OK]
mariadb             MariaDB is a community-developed fork of MySQL, focused on performance, reliability and high availability. 280    [OK]
mysql/mysql-server   Optimized MySQL Server Docker images. Created by MySQL AB or its affiliates. 58      [OK]
centurylink/mysql    Image containing mysql. Optimized to be lightweight. 29      [OK]
...
```

另外我们也可以在网页上查询更详细的说明：



STARS 表示收藏的数量，**PULLS** 表示下载的数量，**OFFICIAL** 表示是官方镜像（此处表示是否是MySQL官方发布）

- 我们下载第一个官方的镜像

```
root@ubuntu:~# docker pull mysql
```

```
Pulling repository mysql
196db1908492: Download complete
575489a51992: Download complete
...
Status: Downloaded newer image for mysql:latest
```



Docker镜像的命名格式 <用户名，官方镜像不需要/>镜像名称</版本号>

3.2. 查看本机镜像

- 查看本机的所有镜像

```
root@ubuntu:~# docker images
REPOSITORY      TAG          IMAGE ID      CREATED        VIRTUAL SIZE
mysql            5.7          196db1908492  6 days ago    359.8 MB
mysql            5.7.9        196db1908492  6 days ago    359.8 MB
mysql            latest       196db1908492  6 days ago    359.8 MB
...
```

3.3. 运行MySQL镜像

- 运行 MySQL 镜像

```
root@ubuntu:~# docker run --name app1-db -e MYSQL_ROOT_PASSWORD=123456 -d mysql:5.7
2dee953f85d697ffc78ac795f06e1a22731ae373b154178ea6220ce9b9c6e04
```



--name 指定容器名称， -e 指定特殊的一些变量， -d 表示后台运行（服务化）

3.4. 查看本机容器

- 查看本机的所有在运行容器

```
root@ubuntu:~# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS      NAMES
2dee953f85d6   mysql:5    "/entrypoint.sh mysq    6 seconds ago  Up 5 seconds  3306/tcp   app1-db
```



`docker ps -a` 可以查看所有容器（包含没有运行的）

- 然后呢，怎么连接？我们需要一个IP和端口，但上面的运行并没有对外暴露端口，所以我们要重新运行一个容器



Docker目前（v1.7）还不支持为容器绑定端口，如有此需求请查看[如何为运行中的容器增加访问端口](#)

3.5. 删除MySQL容器

- 我们先删除刚才的容器

```
root@ubuntu:~# docker rm -f app1-db
app1-db
```



`-f` 表示强制删除运行中的容器



这会删除容器中的所有数据。不科学呀？那我想删除容器时保留某些数据（如这里的数据库文件）怎么办？我们往下看：

- 重新运行 MySQL 镜像

```
root@ubuntu:~# docker run --name app1-db -e MYSQL_ROOT_PASSWORD=123456 -d -p 3308:3306
-v /opt/mysql_data:/var/lib/mysql mysql:5.7
1add96f289a1b8744500a4a6709af6e2e0628b97797e8470be75efe67e2005f4
```



`-p` 表示将外部访问的3308端口映射到容器的3306端口，由于MySQL镜像默认允许对外开放3306端口（我为什么知道？看它的`dockerfile`，后面会有说明），所以也可以直接用`-P`，这样的话外部访问3306也被映射到容器的3306端口上



`-v` 参数可以宿主机的目录映射到容器内目录，上例 `-v /opt/mysql_data:/var/lib/mysql` 会将宿主机 `/opt/mysql_data` 映射到容器 `/var/lib/mysql`（已知此目录为MySQL的数据目录），这样可以保证

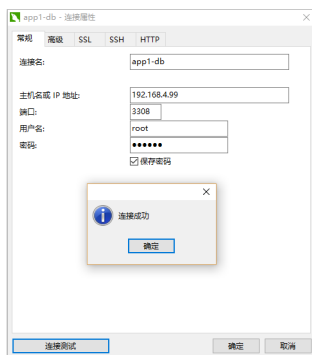
容器删除时对应目录的数据不被删除（因为这个目录是宿主机上的），这点很重要，实际使用中几乎所有容器都要做此映射！

- 有了端口之后那IP呢？IP实际上就是你docker宿主机的IP



Docker的访问机制简单说明：Docker所有的对外服务统一使用 宿主机 IP+特定端口，当一个请求发起时会经由宿主面的iptables根据容器创建时的端口规则转发到对应的容器处理。

- 好了，现在我们做个连接测试，通过



3.6. 停止/运行/重启MySQL容器

- 当然我们也可以对容器做 停止/运行/重启 操作：

```
root@ubuntu:~# docker stop app1-db
app1-db
root@ubuntu:~# docker start app1-db
app1-db
root@ubuntu:~# docker restart app1-db
app1-db
```

3.7. 完成

MySQL 容器如何升级/数据如何备份都方式都可以在获取到指导。

https://hub.docker.com/_/mysql/

至此，我们基础使用就介绍完了！

第 4 章 进阶

本节我们以运行 Gitlab 实例为例，进一步介绍 Docker 的常规使用。



上节 MySQL 是独立的镜像，没有依赖，但 Gitlab 会依赖数据库及缓存，我们看看容器间如何交互。Gitlab 有官方镜像，但我更喜欢 sameersbn 的版本，这个版本介绍详细，并且作者一直在持续更新。

4.1. 获取 Redis 镜像并运行

```
root@ubuntu:~# docker pull redis
...
root@ubuntu:~# docker run --name gitlab-cache -d redis
a2c183f16cee0aa30eca8f27875d3770ac1291fc22b218d4b5e673ce604074c8
```

4.2. 获取 MySQL 镜像并运行

```
root@ubuntu:~# mkdir -p /opt/test/gitlab/db/ ❶
root@ubuntu:~# docker pull mysql
...
root@ubuntu:~# docker run --name gitlab-db -e MYSQL_ROOT_PASSWORD=123456 -d -p 3307:3306
-v /opt/test/gitlab/db:/var/lib/mysql mysql:5.7
7c8110b8151e2ddbc67d2bb1d96a9cc24ef68546a885d065e78bd274775ec7f7
```

❶ 创建数据库目录映射

4.3. 创建相应的数据库及权限

```
root@ubuntu:~# docker exec -it gitlab-db /bin/bash
root@7c8110b8151e:/# mysql -uroot -p123456
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.9 MySQL Community Server (GPL)
```

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective

owners.

Type '**help**;' or '**\h**' for help. Type '**\c**' to clear the current input statement.

```
mysql> CREATE USER 'gitlab'@'%.%.%.%' IDENTIFIED BY 'password';  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE DATABASE IF NOT EXISTS `gitlabhq_production` DEFAULT CHARACTER SET `utf8`  
COLLATE `utf8_unicode_ci`;  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> GRANT ALL PRIVILEGES ON `gitlabhq_production`.* TO 'gitlab'@'%.%.%.%';  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> exit  
Bye  
root@7c8110b8151e:/# exit  
exit
```



`exec` 表示进入一个运行中的容器



MySQL中执行的语句见：<https://hub.docker.com/r/sameersbn/gitlab/#external-mysql-server>

4.4. 获取Gitlab镜像

```
root@ubuntu:~# mkdir -p /opt/test/gitlab/data ❶  
root@ubuntu:~# docker pull sameersbn/gitlab
```

❶ 创建GIT数据目录映射

4.5. 运行Gitlab镜像

```
root@ubuntu:~# docker run --name=gitlab -d \  
-e GITLAB_PORT=10080 \ ❶  
-e GITLAB_SSH_PORT=10022 \ ❷  
-e GITLAB_HOST=inner.ecfront.com \ ❸  
-e DB_TYPE=mysql \ ❹  
-e DB_HOST=192.168.4.99 \
```



```

-e DB_PORT=3307 \
-e DB_NAME=gitlabhq_production \
-e DB_USER=gitlab \
-e DB_PASS=password \
-e GITLAB_EMAIL=git@ecfront.com \ ❶
-e SMTP_ENABLED=true \
-e SMTP_DOMAIN=ecfront.com \
-e SMTP_HOST=smtp.exmail.qq.com \
-e SMTP_PORT=25 \
-e SMTP_USER=git@ecfront.com \
-e SMTP_PASS=xxx \
-e SMTP_AUTHENTICATION=login \
-e NGINX_MAX_UPLOAD_SIZE=512m \ ❷
-e GITLAB_BACKUPS=monthly \ ❸
-p 10022:22 \ ❹
-p 10080:80 \ ❺
-v /opt/test/gitlab/data:/home/git/data \ ❻
--link gitlab-cache:redisio \ ❼
--dns=192.168.4.99 \ ❽
sameersbn/gitlab

```

- ❶ HTTP服务端口
- ❷ SSH请求端口
- ❸ 服务域名
- ❹ 数据库连接信息
- ❺ Email信息
- ❻ 解决 RPC failed; result=22, HTTP code = 413 问题，见 <http://stackoverflow.com/questions/7489813/github-push-error-rpc-failed-result-22-http-code-413>
- ❼ 备份策略，每天
- ❽ SSH请求端口映射
- ❾ HTTP请求端口映射
- ❿ GIT数据目录映射
- ⓫ 连接Redis容器，实现缓存
- ⓬ 配置DNS



如果使用MySQL数据库一定要加上 `-e DB_TYPE=mysql` 否则这个容器会以默认的Postgre类型去连接导致无法启动



`--dns` 用于指定容器的DNS，我们可以指定一个公共DNS以实现不同容器互PING



Docker默认情况下容器间是不能互访的，要实现交互主要有两种途径，一是用 `--link` 这也是Docker推荐的做法，它会在容器中建立被Link容器的Host记录实现单向访问，另一种是开放服务实现调用，对于上述案例，Redis使用的是Link方式，MySQL使用是开放服务方式。

- 访问一下：<http://192.168.4.99:10080> (192.168.4.99是我宿主机的IP)，等等，为什么访问不了？

4.6. 查看运行日志

- 在出问题时我们首先想到的是 **去看看日志**，Docker容器运行日志的查看如下：

```
root@ubuntu:~# docker logs gitlab
ERROR:
Please configure the GITLAB_SECRETS_DB_KEY_BASE parameter.
Cannot continue. Aborting...
```

提示是需要 `GITLAB_SECRETS_DB_KEY_BASE` 变量，查看官方说明：

Note: Since GitLab 8.0.0 you need to provide the `GITLAB_SECRETS_DB_KEY_BASE` parameter while starting the image.

Tip: You can generate a random string using `pwgen -Bsv1 64` and assign it as the value of `GITLAB_SECRETS_DB_KEY_BASE`.

— <https://hub.docker.com/r/sameersbn/gitlab/>

- 好的，我们用 `pwgen -Bsv1 64` 生成一个密钥

```
root@ubuntu:~# pwgen -Bsv1 64
7hpTqCXgf4tVbnFmdC7PNn9n4hWmCnvF479fsJtcdTkhnVfWfzpwTJ4sNRzNkkXf
```


- 把 `GITLAB_SECRETS_DB_KEY_BASE` 加上重新运行

```
root@ubuntu:~# docker rm -f gitlab
root@ubuntu:~# docker run --name=gitlab -d \
-e GITLAB_PORT=10080 \
-e GITLAB_SSH_PORT=10022 \
-e GITLAB_HOST=inner.ecfront.com \
```

```
-e DB_TYPE=mysql \
-e DB_HOST=192.168.4.99 \
-e DB_PORT=3307 \
-e DB_NAME=gitlabhq_production \
-e DB_USER=gitlab \
-e DB_PASS=password \
-e
GITLAB_SECRETS_DB_KEY_BASE=7hpTqCXgf4tVbnFmdC7PNn9n4hWmCnvF479fsJtcdTkHmVfWfzpwTJ4sNRzNkkXf
\
-e GITLAB_EMAIL=git@ecfront.com \
-e SMTP_ENABLED=true \
-e SMTP_DOMAIN=ecfront.com \
-e SMTP_HOST=smtp.exmail.qq.com \
-e SMTP_PORT=25 \
-e SMTP_USER=git@ecfront.com \
-e SMTP_PASS=xxx \
-e SMTP_AUTHENTICATION=login \
-e NGINX_MAX_UPLOAD_SIZE=512m \
-e GITLAB_BACKUPS=monthly \
-p 10022:22 \
-p 10080:80 \
-v /opt/test/gitlab/data:/home/git/data \
--link gitlab-cache:redisio \
--dns=192.168.4.99 \
sameersbn/gitlab
```

4.7. 完成

- 好了，第一次要做的事情比较多，过一会就可以打开页面了



You need to sign in or sign up before continuing.

GitLab Community Edition

Open source software to collaborate on code

Manage git repositories with fine grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Existing user? Sign in

Username or Email

Password

☐ Remember me [Forgot your password?](#)

SIGN IN

New user? Create an account

Name

Username

Email

Password

SIGN UP

Didn't receive a confirmation email? [Request a new one.](#)



用户名 : root 密码 : 5iveL!fe

第 5 章 高级

本节我介绍如何自己构建一个Docker镜像，我们将构建一个支持scala的运行（<http://www.scala-lang.org>）环境，并且对外开放SSH服务。

5.1. 构建镜像的两种方法

构建镜像有两种方法：

1. 运行并进行一个基础容器（如centos，`docker run -it centos /bin/bash`），然后安装需要的环境，再用 `commit` 将容器保存成新的镜像
2. 使用 Dockerfile 构建，这也是推荐的做法

本节我们仅介绍第2种方法。

5.2. Dockerfile介绍

Dockerfile是用于构建Docker镜像的元文件，它的说明及语法见：
docs.docker.com/engine/reference/builder/

[http://](http://docs.docker.com/engine/reference/builder/)

5.3. 构建规划

根据需求可以发现我们安装的主要环境有三个：SSH服务、Java环境（Scala依赖它）、Scala环境，与软件开发一样，我们也可以分模块构建，即构建三个镜像：

1. 只包含SSH服务的基础镜像
2. 带SSH及Java环境的镜像
3. 带SSH、Java及Scala环境的镜像

5.4. SSH镜像

```
FROM centos:centos7 ❶
MAINTAINER gudaoxuri ❷

#-----Use 163 mirrors----- ❸
RUN yum install -y wget && \
    mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup && \
    wget -P /etc/yum.repos.d/ http://mirrors.163.com/help/CentOS7-Base-163.repo && \
    yum clean all && \
```

```

yum makecache ❹

#-----Install Common Tools-----
RUN yum install -y sed curl tar gcc gcc-c++ make git passwd sudo

#-----Modify Time Zone-----
ENV TZ "Asia/Shanghai" ❺
ENV TERM xterm

RUN yum install -y ntpdate && \
    cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

#-----Support Chinese-----
#RUN yum groupinstall -y "Fonts" && \
# echo "LANG=\"zh_CN.UTF-8\"" >> /etc/sysconfig/i18n

#-----Install SSH-----
RUN yum install -y openssh-server openssh-clients && \
    sed -i 's/UsePAM yes/UsePAM no/g' /etc/ssh/sshd_config && \
    echo 'root:123456' | chpasswd && \
    ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key && \
    ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key && \
    mkdir /var/run/ssh

#-----Setting Common Path-----
RUN chmod 777 -R /opt/ && \
    mkdir /opt/env/ && \
    mkdir /opt/workspaces/

EXPOSE 22 ❻

CMD ["/usr/sbin/sshd", "-D"] ❼

```

- ❶ FROM 表示此镜像是基于哪个镜像构建的
- ❷ MAINTAINER 开发者的信息
- ❸ # 注释
- ❹ RUN 常用命令，用于执行Linux的命令，多用于安装组件
- ❺ ENV 常用命令，用于设置环境变量
- ❻ EXPOSE 表明默认对外暴露的端口(docker run -P)
- ❼ CMD 要执行的服务，多条 CMD 只执行最后一条



RUN 参数的学问，Docker镜像是分层的，一个镜像可能由多个层组成，一次 RUN 实际上就产生了一层，在编译 Dockerfile 的过程中可能出错，重新编译时我们会发现之前已成功编译的层不会再次被编译

5.5. 编译SSH镜像

```

root@ubuntu:/opt/test/dockerfile/ssh# ls
Dockerfile
root@ubuntu:/opt/test/dockerfile/ssh# docker build -t gudaoxuri/ssh .
Sending build context to Docker daemon 3.072 kB
Sending build context to Docker daemon
Step 0 : FROM centos:centos7
Pulling repository centos
ce20c473cd8a: Download complete
ce20c473cd8a: Pulling image (centos7) from centos
168a69b62202: Download complete
812e9d9d677f: Download complete
4234bfdd88f8: Download complete
Status: Downloaded newer image for centos:centos7
---> ce20c473cd8a
Step 1 : MAINTAINER gudaoxuri
---> Running in 889ea744c458
---> 5b1151e6cb0b
Removing intermediate container 889ea744c458

...

Step 10 : CMD /usr/sbin/sshd -D
---> Running in ce563073b686
---> b61a4adad85f
Removing intermediate container ce563073b686
Successfully built b61a4adad85f

root@ubuntu:/opt/test/dockerfile/ssh# docker images

```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
gudaoxuri/ssh	latest	b61a4adad85f	10 minutes ago	966 MB

```

...

```

5.6. Java镜像

```

FROM gudaoxuri/ssh:latest ❶
MAINTAINER gudaoxuri

#-----Install Java-----
RUN wget -P /opt/env/ --no-check-certificate --no-cookies --header "Cookie: oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/8u60-b27/jdk-8u60-linux-x64.tar.gz
&& \
    tar -xzf /opt/env/jdk-8u60-linux-x64.tar.gz -C /opt/env/ && \

```

```
rm /opt/env/jdk-8u60-linux-x64.tar.gz && \
mv /opt/env/jdk1.8.0_60 /opt/env/java && \
echo "export JAVA_HOME=/opt/env/java" >> /etc/profile
ENV JAVA_HOME /opt/env/java

RUN echo 'PATH=$PATH:$JAVA_HOME/bin' >> /etc/profile
ENV PATH $PATH:$JAVA_HOME/bin
```

❶ `gudaoxuri/ssh:latest` 就是我们之前编译的镜像

5.7. Scala镜像

```
FROM gudaoxuri/java:latest
MAINTAINER gudaoxuri

#-----Install Scala-----
RUN wget -P /opt/env/ http://downloads.typesafe.com/scala/2.10.6/scala-2.10.6.tgz && \
  tar -xzf /opt/env/scala-2.10.6.tgz -C /opt/env/ && \
  rm /opt/env/scala-2.10.6.tgz && \
  mv /opt/env/scala-2.10.6 /opt/env/scala && \
  echo "export SCALA_HOME=/opt/env/scala" >> /etc/profile
ENV SCALA_HOME /opt/env/scala

RUN sed /^PATH=/d /etc/profile >> /etc/profile && \
  echo 'PATH=$PATH:$JAVA_HOME/bin:$SCALA_HOME/bin' >> /etc/profile
ENV PATH $PATH:$JAVA_HOME/bin:$SCALA_HOME/bin
```

5.8. 编译Java/Scala镜像

过程同上，略

5.9. 发布镜像

有了新的镜像后我们希望把这个镜像发布到 `hub.docker.com` 上去分享，有两种方式：

1. 使用 `docker push` 发布前要先登录 `docker login`
2. 使用 `github` 发布



鉴于国内网络环境恶劣，第1种方式失败概率很高，因为它上传的是生成的镜像（几百MB到几G不等），所以推荐使用第2种方式



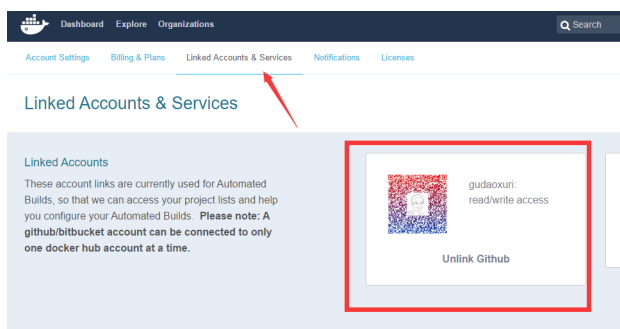
请先注册 `hub.docker.com` 及 `github.com` 的账号

5.9.1. 使用github发布镜像

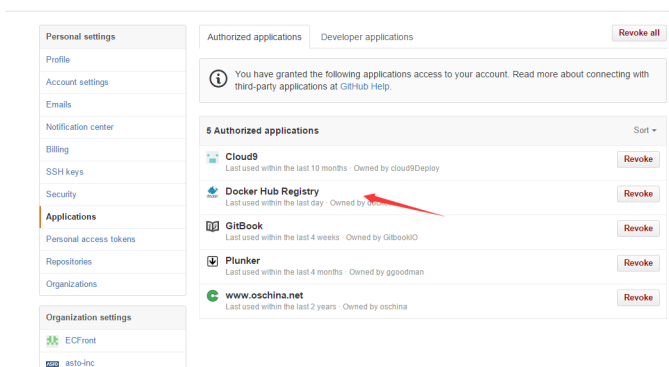


不要用 IE 操作，笔者使用 IE11 在 `hub.docker.com` 上操作时发生过不小的困扰。

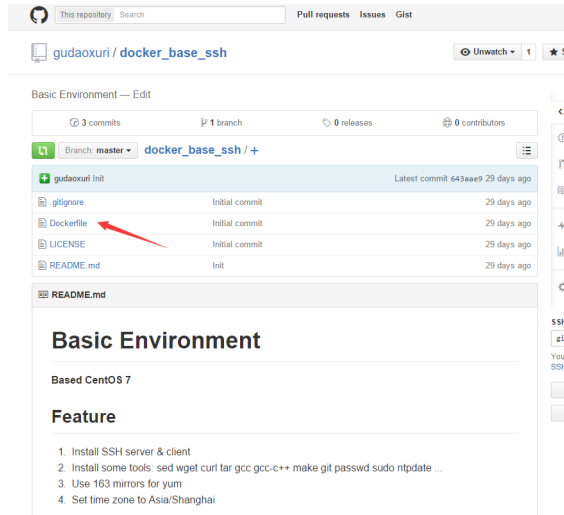
- 在 `hub.docker.com` 上建立 `github` 连接



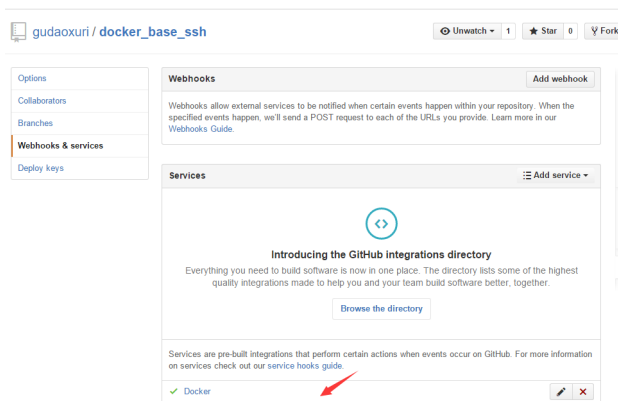
- 在 `github` 中设置权限



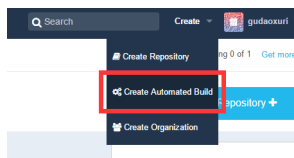
- 在 `github` 中建一个开源项目，注意要包含 `Dockerfile` 文件



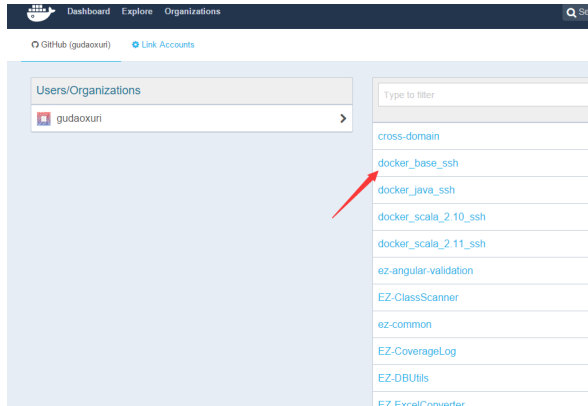
- 在 github 中设置这个项目的权限



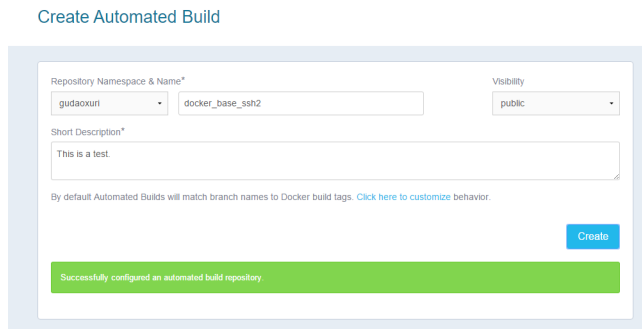
- 在 hub.docker.com 上建立自动构建项目



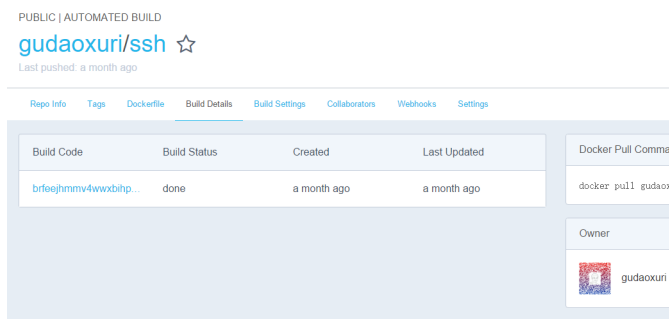
- 在 hub.docker.com 上选择 github 上的项目



- 在 `hub.docker.com` 上设置项目属性



- 不多时就构建好了



更详细的说明见：<https://docs.docker.com/docker-hub/github/>

第 6 章 网络

Docker的很多问题都是网络设置引发的，所以这里有必要介绍一下Docker的网络处理。

在容器运行时Docker提供了 `--net` 参数来指定容器的网络，它有四种选项：

1. `--net=bridge` 这是默认选项，会为容器分配独立的网络环境
2. `--net=host` 让容器使用宿主机的网络环境，**注意** 这意味着放弃了容器的网络隔离，它让容器可以访问宿主机的权限，不推荐使用
3. `--net=container:NAME_or_ID` 让容器与另一个容器共用网络环境，这样的话两者就可以相互通信
4. `--net=none` Docker不去管理容器的网络，用户可以自定义容器的网络设置，这选项自由度最高

第 7 章 生态系统

随着Docker如火如荼地发展，它的应用场景也越来越广泛，群众的想像力是无限的，这导致了Docker本身的功能往往无法满足人们凶残的需求，所以产生了一大堆的扩展应用，Docker与这些扩展应用、“寄生”公司相辅相成，共同发展。

下面介绍几个不错的扩展

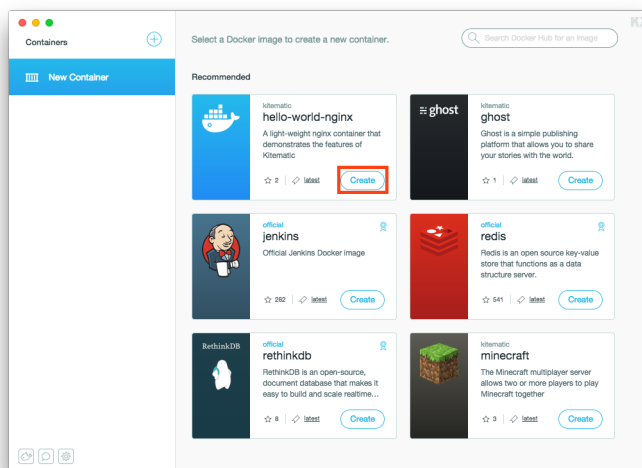
7.1. Docker Toolbox

请您在Windows下使用Docker，实际上就是安装了个Visual Box ~~

<https://www.docker.com/docker-toolbox>

7.2. Kitematic

Docker官方推荐的GUI工具



<https://kitematic.com/>

7.3. pipework

一个功能多多的网络扩展工具包

<https://github.com/jpetazzo/pipework>

7.4. Docker Swarm

让Docker运行在多个宿主机上

<http://docs.docker.com/swarm/install-w-machine/>

第 8 章 热点问题

8.1. 容器如何使用静态IP

默认情况下Docker容器的IP是动态分配的，要使用静态IP时我们会思考一下：

为什么需要静态IP？如果是为了两个容器间通信可以

- a. 使用 `--link`
- b. 指定 `-h` 来指定hostname并指定 `-dns` 到宿主机
- c. 让容器开放上层服务

如果这样都满足不了您的要求那么可以参考 <http://huataihuang.github.io/2014/10/05/docker-container-static-ip/> 此文实现

8.2. 如何为运行中的容器增加访问端口

```
iptables -t nat -A DOCKER -p tcp --dport <外部映射端口> -j DNAT --to-destination <宿主机IP>:<容器新加的端口>
```

<http://stackoverflow.com/questions/19897743/exposing-a-port-on-a-live-docker-container>

8.3. 让Docker容器使用静态独立的外部IP（便于集群组建）

需要使用Docker虚拟化Hadoop/Spark等测试环境，并且要可以对外提供服务，要求是完全分布式的部署（尽量模拟生产环境）。那么我们会遇到几个问题：

- a. Container IP 是动态分配的
- b. Container IP 是内部IP，外部无法访问（如对外提供HDFS服务可能会遇到Client无法访问DataNode，因为DataNode注册的是内部IP）

针对第一个问题有不少的方案，可以指定静态的IP，对第二个问题，我们可以使用`--net=host`解决，但这会导致对外只有一个IP，集群各个Slave的端口都要修改。至于pipework简单地看了下，好像也解决不了。

让Docker容器使用静态独立的外部IP（便于集群组建）

所以目前看上去只能使用看上去不是很优雅的方案解决：***为Docker宿主网卡绑定多个IP，把这些IP分配给不同的容器。**

```
//这是示例，我是在windows下用Docker toolbox运行的
root@default:~# ifconfig
docker0  Link encap:Ethernet HWaddr 02:42:8C:8E:80:F1
        inet addr:172.17.42.1 Bcast:0.0.0.0 Mask:255.255.0.0
        UP BROADCAST MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

eth0     Link encap:Ethernet HWaddr 08:00:27:24:D1:F5
        inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:fe24:d1f5/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:222 errors:0 dropped:0 overruns:0 frame:0
        TX packets:164 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:32277 (31.5 KiB) TX bytes:28136 (27.4 KiB)

eth1     Link encap:Ethernet HWaddr 08:00:27:76:1D:9B
        inet addr:192.168.99.100 Bcast:192.168.99.255 Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:fe76:1d9b/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:66 errors:0 dropped:0 overruns:0 frame:0
        TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:9001 (8.7 KiB) TX bytes:10469 (10.2 KiB)

lo       Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING MTU:65536 Metric:1
        RX packets:16 errors:0 dropped:0 overruns:0 frame:0
        TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:1152 (1.1 KiB) TX bytes:1152 (1.1 KiB)

//eth1网卡是可以与外部交互，所以我们添加IP到这个网卡上
//第一步：添加了两个IP
root@default:~# ifconfig eth1:0 192.168.99.10 netmask 255.255.255.0 up
root@default:~# ifconfig eth1:1 192.168.99.11 netmask 255.255.255.0 up
//再次查看，多了两个IP
root@default:~# ifconfig
...
```

让Docker容器使用静态独立的外部IP（便于集群组建）

```
eth1    Link encap:Ethernet HWaddr 08:00:27:76:1D:9B
        inet addr:192.168.99.100 Bcast:192.168.99.255 Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:fe76:1d9b/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:2258 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1685 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:207033 (202.1 KiB) TX bytes:209587 (204.6 KiB)
```

```
eth1:0  Link encap:Ethernet HWaddr 08:00:27:76:1D:9B
        inet addr:192.168.99.10 Bcast:192.168.99.255 Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```
eth1:1  Link encap:Ethernet HWaddr 08:00:27:76:1D:9B
        inet addr:192.168.99.11 Bcast:192.168.99.255 Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

...

//第二步：运行容器，指定IP，这里的示例容器开启的SSH服务，后面拿它测试

```
root@default:~# docker run -d -p 192.168.99.10:222:22 --name ssh1 gudaoxuri/scala-2.11-env
```

```
root@default:~# docker run -d -p 192.168.99.11:222:22 --name ssh2 gudaoxuri/scala-2.11-env
```

```
root@default:~# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
ab024af9c954	gudaoxuri/scala-2.11-env	"/usr/sbin/sshd -D"	4 seconds ago	Up 3 seconds	
192.168.99.11:222->22/tcp	ssh2				
259351134d16	gudaoxuri/scala-2.11-env	"/usr/sbin/sshd -D"	15 seconds ago	Up 14 seconds	
192.168.99.10:222->22/tcp	ssh1				

//测试连接，在Docker宿主机上SSH到第一个容器

```
root@default:~# ssh 192.168.99.10 -p222
```

```
The authenticity of host '[192.168.99.10]:222 ([192.168.99.10]:222)' can't be established.
```

```
RSA key fingerprint is ac:fe:4b:89:f8:51:b7:e9:9c:34:62:f9:80:38:4b:bf.
```

```
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '[192.168.99.10]:222' (RSA) to the list of known hosts.
```

```
root@192.168.99.10's password:
```

```
Last login: Wed Oct 7 13:12:35 2015 from 192.168.99.1
```

//成功进入

```
[root@259351134d16 ~]#
```

//在第一个容器中SSH到第二个容器

```
[root@259351134d16 ~]# ssh 192.168.99.11 -p222
```

```
root@192.168.99.11's password:
```

```
Last login: Wed Oct 7 13:14:53 2015 from 172.17.42.1
```

//也OK了

```
[root@ab024af9c954 ~]#
```

让Docker容器使用静态独立 的外部IP（便于集群组建）

参考我的Blog <http://my.oschina.net/gudaoxuri/blog/513923>

第 9 章 附

Docker文档 <https://docs.docker.com/>

Docker实战案例视频课程
[course_id-4238.html](http://edu.51cto.com/course/course_id-4238.html)

此课程很不错，推荐

<http://edu.51cto.com/course/>

索引

D

- docker commit , 17
- docker exec , 12
- dockerfile # , 18
- dockerfile CMD , 18
- dockerfile ENV , 18
- dockerfile EXPOSE , 18
- dockerfile FROM , 18
- dockerfile MAINTAINER , 18
- dockerfile RUN , 18
- docker images , 8
- docker login , 20
- docker logs , 14
- docker ps , 8 , 9
- docker pull , 7
- docker push , 20
- docker restart , 10
- docker rm , 9
- docker run , 8 , 12
- docker search , 7
- docker start , 10
- docker stop , 10

