史上最详细的docker学习手册，请查收！

IT服务圈儿 2021-07-05 17:30

以下文章来源于JAVA日知录 ， 作者飘渺Jam

作者丨飘渺Jam
来源丨JAVA日知录（ID：javadaily）

一、docker入门

1、docker的安装及入门示例

- 环境准备：docker需要安装在centos7 64位系统上；docker要求系统内核在3.10以上
- 查看系统内核：

```
uname -r
```

- 安装命令：

```
yum -y install docker-io
```

- 安装完成后，启动命令

```
service docker start
```

- 安装nginx体验

```
```

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

2、doker的理论概念

- 什么是docker:

  鲸鱼通过身上的集装箱（Container）来将不同种类的货物进行隔离；而不是通过生出很多小鲸鱼（Guest OS）来承运不同种类的货物。Docker 是一个开源的应用容器引擎，基于 Go 语言 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。

  物理机、虚拟机、docker的形象对比

  这是物理机：一栋楼一户人家、独立地基、独立花园。

  这是虚拟机：一栋楼包含多套房子，一套房一户人家，共享地基、共享花园，独立卫生间、厨房、宽带

这是docker容器：一套房隔离成多个小空间俗称胶囊公寓，每个胶囊住一个租客，共享地基、花园、卫生间、厨房、宽带等等

- docker和虚拟机的对比

	docker	虚拟机
运行环境	docker构建在操作系统上，所以docker甚至可以在虚拟机上运行	虚拟化技术依赖物理CPU和内存，是硬件级别的
隔离性	docker属于进程之间的隔离	虚拟机可实现系统级别隔离
安全性	Docker的租户root和宿主机root等同，一旦容器内的用户从普通用户权限提升为root权限，它就直接具备了宿主机的root权限，进而可进行无限制的操作。　故：docker的安全性也更弱	虚拟机租户root权限和宿主机的root虚拟机权限是分离的，并且虚拟机利用硬件隔离：如Intel的VT-d和VT-x的ring-1硬件隔离技术，这种隔离技术可以防止虚拟机突破和彼此交互　虚拟机的安全性高
可管理性	docker的集中化管理工具还不算成熟。	各种虚拟化技术都有成熟的管理工具，例如VMware vCenter提供完备的虚拟机管理能力。
快速创建、删除	Docker容器创建是秒级别的，Docker的快速迭代性，决定了无论是开发、测试、部署都可以节约大量时间。	虚拟化创建是分钟级别的
交付、部署	Docker在Dockerfile中记录了容器构建过程，可在集群中实现快速分发和快速部署;	虚拟机可以通过镜像实现环境交付的一致性，但镜像分发无法体系化；

- docker的架构原理

derver daemon：docker server是一个守护进程，它可以部署远程也可以部署本地。它包含2部分网络newtwork和磁盘data volumes

REST API： 实现了client和server间的通信交互协议

CLI（command line interface）: docker client ,它包含2部分容器和镜像，1个镜像可以创建N个容器(container)

Image: 一个只读的镜像模板。可以自己创建一个镜像也可以从网站上下载镜像供自己使用。

Container:由docker client通过镜像创建的实例，用户在容器中运行应用，一旦创建后就可以看做是一个简单的操作系统，每个应用运行在隔离的容器中，享用独自的权限，用户，网络。

Registry:镜像仓库，用来存储和管理image镜像，目前主流的仓库有Docker hub、阿里云镜像仓库，也可以自己创建仓库来管理。

说明：docker client 通过3个命令，先到docker daemon pull拉取images,如果服务端没有，先到仓库拉取，(如果仓库没有你可以build自建images）；最后通过run命令创建容器

二、镜像管理

1、什么是镜像

- 一个只读的模板，就是一个dockerfile,可以在镜像仓库上传或下载
- 先有镜像，后有容器；并且一个镜像可以创建多个容器。
- docker镜像最大的特点：分层结构。
  最底层是一个base层，也即是一个操作系统层。它还会从base层一层层的叠加生成（什么是层层叠加？例如，安装一个软件，它就会在base层的基础上追加一层。它的好处就是资源共享）。

2、镜像相关命令

- 查看docker本地的镜像

```
docker images
```

  搜索一个镜像

```
docker search mysql
```

```

```

  INDEX ： 仓库地址
  NAME ： 仓库+名称
  STARS :用户的喜欢程度
  OFFICIAL：是否为官方，如果为OK的话代表官方，可信度高，放心使用 UTOMATED：是否为公开的dockerfile脚本制成的？也即是说dockerfile是否提供。

- 下载一个镜像

```

```

```
docker pull mysql:5.7
```

下载完后，查看是否在本地 docker images

- 删除docker镜像

```
docker rmi [imageID]
# 如果要删除全部镜像
docker rmi $(docker images -q)
```

- 加速器的配置

```
#centos的阿里云加速器命令
mkdir -p /etc/docker
tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://uqxmqcrw.mirror.aliyuncs.com"]
}
EOF
systemctl daemon-reload
systemctl restart docker
```

3、dockerfile

- dockerfile概念

dockerfile是一个文本的配置文件，它可以快速创建自定义的镜像，文本内容包含了若干的命令行，并支持#作为注释行，文本格式包含基础镜像FROM，维护者MAINTAINER,操作指令ADD,容器启动后指令等共计4部分

- dockerfile文件示例

```
#1.基础镜像：FROM指令：基础镜像名：tag,例如java:8
FROM java:8

#2.维护者：格式：MAINTAINER <name>
MAINTAINER jackly
```

```
#3.镜像的操作指令
# ADD拷贝一个文件到容器中，格式：ADD <src> <dest>
ADD eureka-server-0.0.1-SNAPSHOT.jar  /app/service/eureka/data/app.jar

#告诉docker容器暴露端口，在容器启动的时候，需要通过-p 做端口映射
EXPOSE 8761

#5.配置容器启动后，执行什么命令
ENTRYPOINT ["java","-jar","/app/service/eureka/data/app.jar"]
```

- 构建docker镜像

```
#其中 -t 对镜像进行命名，一般的命名法：仓库名字/镜像名字：版本号
#注意：其中 .号，代表当前目录下的dockerfile文件
docker build -t registry-jackly/eureka-server:1.0.0 .
```

- 查看和运行镜像容器

```
#查看本地镜像
docker images
#启动镜像
docker run -d -p 8761:8761 --name=eureka registry-jackly/eureka-server:1.0.0
```

三、镜像仓库管理

1、建设dockerhub官方仓库

- 什么是镜像仓库

  就是存放镜像的地方

- 推送镜像到dockerhub(需翻墙)

```
①、登录 https://hub.docker.com/，创建一个eureka-server仓库
②、本地linux登录docker官方， docker login
③、改镜像的名，（为什么要改？因为docker官方镜像仓库是以 用户名 来命名仓库的）
   docker tag registry-jackly/eureka-server:1.0.0 jacklydocker/eureka-server:1.0.0
```

```
④、推送镜像到官方仓库
   docker push jacklydocker/eureka-server:1.0.0
⑤、拉取上传的镜像
   先删除旧镜像:docker rmi jacklydocker/eureka-server:1.0.0
   拉取镜像：docker pull jacklydocker/eureka-server:1.0.0
   创建容器：docker run -d -p 8761:8761 --name=eureka docker.io/jacklydocker/eureka
```

2、建设阿里云docker仓库

- 推送镜像到阿里云仓库

```
①、登录阿里云，先创建命名空间，并建个镜像仓库
②、本地linux登录登录阿里云Docker Registry， docker login --username=你的用户名 regist
③、改镜像的名，（为什么要改？因为docker官方镜像仓库是以 用户名 来命名仓库的）
   docker tag 58acc264425c registry.cn-shenzhen.aliyuncs.com/jackly/eureka-server:
④、推送镜像到阿里云仓库
   docker push registry.cn-shenzhen.aliyuncs.com/jackly/eureka-server:1.0.0
⑤、拉取上传的镜像
   docker pull registry.cn-shenzhen.aliyuncs.com/jackly/eureka-server:1.0.0
```

3、建设本地仓库

- 为什么需要搭建本地仓库

  (1).节约带宽：因为如果用docker或阿里云官方的仓库走的互联网浪费带宽，而且慢。
  (2).提供资源利用和安全：因为公司内部的镜像，推送到本地仓库，更方便公司内部人员用，而且安全性高。

- 创建仓库容器

```
#查找官方仓库
docker search registry
#拉取仓库镜像
docker pull registry
#运行仓库容器
docker run -d -p 5000:5000 \
--restart=always \
```

```
--privileged=true \
--name=registry-local-jackly \
-v /date/volume/registry:/var/lib/registry \
registry
```

- 推动镜像到本地私有仓库

```
①、改镜像的名
   docker tag 58acc264425c reg.qxbdocker.com:5000/eureka-server:1.0.0
②、推送镜像到私有仓库(注意：记得改本地hosts：127.0.0.1 reg.qxbdocker.com)
   docker push reg.qxbdocker.com:5000/eureka-server:1.0.0
③、查看搭建仓库的信息
   查看仓库的镜像：
    curl -X GET http://127.0.0.1:5000/v2/_catalog
   仓库仓库某个镜像的版本信息：
    curl -X GET http://127.0.0.1:5000/v2/eureka-server/tags/list
⑤、拉取上传的镜像
 #先删除本地镜像
 docker rmi reg.qxbdocker.com:5000/eureka-server:1.0.0
 #在拉取仓库中的镜像
    docker pull reg.qxbdocker.com:5000/eureka-server:1.0.0
```

四、容器管理

1、容器的生命周期实践

- 什么是容器

  容器类似于胶囊公寓，它是一个精简版的操作系统，一般容器中只运行一个应用（例如：eureka-server镜像）。

- 如何创建容器

  通过镜像创建容器，通过docker run命令创建。

- 容器的作用

  容器起到了隔离的作用，独享空间、网络等等。

- doker命令

```
docker run --help
 -d, --detach=false          指定容器运行于前台还是后台，默认为false
 -i, --interactive=false     打开STDIN，用于控制台交互
 -t, --tty=false             分配tty设备，该可以支持终端登录，默认为false
 -u, --user=""               指定容器的用户
 -a, --attach=[]             登录容器（必须是以docker run -d启动的容器）
 -w, --workdir=""            指定容器的工作目录
 -c, --cpu-shares=0          设置容器CPU权重，在CPU共享场景使用
 -e, --env=[]                指定环境变量，容器中可以使用该环境变量
 -m, --memory=""             指定容器的内存上限
 -P, --publish-all=false     指定容器暴露的端口
 -p, --publish=[]            指定容器暴露的端口
 -h, --hostname=""           指定容器的主机名
 -v, --volume=[]             给容器挂载存储卷，挂载到容器的某个目录
 --volumes-from=[]           给容器挂载其他容器上的卷，挂载到容器的某个目录
 --cap-add=[]                添加权限，权限清单详见：http://linux.die.net/man/7/capab
 --cap-drop=[]               删除权限，权限清单详见：http://linux.die.net/man/7/capab
 --cidfile=""                运行容器后，在指定文件中写入容器PID值，一种典型的监控系统用
 --cpuset=""                 设置容器可以使用哪些CPU，此参数可以用来容器独占CPU
 --device=[]                 添加主机设备给容器，相当于设备直通
 --dns=[]                    指定容器的dns服务器
 --dns-search=[]             指定容器的dns搜索域名，写入到容器的/etc/resolv.conf文件
 --entrypoint=""             覆盖image的入口点
 --env-file=[]               指定环境变量文件，文件格式为每行一个环境变量
 --expose=[]                 指定容器暴露的端口，即修改镜像的暴露端口
 --link=[]                   指定容器间的关联，使用其他容器的IP、env等信息
 --lxc-conf=[]               指定容器的配置文件，只有在指定--exec-driver=lxc时使用
 --name=""                   指定容器名字，后续可以通过名字进行容器管理，links特性需要使
 --net="bridge"              容器网络设置:
                                bridge 使用docker daemon指定的网桥
                                host    //容器使用主机的网络
                                container:NAME_or_ID  >//使用其他容器的网路，共享IP和P
                                none 容器使用自己的网络（类似--net=bridge），但是不进行
 --privileged=false          指定容器是否为特权容器，特权容器拥有所有的capabilities
 --restart="no"              指定容器停止后的重启策略:
                                no：容器退出时不重启
                                on-failure：容器故障退出（返回值非零）时重启
                                always：容器退出时总是重启
```

```
--rm=false                    指定容器停止后自动删除容器(不支持以docker run -d启动的容器
--sig-proxy=true              设置由代理接受并处理信号，但是SIGCHLD、SIGSTOP和SIGKILL不
```

- docker创建示例

```
docker run -d -p 5000:5000 \
 --restart=always \
 --privileged=true \
 --name=registry-local-jackly \
 -v /date/volume/registry:/var/lib/registry \
 registry
```

- docker常用命令

```
#查看正在运行的容器
docker ps
#查看所有的容器(包括已经停止的容器)
docker ps -a
#停止容器
docker stop 容器ID
#强制停止容器
docker kill 容器ID
#启动容器
docker start 容器ID
#重启容器
docker restart 容器ID
#删除已经停止的容器
docker rm 容器ID
#删除正在运行的容器，强制删除
docker rm -f 容器ID
```

2、如何进入容器内部

- 进入容器的方式

  有4种方式能进入容器，分别为 exec、docker attach、ssh、nsenter. 这4种都能进入容器，但是最好用最常用的是exec

- exec命令

```
docker exec --help
 -d 以后台方式执行，这样，我们执行完这条命令，还可以干其他事情，写脚本最常用
 -e 代表环境变量
 -i 以交互方式运行，是阻塞式的
 -t 分配一个伪终端，这个参数通常与-i参数一起使用，然后在后面跟上容器里的/bin/bash，这样就
 -u 指定进入的容器以哪个用户登陆，默认是root
```

- 进入容器命令

```
#启动镜像，若镜像容器已经创建，则通过docker ps -a查询停止的容器id，在通过docker start
docker run -d -p 8761:8761 --name=eureka registry-jackly/eureka-server:1.0.0
#进入容器的相关命令
docker exec -it eureka sh
#进入后可以使用ls查看目录（/app/service/eureka/data/app.jar）
docker exec -it eureka /bin/bash
docker exec -it eureka pwd
docker exec -it eureka top
```

3、容器内容改变后，能否重新生成镜像

- 使用ll命令验证

```
①、进入容器：docker exec -it eureka /bin/bash
②、修改容器内容：echo "alias ll='ls -l'" >> ~/.bashrc && source ~/.bashrc
③、验证ll命令:ll
④、强制删除容器：docker rm -f 容器ID
⑤、启动镜像容器：docker run -d -p 8761:8761 --name=eureka registry-jackly/eureka-ser
⑥、进入容器使用ll命令：发现先前修改的实效
```

- 修改容器后，重新生成镜像

```
①、进入容器：docker exec -it eureka /bin/bash
②、修改容器内容：echo "alias ll='ls -l'" >> ~/.bashrc && source ~/.bashrc
③、验证ll命令:ll
#生成镜像命令
docker commit --help
 -a 用来指定作者
 -c 使用Dockerfile指令来创建镜像
 -m 描述我们此次创建image的信息
 -p 在commit时，将容器暂停
④、重新生成镜像
docker commit -m="add ll" --author="jackly" eureka registry-jackly/eureka-server:2
⑤、启动新镜像
docker run -d -p 8761:8761 --name=eureka registry-jackly/eureka-server:2.0.0
```

五、网络管理

1、外部网络如何访问容器应用

- 外部访问容器应用，是通过端口来实现的

```
#参数-p，指定端口，其中 8080是宿主机的端口，80是容器的端口
docker run -d -p 8080:80 nginx
#参数-P（大写），随机端口，随机范围 32769-60999
docker run -d -P nginx
```

- dockerfile的端口实现规则

（1）有开放 EXPOSE 8761

```
#参数-p，指定端口
docker run -d -p 8761:8761 --name=eureka registry-agan/eureka-server:1.0.0
#参数-P，随机端口（ 达到的效果是0.0.0.0:32771->8761）
docker run -d -P --name=eureka registry-jackly/eureka-server:2.0.0
```

（2）没有开放 EXPOSE 8761

```
#参数-p，指定端口(效果：正常)
docker run -d -p 8761:8761 --name=eureka registry-jackly/eureka-server:2.0.0
#参数-P，随机端口（ 达到的效果是，无端口号，连内部容器都没有端口）
docker run -d -P --name=eureka registry-jackly/eureka-server:2.0.0
```

2、如何实现容器之间的网络通信

- 业务场景为4个容器

  mysql、eureka、product、config

- 安装mysql

```
#启动mysql容器
docker run -p 3306:3306 --name mysql \
 -e MYSQL_ROOT_PASSWORD=agan \
 -d mysql:5.7
#进入mysql容器
docker exec -it mysql /bin/bash
#docker镜像没有ifconfig、ping指令
apt-get update
apt install net-tools # ifconfig
apt install iputils-ping # ping
```

- 创建eureka容器

```
docker run -d -p 8761:8761 --name=eureka registry-jackly/eureka-server:2.0.0
```

- 部署config镜像和容器（加入config有问题，暂时未找到原因）

  dockerfile:

```
#1.基础镜像：FROM指令：基础镜像名：tag,例如java:8
FROM java:8

#2.维护者：格式：MAINTAINER <name>
MAINTAINER jackly
```

```
#3.镜像的操作指令
# ADD拷贝一个文件到容器中，格式：ADD <src> <dest>
ADD config-server-0.0.1-SNAPSHOT.jar  /app/service/config/data/app.jar

#5.配置容器启动后，执行什么命令
ENTRYPOINT ["java","-jar","/app/service/config/data/app.jar"]
```

构建镜像

```
#其中 -t 对镜像进行命名，一般的命名法：仓库名字/镜像名字：版本号
#注意：其中 .号，代表当前目录下的dockerfile文件
docker build -t registry-jackly/config-server:1.0.0 .
```

创建容器

```
#查看本地镜像
docker images
#启动镜像 link eureka:jacklyureka==>link 容器名称：别名（将配置中心服务注册到eureka）
docker run -d -p 9030:9030 --name config \
 --link eureka:jacklyureka \
 registry-jackly/config-server:1.0.0
#查看config信息
http://172.31.65.26:9030/e-book-product/deauflt
```

进入config容器

查看eureka注册信息

- 部署product镜像和容器

  dockerfile:

```
#1.基础镜像：FROM指令：基础镜像名：tag,例如java:8
FROM java:8

#2.维护者：格式：MAINTAINER <name>
MAINTAINER jackly

#3.镜像的操作指令
# ADD拷贝一个文件到容器中，格式：ADD <src> <dest>
ADD e-book-product-core-0.0.1-SNAPSHOT.jar  /app/service/product/data/app.jar

#5.配置容器启动后，执行什么命令
ENTRYPOINT ["java","-jar","/app/service/product/data/app.jar"]
```

  构建镜像

```
#其中 -t 对镜像进行命名，一般的命名法：仓库名字/镜像名字：版本号
#注意：其中 .号，代表当前目录下的dockerfile文件
docker build -t registry-jackly/product-server:1.0.0 .
```

  创建容器

```
#查看本地镜像
docker images
#启动镜像 注：link就是容器直接的连接，你不用IP的情况下可以通过link来实现容器名之间的通信；
```

```
docker run -d -p 8083:8083 --name product \
 --link mysql:jacklymysql \
 --link eureka:jacklyeureka \
 registry-jackly/product-server:1.0.0
#验证效果
http://172.31.65.26:8761/
http://172.31.65.26:8083/product/findAllProduct
```

link原理

```
#原理就是在prodct容器中的hosts加了2条记录。
docker exec -it product /bin/bash
cat /etc/hosts
```

六、数据管理

1、docker容器的数据如何共享给宿主机

- 宿主机查看eureka日志

1）使用docker run volume方式实现

```
#构建镜像
docker build -t registry-jackly/eureka-server:2.0.0 .
#创建容器
#就是把docker的数据保存到宿主机的磁盘中，通常说的就是挂载点，或者叫做卷。
#语法：  -v 宿主机目录：容器目录
docker run -d -p 8761:8761 --name=eureka \
  --privileged=true \
  -v /app/service/eureka/logs:/opt/data \
  registry-jackly/eureka-server:2.0.0
```

2）使用dokcerfile方式实现

```
#1.基础镜像：FROM指令：基础镜像名：tag,例如java:8
FROM java:8
```

```
#2.维护者：格式：MAINTAINER <name>
MAINTAINER jackly

#3.加入挂载点
VOLUME /opt/data

#4.镜像的操作指令
# ADD拷贝一个文件到容器中，格式：ADD <src> <dest>
ADD eureka-server-0.0.1-SNAPSHOT.jar  /app/service/eureka/data/app.jar

#5.告诉docker容器暴露端口，在容器启动的时候，需要通过-p 做端口映射
EXPOSE 8761

#6.配置容器启动后，执行什么命令
ENTRYPOINT ["java","-jar","/app/service/eureka/data/app.jar"]
```

```
#重新构建eureka
docker build -t registry-jackly/eureka-server:3.0.0 .
#创建容器
docker run -d -p 8761:8761 --name=eureka \
  --privileged=true \
  registry-jackly/eureka-server:3.0.0
```

注意点：dockerfile volume 无法指定宿主机的目录，都是自动生成，而且是随机的；默认在/var/lib/docker/volumes/。（为什么是随机生成？因为dockerfile无法确定每台宿主机是否都存在目录）

```
#如何找到宿主机的挂载目录？
  docker inspect eureka
  #信息段
  "Mounts": [
            {
                "Type": "volume",
                "Name": "cf527694ebafb92426a52f1916b26832b4c8977093083450a96fbccb3
                "Source": "/var/lib/docker/volumes/cf527694ebafb92426a52f1916b2683
                "Destination": "/opt/data",
                "Driver": "local",
```

```
            "Mode": "",
            "RW": true,
            "Propagation": ""
        }
```

- 总结

  docker run 是能指定宿主机的目录。dockerfile volume 无法指定宿主机的目录，都是自动生成，而且是随机的；默认在/var/lib/docker/volumes/。

2、宿主机如何直接维护docker容器的数据

在没有使用-v挂载点时，创建的容器，在容器删除后，根据镜像重新生成容器后，数据也随之流失。如果使用了挂载点，删除容器后，在根据镜像生成容器，数据还会保留。

```
docker run -p 3306:3306 --name mysql \
-e MYSQL_ROOT_PASSWORD=agan \
--privileged=true \
-v /app/data/mysql:/var/lib/mysql \
-d mysql:5.7
```

```
[root@iflydev-no mysql]# ll
total 188476
-rw-r----- 1 polkitd input       56 May 26 14:37 auto.cnf
-rw------- 1 polkitd input     1680 May 26 14:37 ca-key.pem
-rw-r--r-- 1 polkitd input     1112 May 26 14:37 ca.pem
-rw-r--r-- 1 polkitd input     1112 May 26 14:37 client-cert.pem
-rw------- 1 polkitd input     1676 May 26 14:37 client-key.pem
-rw-r----- 1 polkitd input     1359 May 26 14:37 ib_buffer_pool
-rw-r----- 1 polkitd input 79691776 May 26 14:37 ibdata1
-rw-r----- 1 polkitd input 50331648 May 26 14:37 ib_logfile0
-rw-r----- 1 polkitd input 50331648 May 26 14:37 ib_logfile1
-rw-r----- 1 polkitd input 12582912 May 26 14:37 ibtmp1
drwxr-x--- 2 polkitd input     4096 May 26 14:37 mysql
drwxr-x--- 2 polkitd input     4096 May 26 14:37 performance_schema
-rw------- 1 polkitd input     1676 May 26 14:37 private_key.pem
-rw-r--r-- 1 polkitd input      452 May 26 14:37 public_key.pem
-rw-r--r-- 1 polkitd input     1112 May 26 14:37 server-cert.pem
-rw------- 1 polkitd input     1676 May 26 14:37 server-key.pem
drwxr-x--- 2 polkitd input    12288 May 26 14:37 sys
[root@iflydev-no mysql]#
```

七、镜像仓库管理系统搭建

搭建一个镜像仓库管理系统需要3个步骤，分别是：生成一个认证文件，rsa的认证文件；创建一个仓库容器；创建一个仓库web管理系统

- 生成一个认证文件，rsa的认证文件

```
#建立/app/registry-jackly/conf，在/app/registry-jackly目录下执行 以下命令：
openssl req -new -newkey rsa:4096 -days 365 -subj "/CN=localhost"  -nodes -x509 -k
```

- 创建一个仓库容器

```
#创建配置文件/app/registry-jackly/conf/registry-jackly.yml

version: 0.1
#镜像存储地方
storage:
  filesystem:
    rootdirectory: /var/lib/registry
  #镜像的删除权限，enabled: true代表开启删除权限
  delete:
    enabled: true
log:
  level: info
#开启仓库的网络，端口号为5000
http:
  addr: 0.0.0.0:5000
```

```
#创建仓库命令
docker run \
-v /app/registry-jackly/conf/registry-jackly.yml:/etc/docker/registry/config.yml:r
-v /app/registry-jackly/conf/auth.cert:/etc/docker/registry/auth.cert:ro \
-p 5000:5000  --name registry-docker -d \
--privileged=true \
registry
```

- 创建一个仓库web管理系统

```
#创建一个配置文件：/app/registry-jackly/conf/registry-web.yml
```

```
registry:
  # 指定registry的地址（注意：registry-docker为仓库的容器名字）
  url: http://registry-docker:5000/v2
  #仓库的名称（注意：registry-docker为仓库的容器名字）
  name: registry-docker:5000
  #是否为只读模式，设置true时，不允许删除镜像
  readonly: false
  #权限验证
  auth:
    #是否开启验证
    enabled: true
    #验证证书的key
    key: /conf/auth.key
    #证书颁发者的名称
    issuer: docker
```

```
#创建仓库web管理系统命令(注意：--link registry-docker很重要，没有的话，无法连接仓库。)
docker run \
-v /app/registry-jackly/conf/registry-web.yml:/conf/config.yml:ro \
-v /app/registry-jackly/conf/auth.key:/conf/auth.key \
-v /app/registry-jackly/db:/data \
-d -p 8080:8080 --link registry-docker --name registry-web \
--privileged=true \
hyper/docker-registry-web
```

- 效果验证

```
#登录仓库管理系统
http://172.31.65.26:8080/login/auth
用户名=admin
密码 =admin

#构建镜像
#其中 -t 对镜像进行命名，一般的命名法：仓库名字/镜像名字：版本号
#注意：其中 .号，代表当前目录下的dockerfile文件
docker build -t registry-docker:5000/eureka-server:3.0.0 .
```

```
#修改hosts(vi /etc/hosts)
127.0.0.1 registry-docker

#给搭建好的仓库加个镜像
docker push registry-docker:5000/eureka-server:3.0.0

#权限设置
默认admin用户是没有删除权限，需要重新创建用户，并且给予权限。
```

八、maven构建springcloud镜像

在先前构建项目时，先通过maven打包出jar,在手动上传到虚拟机上，并编写dockerfile文件，在使用docker build命令构建镜像，比较繁琐。实际上可以通过maven来直接构建springcloud镜像，maven构建springcloud镜像推送给仓库，需要2步骤：开启docker远程API；编写maven的docker插件。

- 开启docker远程API

```
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket containerd.service

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock \
                           -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always

# Note that StartLimit* options were moved from "Service" to "Unit" in systemd 229.
# Both the old, and new location are accepted by systemd 229 and up, so using the old location
# to make them work for either version of systemd.
StartLimitBurst=3

# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd 230.
# Both the old, and new name are accepted by systemd 230 and up, so using the old name to make
# this option work for either version of systemd.
StartLimitInterval=60s

# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity

# Comment TasksMax if your systemd version does not support it.
# Only systemd 226 and above support this option.
TasksMax=infinity

# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes
-- INSERT --
```

```
#在配置文件中，加入：-H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock
```

```
vi /usr/lib/systemd/system/docker.service
#重启docker
systemctl daemon-reload
systemctl restart docker
#验证docker远程api是否生效
netstat -anp|grep 2375
curl 127.0.0.1:2375/info
```

- 编写maven的docker插件

```
#本地配置hosts(c:\windows\system32\drivers\etc)
172.31.65.26 registry-docker

#docker maven插件
<build>
  <plugins>
   <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
   </plugin>
   <!-- 添加docker maven插件 -->
   <plugin>
    <groupId>com.spotify</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>1.1.1</version>
    <configuration>
     <!-- 推送到指定的仓库 -->
     <registryUrl>registry-docker:5000</registryUrl>
     <!-- 开启docker远程API的端口 -->
     <dockerHost>http://registry-docker:2375</dockerHost>
     <!-- 指定镜像名称 格式：仓库域名：端口/镜像名字：镜像的版本号 -->
     <imageName>registry-docker:5000/${project.artifactId}:${project.version}</ima
     <!-- 指定基础镜像 类似dockerfile的FROM指令 -->
     <baseImage>java:8</baseImage>
     <!-- 配置容器启动后，执行什么命令，等于与 dockerfile的ENTRYPOINT -->
     <entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]</entryPoint>
     <!-- 为dockerde tag指定版本号、latest -->
     <imageTags>
      <imageTag>${project.version}</imageTag>
```

```
      <imageTag>latest</imageTag>
     </imageTags>
     <!-- 是否有push的功能 true代表有 -->
     <pushImage>true</pushImage>
     <!-- push后是否覆盖已存在的标签镜像 -->
     <forceTags>true</forceTags>
     <!-- 复制jar包到docker容器指定的目录 -->
     <resources>
      <resource>
       <targetPath>/</targetPath>
       <!-- 指定要复制jar包的根目录，${project.build.directory}代表 target的目录 -->
       <directory>${project.build.directory}</directory>
       <!-- 指定要复制的文件，${project.build.finalName}代表打包后的jar -->
       <include>${project.build.finalName}.jar</include>
      </resource>
     </resources>
    </configuration>
   </plugin>
  </plugins>
 </build>

#执行命令
clean package -DskipTests docker:build
```

验证效果

九、编排构建**springcloud**实例

- 什么是compose为什么要使用compose

  因为运行一个docker镜像，通常是需要docker run 命令，在运行镜像的时候还需要一定的参数；例如 容器的名称 映射的卷，绑定端口等等，非常麻烦。那如果有个一个文件来记录

保存这些命令该多好？所以compose就是用于存储这些命令，而且呢是比docker run还要简单存储。那compose是什么呢？它既是一个yaml格式的文件，例如docker-compose.yml文件。

```
#安装最新compose
curl -L https://get.daocloud.io/docker/compose/releases/download/1.21.2/docker-com

#添加可执行权限
chmod +x /usr/local/bin/docker-compose

#测试安装结果
docker-compose --version
```

- 自动构建spring cloud注册中心eureka

  创建一个网络

```
创建命令：docker network create dockernet
查看命令：docker network ls
```

  compose内容：

```
#docker compse 的配置文件包含3大部分：version services networks
version: '3'
services:
  # 服务名称
  eureka:
    # 容器名称
    container_name: eureka
    # 镜像名称
    image: registry-docker:5000/eureka-server:0.0.1-SNAPSHOT
    # 暴露的端口号
    ports:
      - "8761:8761"
    # 设置卷挂载的路径 /opt/data代表的是日志存储路径
    volumes:
```

```
      - /app/service/eureka/logs:/opt/data
    # 设置权限 ：拥有root权限
    privileged: true

    networks:
      - default

networks:
  default:
    external:
      name: dockernet
```

运行命令

```
docker-compose -f docker-compose-eureka.yml  up -d
```

演示效果

```
http://172.31.65.26:8761/
```

- 自动构建spring cloud 配置中心config

  compose内容

```
version: '3'
services:
  config-server:
    container_name: config-server
    image: registry-docker:5000/config-server:0.0.1-SNAPSHOT
    ports:
      - "9030:9030"
    networks:
      - default

networks:
  default:
```

```
    external:
      name: dockernet
```

运行命令

```
docker-compose -f docker-compose-config.yml  up -d
```

演示效果

```
http://172.31.65.26:9030/e-book-product/default
```

- 自动构建spring cloud 调用链zipkin

  compose内容

```
version: '3'
services:
  zipkin-server:
    container_name: zipkin-server
    image: registry-docker:5000/zipkin-server:0.0.1-SNAPSHOT
    ports:
      - "9411:9411"
    networks:
      - default

networks:
  default:
    external:
      name: dockernet
```

  运行命令

```
docker-compose -f docker-compose-zipkin.yml  up -d
```

  演示效果

```
http://172.31.65.26:9411
```

- 自动构建spring cloud 日志系统ELK

  compose内容

```
version: '3'
services:
 elasticsearch:
  container_name: elasticsearch
  image: docker.elastic.co/elasticsearch/elasticsearch:6.1.1
  command: elasticsearch
  ports:
   - "9200:9200"
   - "9300:9300"
  privileged: true
  networks:
   - default

 logstash:
  container_name: logstash
  image: docker.elastic.co/logstash/logstash:6.1.1
  command: logstash -f /etc/logstash/conf.d/logstash.conf
  volumes:
   # 挂载logstash配置文件
   - /app/service/logstash/config:/etc/logstash/conf.d
   - /app/service/logstash/build/:/opt/build/
  ports:
   - "6000:5000"
  privileged: true
  networks:
   - default

 kibana:
  container_name: kibana
  image: docker.elastic.co/kibana/kibana:6.1.1
  environment:
   - ELASTICSEARCH_URL=http://elasticsearch:9200
  ports:
```

```
    - "5601:5601"
  privileged: true
  networks:
    - default

networks:
  default:
    external:
      name: dockernet
```

logstash.conf存放在/app/service/config下

```
# For detail structure of this file
# Set: https://www.elastic.co/guide/en/logstash/current/configuration-file-structu
input {
  # For detail config for log4j as input,
  # See: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-log4j.htm
  tcp {
    mode => "server"
    host => "logstash"  #Logstash容器名称
    port => 9250
  }
}
filter {
  #Only matched data are send to output.
}
output {
  # For detail config for elasticsearch as output,
  # See: https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elastics
  elasticsearch {
    action => "index"          #The operation on ES
    hosts  => "elasticsearch:9200"   #ElasticSearch host, can be array.  elasticse
    index  => "applog"         #The index to write data to.
  }
}
```

运行命令

```
docker-compose -f docker-compose-elk.yml  up -d

如果elasticsearch 报这个错误
 ERROR: [1] bootstrap checks failed
 [1]: max virtual memory areas vm.max_map_count [65530] is too low, increase to a
解决方案：sudo sysctl -w vm.max_map_count=262144
```

演示效果

```
#创建一个索引
curl -XPUT http://172.31.65.26:9200/applog

#进入elk页面
http://172.31.65.26:5601/app/kibana
```

- 自动构建product 服务

  第一步：自动构建 mysql 微服务

```
# Docker Compose 配置文件，包含3大部分 version、services、networks
version: '3'
services:
  # 服务名称
  mysql:
    # 容器名称
    container_name: mysql
    # 镜像名称
    image: mysql:5.7
    # 暴露端口
    ports:
      - "3306:3306"
    # 设置卷挂载路径
    volumes:
      - /app/data/mysql:/var/lib/mysql
    # 环境变量
    environment:
      MYSQL_USER: root
```

```
      MYSQL_PASSWORD: agan
      MYSQL_ROOT_PASSWORD: agan

    # 设置权限 ：拥有root权限
    privileged: true

    networks:
      - default

networks:
  default:
    external:
      name: dockernet


#启动mysql容器
docker-compose -f docker-compose-mysql.yml  up -d
```

第二步：自动构建 product 微服务

```
#docker-compose-product.yml
version: '3'
services:
  product:
    container_name: e-book-product
    image: registry-docker:5000/e-book-product-core:0.0.1-SNAPSHOT
    ports:
      - "8083:8083"
    # 设置权限 ：拥有root权限
    privileged: true

    networks:
      - default

networks:
  default:
    external:
      name: dockernet
```

```
#启动容器
docker-compose -f docker-compose-product.yml  up -d
```

第三步：演示效果

```
#看注册中心
http://172.31.65.26:8761/
#产品查询接口
http://172.31.65.26:8083/product/findAllProduct
#看日志
http://172.31.65.26:5601/app/kibana
#调用链
http://172.31.65.26:9411/
```

- 自动构建自动构建 user order trade consumer 微服务

  第一步：构建相关服务

```
#docker-compose-service.yml
version: '3'
services:
  product:
    container_name: e-book-product
    image: registry-docker:5000/e-book-product-core:0.0.1-SNAPSHOT
    ports:
      - "8083:8083"
    networks:
      - default
  user:
    container_name: e-book-user
    image: registry-docker:5000/e-book-user-core:0.0.1-SNAPSHOT
    ports:
      - "8084:8084"
    networks:
      - default
  order:
    container_name: e-book-order
```

```
    image: registry-docker:5000/e-book-order-core:0.0.1-SNAPSHOT
    ports:
      - "8085:8085"
    networks:
      - default
  trade:
    container_name: e-book-trade
    image: registry-docker:5000/e-book-trade-core:0.0.1-SNAPSHOT
    ports:
      - "8086:8086"
    networks:
      - default
  consumer:
    container_name: e-book-consumer-order
    image: registry-docker:5000/e-book-consumer-order:0.0.1-SNAPSHOT
    ports:
      - "8090:8090"
    networks:
      - default

networks:
  default:
    external:
      name: dockernet


#启动容器命令
docker-compose -f docker-compose-service.yml  up -d
```

第二步：演示效果

```
#看注册中心
http://172.31.65.26:8761/
#产品查询接口
http://172.31.65.26:8083/product/findAllProduct
#创建订单
http://172.31.65.26:8090/createOrder
#看日志
http://172.31.65.26:5601/app/kibana
```

```
#调用链
http://172.31.65.26:9411/
```

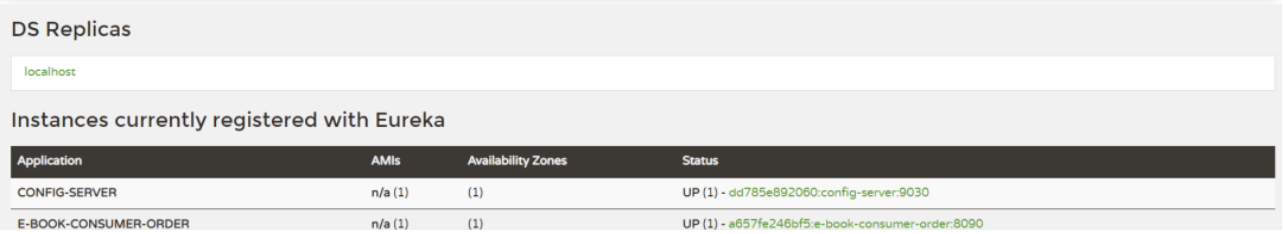
- 自动构建spring cloud 网关zuul

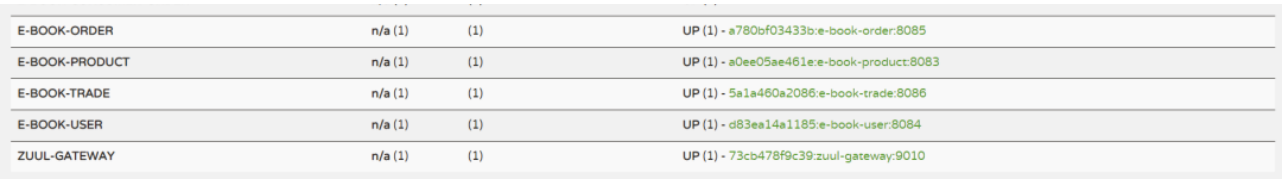
  第一步:构建zuul服务

```
#docker-compose-zuul.yml
version: '3'
services:
  zuul-gateway:
    container_name: zuul-gateway
    image: registry-docker:5000/zuul-gateway:0.0.1-SNAPSHOT
    ports:
      - "9010:9010"
    # 设置权限 ： 拥有root权限
    privileged: true

    networks:
      - default

networks:
  default:
    external:
      name: dockernet


#启动容器
docker-compose -f docker-compose-zuul.yml  up -d
```

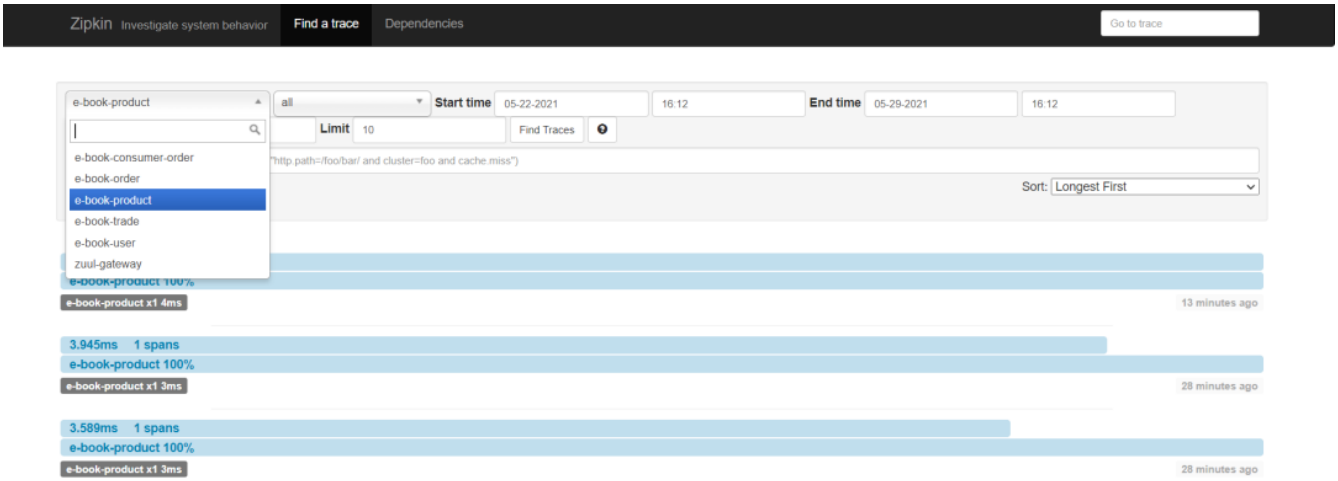
  第二步：演示效果

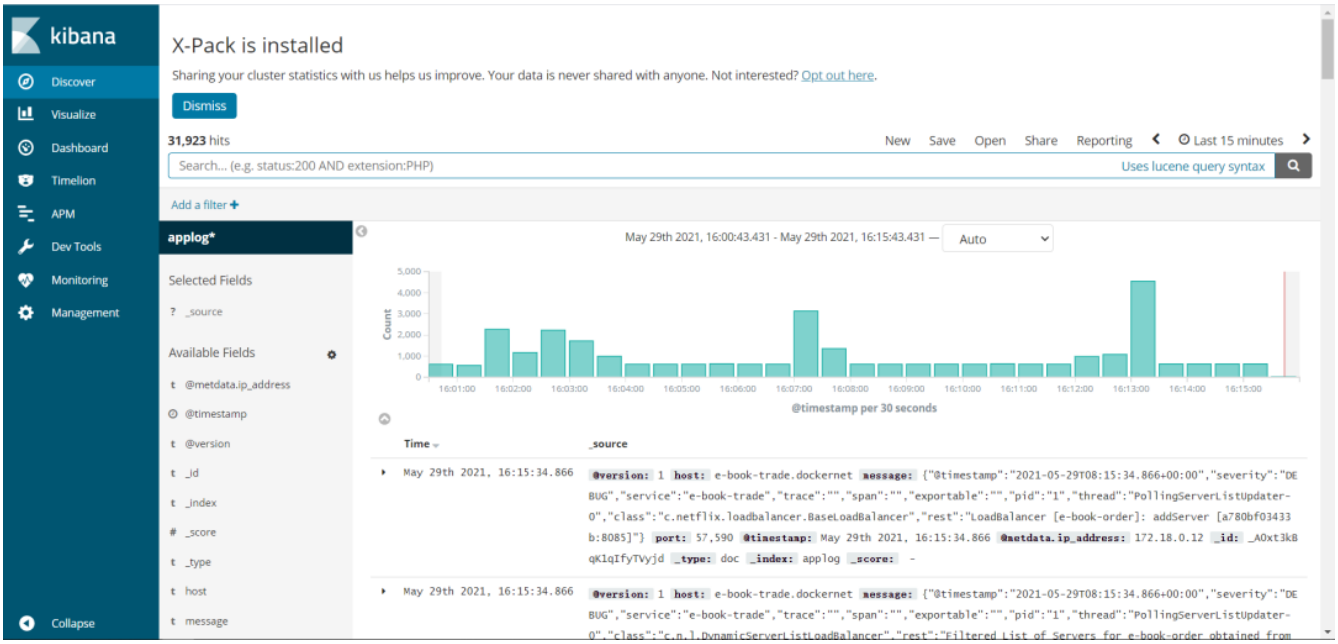
```
http://172.31.65.26:9010/e-book-consumer-order/createOrder
http://172.31.65.26:9010/e-book-product/product/findAllProduct
```





E-BOOK-ORDER	n/a (1)	(1)	UP (1) - a780bf03433b:e-book-order:8085
E-BOOK-PRODUCT	n/a (1)	(1)	UP (1) - a0ee05ae461e:e-book-product:8083
E-BOOK-TRADE	n/a (1)	(1)	UP (1) - 5a1a460a2086:e-book-trade:8086
E-BOOK-USER	n/a (1)	(1)	UP (1) - d83ea14a1185:e-book-user:8084
ZUUL-GATEWAY	n/a (1)	(1)	UP (1) - 73cb478f9c39:zuul-gateway:9010





以上，希望对你有所帮助！

1、一个时代即将终结！安卓应用告别APK格式

2、阿里终面：为什么SSD不能当做内存用？

3、旧手机别卖掉换脸盆了，自制服务器了解一下！

4、经典智力面试题：一家人过桥

点分享

点点赞 点在看

喜欢此内容的人还喜欢

把元素周期表也禁了？

IT服务圈儿