2013年发布至今， Docker 一直广受瞩目，被认为可能会改变软件行业。
但是，许多人并不清楚 Docker 到底是什么，要解决什么问题，好处又在哪里？今天就来详细解释，帮助大家理解它，还带有简单易懂的实例，教你如何将它用于日常开发。

Docker简介

Docker是一个开源的**容器引擎**，它有助于更快地交付应用。 Docker可将应用程序和基础设施层隔离，并且能将基础设施当作程序一样进行管理。使用 Docker可更快地打包、测试以及部署应用程序，并可以缩短从编写到部署运行代码的周期。

Docker的优点如下：

1、简化程序

Docker 让开发者可以打包他们的应用以及依赖包到一个**可移植**的容器中，然后发布到任何流行的 Linux 机器上，便可以实现虚拟化。Docker改变了虚拟化的方式，使开发者可以直接将自己的成果放入Docker中进行管理。方便快捷已经是 Docker的最大优势，过去需要用数天乃至数周的 任务，在Docker容器的处理下，只需要数秒就能完成。

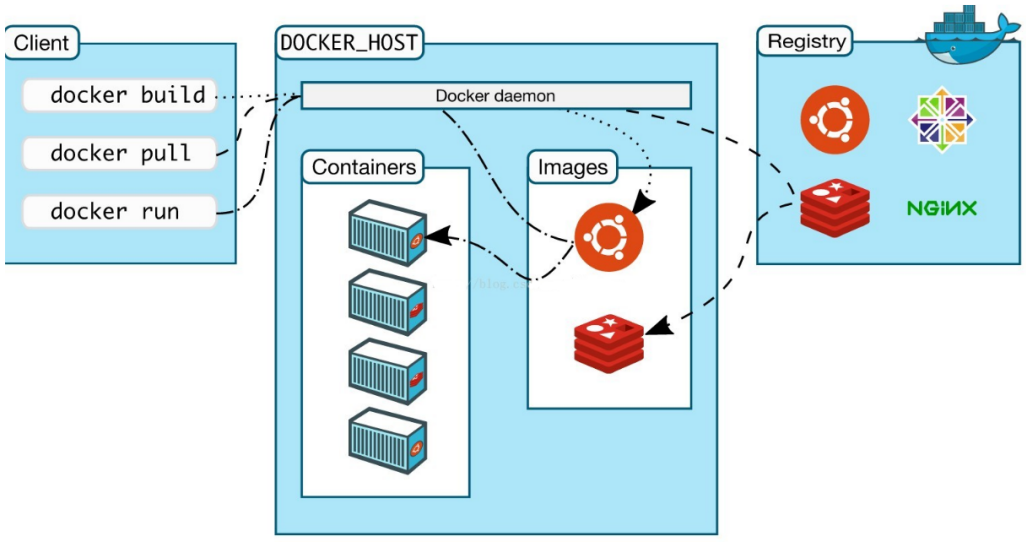
2、避免选择恐惧症

如果你有选择恐惧症，还是资深患者。Docker 帮你 打包你的纠结！比如 Docker 镜像；Docker 镜像中包含了运行环境和配置，所以 Docker 可以简化部署多种应用实例工作。比如 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

3、节省开支

一方面，云计算时代到来，使开发者不必为了追求效果而配置高额的硬件，Docker 改变了高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化的方式。

Docker的架构



- **Docker daemon（ Docker守护进程）**

Docker daemon是一个运行在宿主机（ DOCKER-HOST）的后台进程。可通过 Docker客户端与之通信。

- **Client （ Docker客户端)**

Docker客户端是 Docker的用户界面，它可以接受用户命令和配置标识，并与 Docker daemon通信。图中， docker build 等都是 Docker的相关命令。

- **Images （ Docker镜像)**

Docker镜像是一个只读模板，它包含创建 Docker容器的说明。**它和系统安装光盘有点像**，使用系统安装光盘可以安装系统，同理，使用Docker镜像可以运行 Docker镜像中的程序。

- **Container（容器）**

容器是镜像的可运行实例。**镜像和容器的关系有点类似于面向对象中，类和对象的关系。**可通过 Docker API或者 CLI命令来启停、移动、删除容器。

- **Registry**

Docker Registry是一个集中存储与分发镜像的服务。构建完 Docker镜像后，就可在当前宿主机上运行。但如果想要在其他机器上运行这个镜像，就需要手动复制。此时可借助 Docker Registry来避免镜像的手动复制。

一个 Docker Registry可包含多个 Docker仓库，每个仓库可包含多个镜像标签，每个标签对应一个 Docker镜像。这跟Maven的仓库有点类似，如果把 Docker Registry比作 Maven仓库的话，那么 Docker仓库就可理解为某jar包的路径，而镜像标签则可理解为jar包的版本号。

Docker Registry可分为公有Docker Registry和私有Docker Registry。 最常用的Docker Registry莫过于官方的Docker Hub， 这也是默认的Docker Registry。 Docker Hub上存放着大量优秀的镜像， 我们可使用Docker命令下载并使用。

Docker 的安装

Docker 是一个开源的商业产品，有两个版本：社区版（Community Edition，缩写为 CE）和企业版（Enterprise Edition，缩写为 EE）。企业版包含了一些收费服务，个人开发者一般用不到。下面的介绍都针对社区版。

Docker CE 的安装请参考官方文档，**我们这里以CentOS为例：**

1、Docker 要求 CentOS 系统的内核版本高于 3.10

通过 uname -r 命令查看你当前的内核版本

```
uname -r
```

2、使用 root 权限登录 Centos。确保 yum 包更新到最新。

```
yum -y update
```

3、卸载旧版本(如果安装过旧版本的话)

```
sudo yum remove -y docker*
```

4、安装需要的软件包， yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的

```
yum install -y yum-utils
```

5、设置yum源，并更新 yum 的包索引

```
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
yum makecache fast
```

```
[root@localhost local]# sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
已加载插件：fastestmirror, langpacks
adding repo from: https://download.docker.com/linux/centos/docker-ce.repo
grabbing file https://download.docker.com/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
```

6、可以查看所有仓库中所有docker版本，并选择特定版本安装

```
yum list docker-ce --showduplicates | sort -r
```

```
[root@192-168-65-160 ~]# docker -version^C
[root@192-168-65-160 ~]# yum list docker-ce --showduplicates | sort -r
已加载插件：fastestmirror, langpacks
可安装的软件包
 * updates: mirrors.163.com
Loading mirror speeds from cached hostfile
 * extras: mirrors.163.com
docker-ce.x86_64            3:20.10.6-3.el7                     docker-ce-stable
docker-ce.x86_64            3:20.10.5-3.el7                     docker-ce-stable
docker-ce.x86_64            3:20.10.4-3.el7                     docker-ce-stable
docker-ce.x86_64            3:20.10.3-3.el7                     docker-ce-stable
docker-ce.x86_64            3:20.10.2-3.el7                     docker-ce-stable
docker-ce.x86_64            3:20.10.1-3.el7                     docker-ce-stable
docker-ce.x86_64            3:20.10.0-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.9-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.8-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.7-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.6-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.5-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.4-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.3-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.2-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.15-3.el7                    docker-ce-stable
docker-ce.x86_64            3:19.03.14-3.el7                    docker-ce-stable
docker-ce.x86_64            3:19.03.1-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.13-3.el7                    docker-ce-stable
docker-ce.x86_64            3:19.03.12-3.el7                    docker-ce-stable
docker-ce.x86_64            3:19.03.11-3.el7                    docker-ce-stable
docker-ce.x86_64            3:19.03.10-3.el7                    docker-ce-stable
```

7、安装docker

```
yum install -y docker-ce-3:19.03.9-3.el7.x86_64 # 这是指定版本安装
```

8、启动并加入开机启动

```
systemctl start docker && systemctl enable docker
```

9、验证安装是否成功(有client和service两部分表示docker安装启动都成功了)

```
docker version
```

```
[root@192-168-65-160 docker]# docker version
Client: Docker Engine - Community
 Version:           19.03.9
 API version:       1.40
 Go version:        go1.13.10
 Git commit:        9d988398e7
 Built:             Fri May 15 00:25:27 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.9
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.13.10
  Git commit:       9d988398e7
  Built:            Fri May 15 00:24:05 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.4
  GitCommit:        05f951a3781f4f2c1911b05e61c160e9c30eaa8e
 runc:
  Version:          1.0.0-rc93
  GitCommit:        12644e614e25b05da6fd08a38ffa0cfe1903fdec
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

注意：一般需要配置docker镜像加速器

我们可以借助阿里云的镜像加速器，登录阿里云(https://cr.console.aliyun.com/#/accelerator)

可以看到镜像加速地址如下图：


```
cd /etc/docker
```

查看有没有 daemon.json。这是docker默认的配置文件。

如果没有新建，如果有，则修改。

```
vim daemon.json
{
  "registry-mirrors": ["https://m9r2r2uj.mirror.aliyuncs.com"]
}
```

保存退出。

重启docker服务

```
systemctl daemon-reload
systemctl restart docker
```

成功!

10、卸载docker

```
yum remove -y docker*
rm -rf /etc/systemd/system/docker.service.d
rm -rf /var/lib/docker
rm -rf /var/run/docker
```

Docker常用命令

镜像相关命令

1、搜索镜像

可使用 docker search命令搜索存放在 Docker Hub中的镜像。执行该命令后， Docker就会在Docker Hub中搜索含有 java这个关键词的镜像仓库。

```
docker search java
```

```
[root@centos-new ~]# docker search java
NAME                                      DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
node                                      Node.js is a JavaScript-based platform for s…   5720                [OK]
tomcat                                    Apache Tomcat is an open source implementati…   1890                [OK]
java                                      Java is a concurrent, class-based, and objec…   1745                [OK]
openjdk                                   OpenJDK is an open-source implementation of …   1021                [OK]
ghost                                     Ghost is a free and open source blogging pla…   778                 [OK]
anapsix/alpine-java                       Oracle Java 8 (and 7) with GLIBC 2.23 over A…   322                                     [OK]
jetty                                     Jetty provides a Web server and javax.servle…   251                 [OK]
couchdb                                   CouchDB is a database that uses JSON for doc…   210                 [OK]
tomee                                     Apache TomEE is an all-Apache Java EE certif…   51                  [OK]
ibmjava                                   Official IBM® SDK, Java™ Technology Edition …   47                  [OK]
groovy                                    Apache Groovy is a multi-faceted language fo…   44                  [OK]
lwieske/java-8                            Oracle Java 8 Container - Full + Slim - Base…   38                                      [OK]
cloudbees/jnlp-slave-with-java-build-tools Extends cloudbees/java-build-tools docker im…   17                                      [OK]
zabbix/zabbix-java-gateway                Zabbix Java Gateway                             12                                      [OK]
davidcaste/alpine-java-unlimited-jce      Oracle Java 8 (and 7) with GLIBC 2.21 over A…   11                                      [OK]
frekele/java                              docker run --rm --name java frekele/java        9                                       [OK]
blacklabelops/java                        Java Base Images.                               8                                       [OK]
fabric8/s2i-java                          S2I Builder Image for plain Java applications   5
rightctrl/java                            Oracle Java                                     2                                       [OK]
dwolla/java                               Dwolla's custom Java image                      1                                       [OK]
appuio/s2i-maven-java                     S2I Builder with Maven and Java                 1                                       [OK]
thingswise/java-docker                    Java + dcd                                      0                                       [OK]
cfje/java-buildpack                       Java Buildpack CI Image                         0
cfje/java-test-applications               Java Test Applications CI Image                 0
appuio/s2i-gradle-java                    S2I Builder with Gradle and Java                0                                       [OK]
```

以上列表包含五列，含义如下：

- NAME:镜像仓库名称。
- DESCRIPTION:镜像仓库描述。
- STARS：镜像仓库收藏数，表示该镜像仓库的受欢迎程度，类似于 GitHub的 stars0
- OFFICAL:表示是否为官方仓库，该列标记为[0K]的镜像均由各软件的官方项目组创建和维护。
- AUTOMATED：表示是否是自动构建的镜像仓库。

2、下载镜像

使用命令docker pull命令即可从 Docker Registry上下载镜像，执行该命令后，Docker会从 Docker Hub中的 java仓库下载最新版本的 Java镜像。如果要下载指定版本则在java后面加冒号指定版本，例如：docker pull java:8

```
1 docker pull java:8
```

```
[root@centos-new ~]# docker pull java:8
8: Pulling from library/java
5040bd298390: Pull complete
fce5728aad85: Pull complete
76610ec20bf5: Pull complete
60170fec2151: Pull complete
e98f73de8f0d: Pull complete
11f7af24ed9c: Pull complete
49e2d6393f32: Pull complete
bb9cdec9c7f3: Pull complete
Digest: sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66b7f8c0e9d
Status: Downloaded newer image for java:8
```

```
1 docker pull nginx
```

```
[root@192-168-65-42 eureka]# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:353c20f74d9b6aee359f30e8e4f69c3d7eaea2f610681c4a95849a2fd7c497f9
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

3、列出镜像

使用 docker images命令即可列出已下载的镜像

```
1 docker images
```

```
[root@centos-new ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
java                8                   d23bdf5b1b1b        17 months ago       643MB
```

以上列表含义如下

- REPOSITORY：镜像所属仓库名称。
- TAG:镜像标签。默认是 latest,表示最新。
- IMAGE ID：镜像 ID，表示镜像唯一标识。
- CREATED：镜像创建时间。

- SIZE: 镜像大小。

4、删除本地镜像

使用 docker rmi命令即可删除指定镜像，强制删除加 -f

```
docker rmi java
```

删除所有镜像

```
docker rmi $(docker images -q)
```

容器相关命令

1、新建并启动容器

使用以下docker run命令即可新建并启动一个容器，该命令是最常用的命令，它有很多选项，下面将列举一些常用的选项。

-d选项：表示后台运行

-P选项：随机端口映射

-p选项：指定端口映射，有以下四种格式。

-- ip:hostPort:containerPort

-- ip::containerPort

-- hostPort:containerPort

-- containerPort

--net选项：指定网络模式，该选项有以下可选参数：

--net=bridge:**默认选项**，表示连接到默认的网桥。

--net=host:容器使用宿主机的网络。

--net=container:NAME-or-ID：告诉 Docker让新建的容器使用已有容器的网络配置。

--net=none：不配置该容器的网络，用户可自定义网络配置。

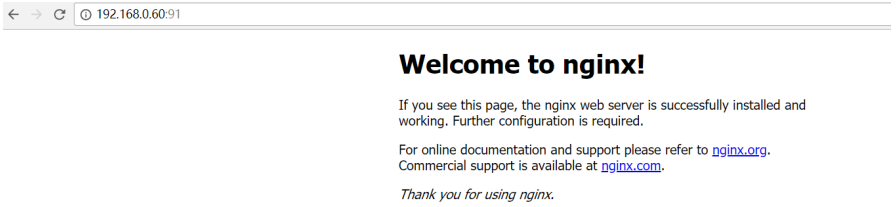
```
docker run -d -p 91:80 nginx
```

这样就能启动一个 Nginx容器。在本例中，为 docker run添加了两个参数，含义如下：

-d 后台运行

-p 宿主机端口:容器端口 #开放容器端口到宿主机端口

访问 http://Docker宿主机 IP:91/，将会看到nginx的主界面如下：



需要注意的是，使用 docker run命令创建容器时，会先检查本地是否存在指定镜像。如果本地不存在该名称的镜像， Docker就会自动从 Docker Hub下载镜像并启动一个 Docker容器。

2、列出容器

用 docker ps命令即可列出**运行中**的容器

```
docker ps
```

```
[root@centos-new ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                  NAMES
f0b1c8ab3633        nginx               "nginx -g 'daemon of…"   About a minute ago   Up About a minute   0.0.0.0:91->80/tcp   xenodochial_neumann
```

如需列出所有容器（包括已停止的容器），可使用-a参数。该列表包含了7列，含义如下

- CONTAINER_ID：表示容器 ID。
- IMAGE:表示镜像名称。
- COMMAND：表示启动容器时运行的命令。
- CREATED：表示容器的创建时间。
- STATUS：表示容器运行的状态。UP表示运行中， Exited表示已停止。
- PORTS:表示容器对外的端口号。
- NAMES:表示容器名称。该名称默认由 Docker自动生成，也可使用 docker run命令的--name选项自行指定。

3、停止容器

使用 docker stop命令，即可停止容器

```
docker stop f0b1c8ab3633
```

其中f0b1c8ab3633是容器 ID,当然也可使用 docker stop容器名称来停止指定容器

4、强制停止容器

可使用 docker kill命令发送 SIGKILL信号来强制停止容器

```
docker kill f0b1c8ab3633
```

5、启动已停止的容器

使用docker run命令，即可**新建**并启动一个容器。对于已停止的容器，可使用 docker start命令来**启动**

```
docker start f0b1c8ab3633
```

6、查看容器所有信息

```
docker inspect f0b1c8ab3633
```

7、查看容器日志

```
docker container logs f0b1c8ab3633
```

8、查看容器里的进程

```
docker top f0b1c8ab3633
```

9、容器与宿主机相互复制文件

- 从容器里面拷文件到宿主机：

```
docker cp 容器id:要拷贝的文件在容器里面的路径 宿主机的相应路径
如：docker cp 7aa5dc458f9d:/etc/nginx/nginx.conf /mydata/nginx
```

- 从宿主机拷文件到容器里面：

```
docker cp 要拷贝的宿主机文件路径 容器id:要拷贝到容器里面对应的路径
```

10、进入容器

使用docker exec命令用于进入一个正在运行的docker容器。如果docker run命令运行容器的时候，没有使用-it参数，就要用这个命令进入容器。一旦进入了容器，就可以在容器的 Shell 执行命令了

```
docker exec -it f0b1c8ab3633 /bin/bash（有的容器需要把 /bin/bash 换成 sh）
```

11、容器内安装vim、ping、ifconfig等指令

```
apt-get update
apt-get install vim #安装vim
apt-get install iputils-ping #安装ping
apt-get install net-tools #安装ifconfig
```

12、删除容器

使用 docker rm命令即可删除指定容器

```
docker rm f0b1c8ab3633
```

该命令只能删除**已停止**的容器，如需删除正在运行的容器，可使用-f参数

强制删除所有容器

```
docker rm -f $(docker ps -a -q)
```

将微服务运行在docker上

使用Dockerfile构建Docker镜像

Dockerfile是一个文本文件，其中包含了若干条指令，指令描述了构建镜像的细节

先来编写一个最简单的Dockerfile，以前文下载的Nginx镜像为例，来编写一个Dockerfile修改该Nginx镜像的首页

1、新建一个空文件夹docker-demo，在里面再新建文件夹app，在app目录下新建一个名为Dockerfile的文件，在里面增加如下内容：

```
FROM nginx
RUN echo '<h1>This is Tuling Nginx!!!</h1>' > /usr/share/nginx/html/index.html
```

该Dockerfile非常简单，其中的 FROM、 RUN都是 Dockerfile的指令。 FROM指令用于指定基础镜像， RUN指令用于执行命令。

2、在Dockerfile所在路径执行以下命令构建镜像：

```
docker build -t nginx:tuling .
```

其中，-t指定镜像名字，命令最后的点（.）表示Dockerfile文件所在路径

3、执行以下命令，即可使用该镜像启动一个 Docker容器

```
docker run -d -p 92:80 nginx:tuling
```

4、访问 http://Docker宿主机IP:92/，可看到下图所示界面

This is Tuling Nginx!!!

Dockerfile常用指令

命令	用途
FROM	基础镜像文件
RUN	构建镜像阶段执行命令
ADD <src> <dest>	添加文件，从src目录复制文件到容器的dest，其中 src可以是Dockerfile所在目录的相对路径，也可以是一个 URL,还可以是一个压缩包
COPY	拷贝文件，和ADD命令类似，但不支持URL和压缩包
CMD	容器启动后执行命令
EXPOSE	声明容器在运行时对外提供的服务端口
WORKDIR	指定容器工作路径
ENV	指定环境变量
ENTRYPINT	容器入口， ENTRYPOINT和 CMD指令的目的一样，都是指定Docker容器启动时执行的命令，可多次设置，但只有最后一个有效。
USER	该指令用于设置启动镜像时的用户或者 UID,写在该指令后的 RUN、CMD以及 ENTRYPOINT指令都将使用该用户执行命令。
VOLUME	指定挂载点，该指令使容器中的一个目录具有持久化存储的功能，该目录可被容器本身使用，也可共享给其他容器。当容器中的应用有持久化数据的需求时可以在 Dockerfile中使用该指令。格式为：VOLUME["/data"]。

注意：RUN命令在 image 文件的构建阶段执行，执行结果都会打包进入 image 文件；CMD命令则是在容器启动后执行。另外，一个 Dockerfile 可以包含多个RUN命令，但是只能有一个CMD命令。

注意，指定了CMD命令以后，docker container run命令就不能附加命令了（比如前面的/bin/bash），否则它会覆盖CMD命令。

使用Dockerfile构建微服务镜像

以项目**05-ms-eureka-server**为例，将该微服务的可运行jar包构建成docker镜像

1、将jar包上传linux服务器/usr/local/docker-app/docker-demo/app/eureka目录，在jar包所在目录创建名为Dockerfile的文件

2、在Dockerfile中添加以下内容

```
# 基于哪个镜像
From java:8
# 复制文件到容器
ADD microservice-eureka-server-0.0.1-SNAPSHOT.jar /app.jar
# 声明需要暴露的端口
EXPOSE 8761
# 配置容器启动后执行的命令
ENTRYPOINT java ${JAVA_OPTS} -jar /app.jar
```

3、使用docker build命令构建镜像

```
docker build -t microservice-eureka-server:0.0.1 .
```

格式： docker build -t 镜像名称:标签 Dockerfile的相对位置

在这里，使用-t选项指定了镜像的标签。执行该命令后，终端将会输出如下的内容

```
[root@centos-new soft]# docker build -t microservice-eureka-server:0.0.1 .
Sending build context to Docker daemon  39.93MB
Step 1/5 : From java:8
 ---> d23bdf5b1b1b
Step 2/5 : VOLUME /tmp
 ---> Running in 47df11adc0e5
Removing intermediate container 47df11adc0e5
 ---> 0122276d643d
Step 3/5 : ADD microservice-eureka-server-0.0.1-SNAPSHOT.jar /app.jar
 ---> 14d35bbae674
Step 4/5 : EXPOSE 8761
 ---> Running in 3adb5133d715
Removing intermediate container 3adb5133d715
 ---> 0fda876dda27
Step 5/5 : ENTRYPOINT ["java","-jar","/app.jar"]
 ---> Running in 3d3ad98c0e2d
Removing intermediate container 3d3ad98c0e2d
 ---> 5ab4d23c9b69
Successfully built 5ab4d23c9b69
Successfully tagged microservice-eureka-server:0.0.1
```

4、启动镜像，加-d可在后台启动

```
docker run -d -p 8761:8761 microservice-eureka-server:0.0.1
```

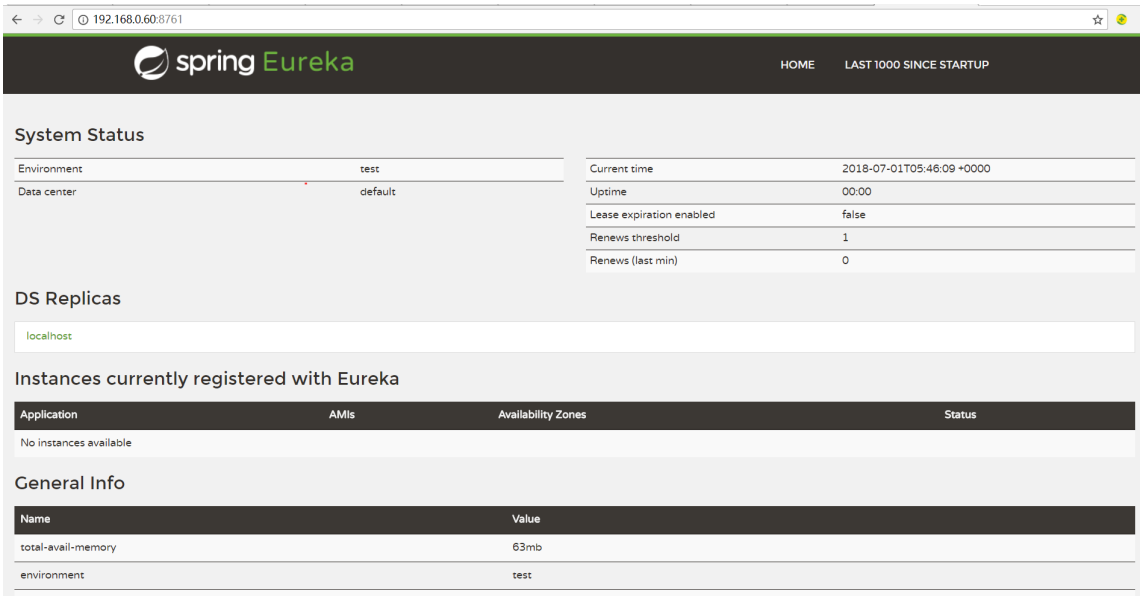
使用 -v 可以挂载一个主机上的目录到容器的目录

```
docker run -d -p 8761:8761 -v /log:/container-log microservice-eureka-server:0.0.1
```

加上JVM参数：

```
# --cap-add=SYS_PTRACE 这个参数是让docker能支持在容器里能执行jdk自带类似jinfo，jmap这些命令，如果不需要在容器里执行这些命令可以不加
docker run -e JAVA_OPTS='-Xms1028M -Xmx1028M -Xmn512M -Xss512K -XX:MetaspaceSize=256M -XX:MaxMetaspaceSize=256M' --cap-add=SYS_PTRACE -d -p 8761:8761 microservice-eureka-server:0.0.1
```

5、访问http://Docker宿主机IP:8761/，可正常显示Eureka Server首页



将微服务镜像发布到远程镜像仓库

我们制作好了微服务镜像，一般需要发布到镜像仓库供别人使用，我们可以选择自建镜像仓库，也可以直接使用docker官方镜像仓库，这里我们选择docker官方镜像仓库：https://hub.docker.com/

首先，我们需要在docke官方镜像仓库里注册一个账号

然后，在linux服务器上用docker login命令登录镜像仓库

```
[root@k8s-master ~]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one
Username: zhuge666
Password:
Login Succeeded
```

要把镜像推送到镜像仓库，需要将镜像前面加个分组名(一般就是docker hub的账户名)，执行如下命令修改镜像名字

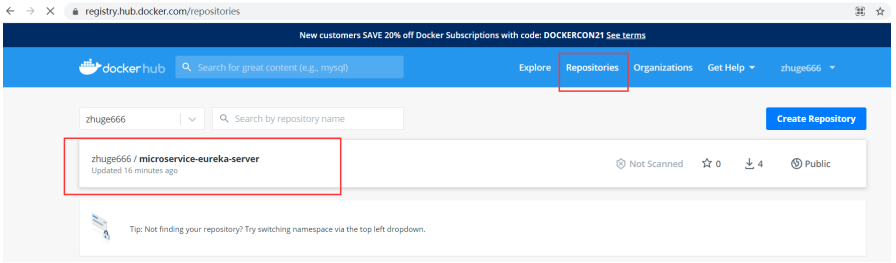
```
docker tag microservice-eureka-server:0.0.1 zhuge666/microservice-eureka-server:0.0.1
```

最后将镜像推送到远程仓库

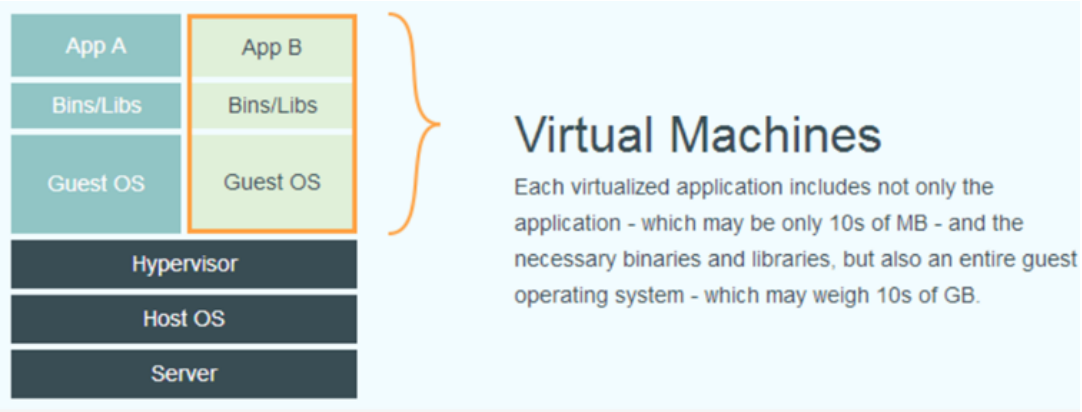
```
docker push zhuge666/microservice-eureka-server:0.0.1
```

```
[root@k8s-master ~]# docker push zhuge666/microservice-eureka-server:0.0.1
The push refers to repository [docker.io/zhuge666/microservice-eureka-server]
399d113acc87: Pushed
35c20f26d188: Pushed
c3fe59dd9556: Pushed
6ed1a81ba5b6: Mounted from pinpointdocker/pinpoint-hbase
a3483ce177ce: Mounted from pinpointdocker/pinpoint-hbase
ce6c8756685b: Pushed
30339f20ced0: Pushed
0eb22bfb707d: Pushed
a2ae92ffcd29: Pushed
0.0.1: digest: sha256:091f8b6a231596234d784cdba7ce6f78eebf5ced9c44e2aba6dcfd5b111532bc size: 2212
```

我们登录到docker镜像查看下刚刚推送的镜像，这样镜像就能给别人用了



Docker虚拟化原理



传统虚拟化和容器技术结构比较：传统虚拟化技术是在硬件层面实现虚拟化，增加了系统调用链路的环节，有性能损耗；容器虚拟化技术以共享宿主机Kernel的方式实现，几乎没有性能损耗。

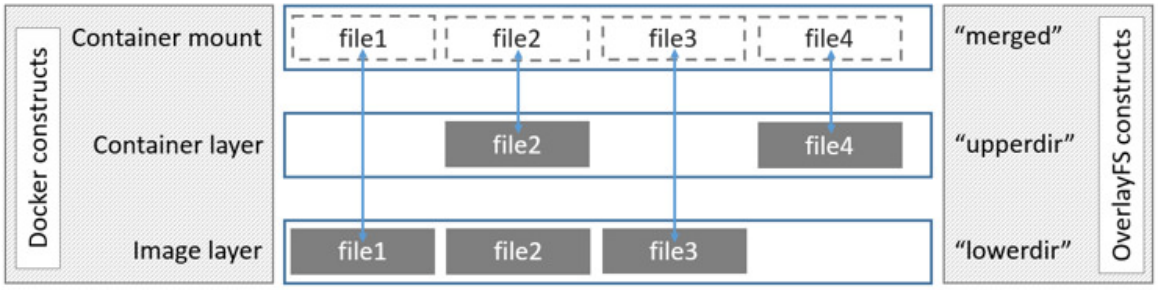
docker利用的是宿主机的内核,而不需要Guest OS。因此,当新建一个容器时,docker不需要和虚拟机一样重新加载一个操作系统内核。避免了寻址、加载操作系统内核这些比较费时费资源的过程,当新建一个虚拟机时,虚拟机软件需要加载Guest OS,这个新建过程是分钟级别的。而docker由于直接利用宿主机的操作系统,则省略了这个过程,因此新建一个docker容器只需要几秒钟。

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

Docker是如何将机器的资源进行隔离的?

答案是**联合文件系统**，常见的有AUFS、Overlay、devicemapper、BTRFS和ZFS等。

以Overlay2举例说明，Overlay2的架构图如下：



原理：overlayfs在linux主机上只有两层，一个目录在下层，用来保存镜像(docker)，另外一个目录在上层，用来存储容器信息。在overlayfs中，底层的目录叫做lowerdir，顶层的目录称之为upperdir，对外提供统一的文件系统为merged。当需要修改一个文件时，使用**COW(Copy-on-write)**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。

写时复制 (CoW) 技术详解

所有驱动都用到的技术—写时复制，Cow全称copy-on-write，表示只是在需要写时才去复制，这个是**针对已有文件的修改场景。**比如基于一个image启动多个Container，如果每个Container都去分配一个image一样的文件系统，那么将会占用大量的磁盘空间。而CoW技术可以让所有的容器共享image的文件系统，所有数据都从image中读取，只有当要对文件进行写操作时，才从image里把要写的文件复制到自己的文件系统进行修改。所以无论有多少个容器共享一个image，所做的写操作都是对从image中复制到自己的文件系统的副本上进行，并不会修改image的源文件，且多个容器操作同一个文件，会在每个容器的文件系统里生成一个副本，每个容器修改的都是自己的副本，互相隔离，互不影响。使用CoW可以有效的提高磁盘的利用率。**所以容器占用的空间是很少的。**

查看容器占用磁盘大小指令：

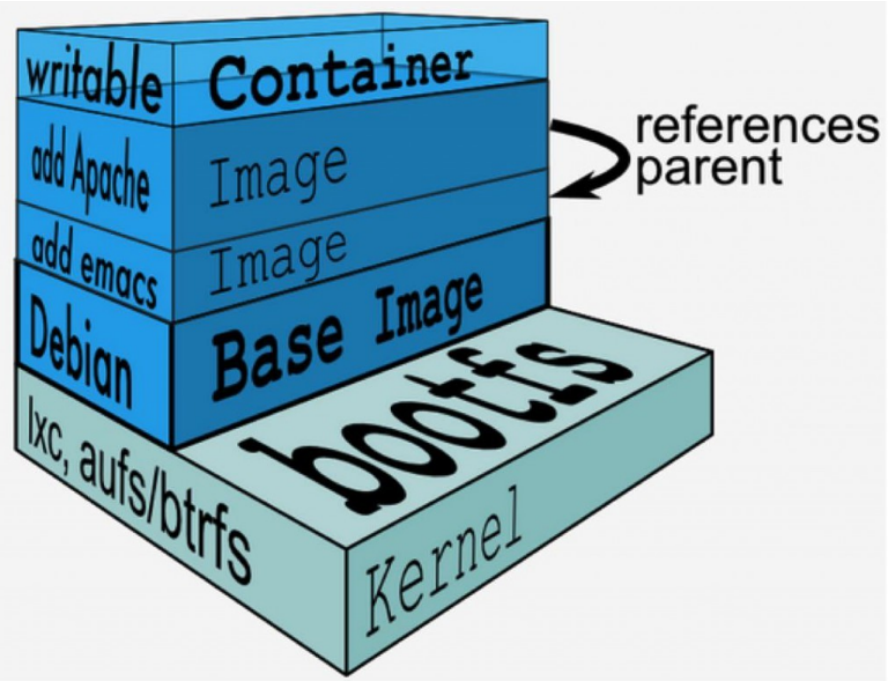
```
# 查看所有容器的大小
cd /var/lib/docker/containers # 进入docker容器存储目录
du -sh * # 查看所有容器的大小
du -sh <容器完整id> #查看某一个容器的大小
```

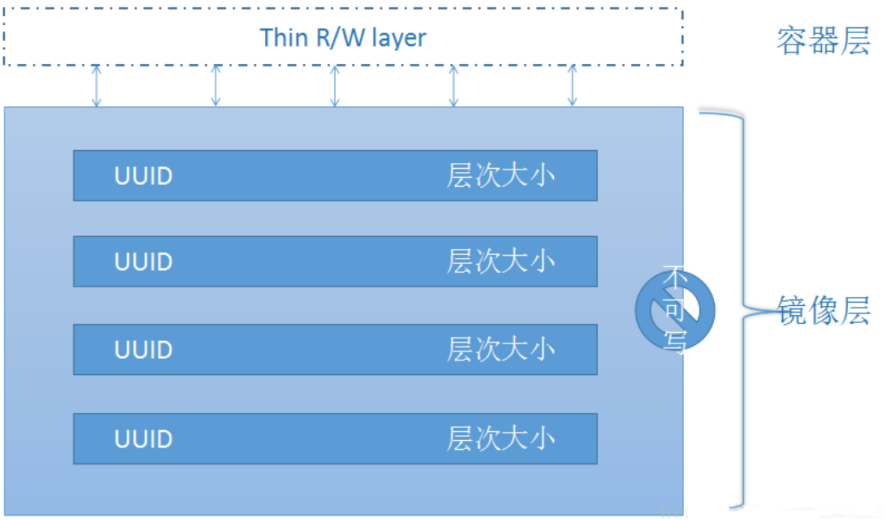
用时分配 （allocate-on-demand）

用时分配是**针对原本没有这个文件的场景**，只有在要新写入一个文件时才分配空间，这样可以提高存储资源的利用率。比如启动一个容器，并不会因为这个容器分配一些磁盘空间，而是当有新文件写入时，才按需分配新空间。

docker中的镜像分层技术的原理是什么呢?

docker使用共享技术减少镜像存储空间，所有镜像层和容器层都保存在宿主机的文件系统/var/lib/docker/中，由存储驱动进行管理，尽管存储方式不尽相同，但在所有版本的Docker中都可以**共享镜像层。**在下载镜像时，Docker Daemon会检查镜像中的镜像层与宿主机文件系统中的镜像层进行对比，如果存在则不下载，只下载不存在的镜像层，这样可以非常**节约存储空间。**





最后附一个查看容器资源使用情况的指令：

```
docker stats # 返回容器资源的实时使用情况，1秒刷新一次
docker stats --no-stream # 返回容器当时的资源使用情况
```

```
CONTAINER ID        NAME                      CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
7e51e49b6ce5        tulingmall-order          3.06%               301.6MiB / 3.692GiB   7.98%               4.38MB / 9.62MB     138MB / 65.5kB      102
60af8d7ab38f        tulingmall-gateway        5.21%               201.1MiB / 3.692GiB   5.32%               7.41MB / 10.7MB     110MB / 12.3kB      55
a939c76576c6        tulingmall-product        1.56%               272.1MiB / 3.692GiB   7.20%               1.41MB / 5MB        132MB / 0B          70
216e350e2044        tulingmall-member         0.89%               227MiB / 3.692GiB     6.00%               2.79MB / 6.22MB     125MB / 65.5kB      64
10ece81d8ff0        tulingmall-authcenter     0.61%               217MiB / 3.692GiB     5.74%               1.34MB / 4.9MB      119MB / 65.5kB      57
d01e5d20f70b        kibana                    0.97%               143.8MiB / 3.692GiB   3.80%               39MB / 30.6MB       402MB / 4.1kB       10
b526d760d239        logstash                  0.63%               312.6MiB / 3.692GiB   8.27%               5.11MB / 22.7MB     601MB / 156kB       38
154d5e427252        mongo                     1.10%               7.918MiB / 3.692GiB   0.21%               411kB / 544kB       80.4MB / 940kB      21
c797ce06af3c        docker_zookeeper_1        0.19%               43.39MiB / 3.692GiB   1.15%               229kB / 132kB       268MB / 124kB       31
10c890014e9d        rabbitmq                  0.18%               30.73MiB / 3.692GiB   0.81%               393kB / 781kB       284MB / 933kB       86
ebd41ca8027e        redis                     0.16%               644KiB / 3.692GiB     0.02%               135kB / 71.8kB      23.4MB / 624kB      4
ad43b2db50b5        elasticsearch             4.74%               318MiB / 3.692GiB     8.41%               53MB / 41.1MB       1.04GB / 181MB      48
c97460ddd888        nginx                     0.00%               1.344MiB / 3.692GiB   0.04%               18.8kB / 5.38kB     8.47MB / 12.3kB     2
2002524c3ce7        nacos                     1.29%               280.5MiB / 3.692GiB   7.42%               49.5MB / 25.9MB     650MB / 8.58MB      83
e18d5ebe5e3f        mysql                     0.06%               17.39MiB / 3.692GiB   0.46%               455kB / 559kB       299MB / 29.7MB      47
```

默认情况下，stats 命令会每隔 1 秒钟刷新一次输出的内容直到你按下 ctrl + c。下面是输出的主要内容：

[CONTAINER]：以短格式显示容器的 ID。

[CPU %]：CPU 的使用情况。

[MEM USAGE / LIMIT]：当前使用的内存和最大可以使用的内存。

[MEM %]：以百分比的形式显示内存使用情况。

[NET I/O]：网络 I/O 数据。

[BLOCK I/O]：磁盘 I/O 数据。

[PIDS]：PID 号。

注意：容器的内存使用最大限制默认可以接近宿主机的物理内存，可以通过"-m"参数限制容器可以使用的最大内存：

```
docker run -m 500M redis #限制容器的最大使用内存为500M
```

文档：Docker详解与部署微服务实战

```
链接：http://note.youdao.com/noteshare?id=42384826563c36cddfa032983505bd0d&sub=C4C689D20A69487F90A3E7CA78592257
```