**Docker学习手册**

1. **体系结构**

**1.常见定义**

Docker 是一个开源的应用容器引擎，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。Docker重新定义了程序开发测试、交付和部署过程的工程，可以称为构建一次，到处运行。

Docker下的概念有容器，镜像等。

Kubernetes下的概念有namespace，pod，service,replicationController等**。**（service和replicationController只是建立在pod之上的抽象，最终是要作用于pod的）

**docker镜像**

在node节点上安装docker后，需要下载镜像文件，镜像文件就相当于一个微型的操作系统，是一个只读的模板。一个镜像可以包含一个完整的操作系统环境，里面只安装了tomcat和其他用户需要的程序。用户可以从官方或者其他人那里下载已经制作好的镜像。

Docker images 查看当前所有镜像

**Docker容器**

docker利用容器来运行应用，容器是从镜像创建的应用实例，每个容器都是被隔离的，可以单独进行开始，暂停，删除等操作。容器启动时在镜像上建立了一层可写层。

Docker ps 查看当前运行容器

Docker ps -a 查看当前所有容器

**Docker常用指令：**http://www.open-open.com/lib/view/open1423703640748.html

**Kubernetes**

Kubernetes是一个用来管理docker集群的系统，在master节点安装kubernetes后，可以使用kubectl工具来管理docker集群。

Pod是kubernetes最基本的调度单元，一个pod可以包含一个或者多个容器，例如，一个web项目包含前端，后端，数据库，就可以创建成一个包含三个容器的pod。Pod可以对资源进行限定，比如CPU，Memory等。

Namespace是隔离不同用户进程的，一个namespace可以包含很多pod。

**2.docker核心原理**

**2.1 namespace资源隔离**

首先，容器的文件系统需要被隔离，然后为了在分布式的环境下进行通信和定位，容器必然需要一个独立的IP、端口、路由等等，自然就想到了网络的隔离。同时，你的容器还需要一个独立的主机名以便在网络中标识自己。想到网络，顺其自然就想到通信，也就想到了进程间通信的隔离。可能你也想到了权限的问题，对用户和用户组的隔离就实现了用户权限的隔离。最后，运行在容器中的应用需要有自己的PID,自然也需要与宿主机中的PID进行隔离。

如下表

|  |  |
| --- | --- |
| 系统调用参数 | 隔离内容 |
| UTS | CLONE\_NEWUTS | 主机名与域名 |
| IPC | CLONE\_NEWIPC | 信号量、消息队列和共享内存 |
| PID | CLONE\_NEWPID | 进程编号 |
| Network | CLONE\_NEWNET | 网络设备、网络栈、端口等等 |
| Mount | CLONE\_NEWNS | 挂载点（文件系统） |
| User | CLONE\_NEWUSER | 用户和用户组 |

**2.2 Docker架构**

**2.3 client和daemon**

**2.4 libcontainer**

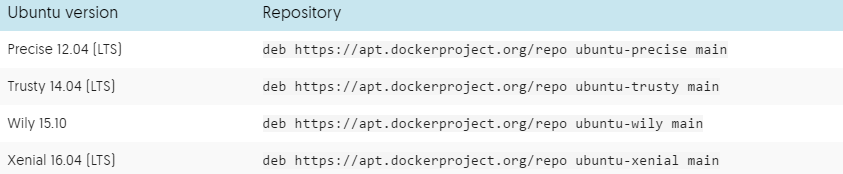
**2.5 docker存储**

**2.6 docker网络管理**

注：docker安装参见官网docker.io

apt-get update

vim /etc/apt/sources.list.d/docker.list



apt-get install docker-engine

**二、搭建私有仓库**

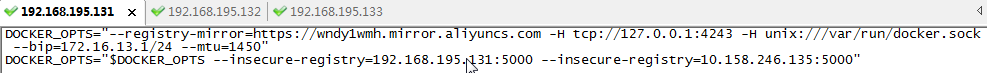
**1.在master节点创建镜像仓库**

docker pull registry

**2.配置镜像仓库**地址，需要在master节点和node节点都添加如下修改。

vi /etc/default/docker

修改成以下格式：

哈哈

修改完成后需要service docker restart（restart docker）

**3.通过registry镜像启动容器**

**3.1常用启动方法**

docker run -d -p 5000:5000 registry（不常用）

默认情况下，会将仓库存放于容器内的/tmp/registry目录下，这样如果容器被删除，则存放于容器中的镜像也会丢失，所以我们一般情况下会指定本地一个目录挂载到容器内的/tmp/registry下，如下：

docker run -d -p 5000:5000 -v /opt/data/registry:/tmp/registry registry:2.0

docker run -d -p 5000:5000 --restart=always --name registry \

-v `pwd`/data:/var/lib/registry \

registry:2.0

（官网指令，亲测可行，将registry数据保留在`pwd`/data下）

docker run -d -p 5000:5000 --restart=always --name registry \

-v `pwd`/config.yml:/etc/docker/registry/config.yml \

registry:2.0

(根据配置文件启动)

**ps：** 根据配置文件启动报错docker getsockopt: connection refused尚未解决

通过执行docker ps查看镜像仓库是否启动，如果启动成功能看到如下：

成功C:\Users\123\Desktop\2.png

**3.2 registry配置文件**

**3.3 insecure registry**

insecure registry: <https://docs.docker.com/registry/insecure/>

通过这种方法，可以通过https访问本地registry。

**4.测试**，在node节点执行 docker pull hello-world拉去新镜像。

执行命令docker tag hello-world 192.168.195.131:5000/hello-world给镜像添加新标识。

Docker push 192.168.195.131:5000/hello-world将镜像push到master节点的镜像仓库中。

Docker pull 192.168.195.131:5000/hello-world从master节点的镜像仓库获取镜像。

注： 当push到本地仓库时，保存在/var/lib/docker/aufs/下，挂载目录是由用户自己设置的。

**5.常用命令：**docker images 查看docker 镜像

docker ps 查看当前运行的容器 (docker ps –a)

-l：显示最后启动的容器

-a：同时显示停止的容器，默认只显示启动状态

docker stop [容器id]stop容器

docker restart [容器id] restart容器

service docker restart 重启docker

docker rm [容器id] 删除容器（-f删除正在运行中容器）

docker rmi [镜像名称：版本号] 删除镜像(-f强制删除)

docker logs [CONTAINER\_ID]  : 输出容器日志

-f：实时输出

docker save aarch64/registry > ./registry.tar

kubectl get pod –all-namespaces 查看所有pod

kubectl delete –namespace=[name] 删除pod

**6.docker run命令总结：**

<http://www.tuicool.com/articles/yiIVNz>讲创建镜像，容器的。

docker run -t -i ubuntu:14.04 /bin/bash

-i:标准输出给容器

-t:分配一个虚拟终端

-d：以守护进程方式运行（后台）

-p：默认匹配docker容器的5000端口号到宿主机的49153 to 65535端口

-p <HOT\_PORT>:<CONTAINER\_PORT>：指定端口号

docker run 192.168.195.133:5000/wdx-whale cowsay hello

**三、docker registry深入**

1. **docker registry ui（貌似只兼容registry v1）**

参考<https://hub.docker.com/r/atcol/docker-registry-ui/>

1. **registry管理**

本地启动registry后，访问http:192.168.195.131:5000/v2/\_catalog

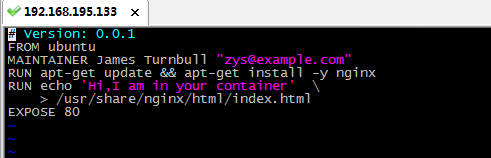
查看registry

**四、docker镜像制作**

**1.Dockerfile方式**

**1.1一个简单的示例**：

在static\_web目录下创建Dockerfile文件（touch Dockerfile），按下图进行编辑。



在当前文件夹下执行命令**docker build –t=“zys/static\_web” .**

或**docker build –no-cache –t=“zys/static\_web” .**（无缓存模式）

执行成功的话就会多出一个zys/static\_web镜像了

在宿主机上执行**docker run -d -p 8080:80 --name web zys/static\_web \**

**nginx –g “deamon off;”**

在浏览器中访问 192.168.195.133:8080就可以看到容器运行的效果了

**1.2 基于tomcat+jdk的镜像：**

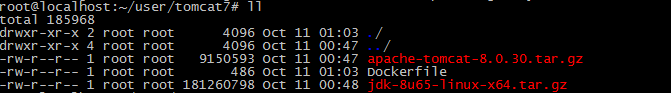
执行命令

**cd /root/user**

**mkdir tomcat7**

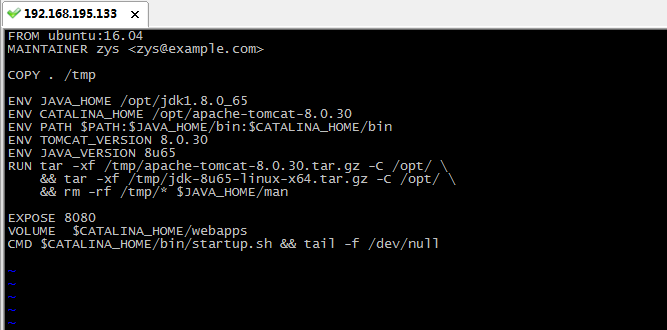
**touch Dockerfile**

然后在网上下载两个包，分别为apache-tomcat-8.0.30.tar.gz，jdk-8u65-linux-x64.tar.gz。将这两个包导入到/root/user/tomcat7文件夹下。如下



**vi Dockerfile** (编辑Dockerfile文件)

内容如下：



编辑完成后，执行

**docker build -t asdw747/tomcat7 .** （生成镜像）

**docker port tomcat\_app 8080** （查看对应宿主机端口号，如结果为32768）

**docker images** （查看当前镜像列表）

**docker run --name tomcat\_app -d -P asdw747/tomcat7** （运行镜像）

在浏览器中输入<http://192.168.195.133:32768/>可以看到效果。

/////////////////////////////////////////////////////////////////////

// **通过此镜像运行war包**  //

/////////////////////////////////////////////////////////////////////

**docker stop [容器id]**

**docker rm [容器id]**

在/root/user目录下创建一个webapps目录，

**cd /root/user/webapps**

**wget** [**https://tomcat.apache.org/tomcat-7.0-doc/appdev/sample/sample.war**](https://tomcat.apache.org/tomcat-7.0-doc/appdev/sample/sample.war)（从网上下载war包）

此时再启动asdw747/tomcat7镜像时，把存放war包的目录挂载到容器的tomcat/webapps目录下。

**docker run --name tomcat\_app \**

**-v /root/user/webapps:/opt/apache-tomcat-8.0.30/webapps \**

**-d -P asdw747/tomcat7**

查看对应端口后（如32769），访问<http://192.168.195.133:32769/sample>可以看到效果。

**!创建应用(manage)**

应用在k8s-install/k8s-install/docker-apps/下

启动前需要先修改yaml文件，然后执行./build.sh或者手动执行kubectl create -f manage-svc.yaml

1. **启动系统应用**

当前状态下执行kubectl get pod --all-namespaces发现kube-system

的容器并未启动，因为有一些应用的配置文件需要修改,三个节点也需要先下载镜像。

先在master，node1，node2节点上分别下载镜像:

docker pull 10.158.246.135:5000/gcr.io/google\_containers/pause:2.0

docker tag 10.158.246.135:5000/gcr.io/google\_containers/pause:2.0 gcr.io/google\_containers/pause:2.0

进入到k8s-install/k8s-install/system-apps/heapster下:

vi heapster-rc.yaml将其中的source的ip地址改为本机ip，退出。

Kubectl create –f heapster-rc.yaml

Kubectl create –f heapster-svc.yaml

执行kubectl get pod --all-namespaces发现应用都正常启动了。