目录

**[Part 1 部署docker容器虚拟化平台](#_Toc8816_WPSOffice_Level1)** **[1](#_Toc8816_WPSOffice_Level1)**

**[一、 Docker概述](#_Toc27361_WPSOffice_Level1)** **[1](#_Toc27361_WPSOffice_Level1)**

[1. Docker概述](#_Toc27361_WPSOffice_Level2) [1](#_Toc27361_WPSOffice_Level2)

[2. 沙盒：](#_Toc26972_WPSOffice_Level2) [1](#_Toc26972_WPSOffice_Level2)

[3. Docker发展](#_Toc14333_WPSOffice_Level2) [1](#_Toc14333_WPSOffice_Level2)

[4. 形象举例](#_Toc3092_WPSOffice_Level2) [2](#_Toc3092_WPSOffice_Level2)

[5. docker容器技术和虚拟机对比](#_Toc13646_WPSOffice_Level2) [3](#_Toc13646_WPSOffice_Level2)

[6. Docker核心技术](#_Toc22734_WPSOffice_Level2) [4](#_Toc22734_WPSOffice_Level2)

[7. Docker特性](#_Toc8625_WPSOffice_Level2) [4](#_Toc8625_WPSOffice_Level2)

**[二、 部署docker容器化虚拟平台](#_Toc26972_WPSOffice_Level1)** **[6](#_Toc26972_WPSOffice_Level1)**

[1. 实验环境：CentOS 7.4\_64位 IP：192.168.1.100](#_Toc14867_WPSOffice_Level2) [6](#_Toc14867_WPSOffice_Level2)

[2. 配置本地yum源](#_Toc12348_WPSOffice_Level2) [7](#_Toc12348_WPSOffice_Level2)

[3. 安装docker软件包](#_Toc1672_WPSOffice_Level2) [7](#_Toc1672_WPSOffice_Level2)

[4. 启动docker平台](#_Toc13632_WPSOffice_Level2) [7](#_Toc13632_WPSOffice_Level2)

[5. 查看docker信息](#_Toc16495_WPSOffice_Level2) [7](#_Toc16495_WPSOffice_Level2)

[6. 下载镜像](#_Toc5194_WPSOffice_Level2) [7](#_Toc5194_WPSOffice_Level2)

[7. 从Docker Hub中搜索复合条件的Image镜像](#_Toc17971_WPSOffice_Level2) [7](#_Toc17971_WPSOffice_Level2)

**[三、 docker平台基本使用方法](#_Toc14333_WPSOffice_Level1)** **[9](#_Toc14333_WPSOffice_Level1)**

[1. 运行一个container并加载镜像centos，运行起来这个实例后，在实例中执行/bin/bash命令](#_Toc16125_WPSOffice_Level2) [9](#_Toc16125_WPSOffice_Level2)

[2. 在 container 中启动一个长久运行的进程，不断向 stdin 输出hello world 。模拟一个后台运行的服务](#_Toc9279_WPSOffice_Level2) [9](#_Toc9279_WPSOffice_Level2)

[3. 开启路由转发功能](#_Toc21511_WPSOffice_Level2) [11](#_Toc21511_WPSOffice_Level2)

[4. 启动、停止、重启container容器实例](#_Toc31415_WPSOffice_Level2) [11](#_Toc31415_WPSOffice_Level2)

**[四、 docker镜像制作和发布方法](#_Toc3092_WPSOffice_Level1)** **[11](#_Toc3092_WPSOffice_Level1)**

[1. Docker Image的两种制作方法](#_Toc9856_WPSOffice_Level2) [11](#_Toc9856_WPSOffice_Level2)

[2. docker commit制作镜像](#_Toc15817_WPSOffice_Level2) [12](#_Toc15817_WPSOffice_Level2)

[3. docker build](#_Toc31087_WPSOffice_Level2) [12](#_Toc31087_WPSOffice_Level2)

[4. Docker镜像的发布](#_Toc3501_WPSOffice_Level2) [15](#_Toc3501_WPSOffice_Level2)

**[五、 Container容器端口映射](#_Toc13646_WPSOffice_Level1)** **[16](#_Toc13646_WPSOffice_Level1)**

[1. 启动container](#_Toc22604_WPSOffice_Level2) [16](#_Toc22604_WPSOffice_Level2)

[2. 访问正在运行的container容器实例](#_Toc2967_WPSOffice_Level2) [16](#_Toc2967_WPSOffice_Level2)

**[Part2 docker容器命名和资源配额控制](#_Toc22734_WPSOffice_Level1)** **[1](#_Toc22734_WPSOffice_Level1)**

**[一、 docker容器命名和重命名](#_Toc8625_WPSOffice_Level1)** **[1](#_Toc8625_WPSOffice_Level1)**

[1. 查看容器的名字：](#_Toc31923_WPSOffice_Level2) [1](#_Toc31923_WPSOffice_Level2)

[2. 命名和重命名](#_Toc31512_WPSOffice_Level2) [1](#_Toc31512_WPSOffice_Level2)

**[二、 创建docker容器实例时指定主机名](#_Toc14867_WPSOffice_Level1)** **[2](#_Toc14867_WPSOffice_Level1)**

[1. 语法：](#_Toc29359_WPSOffice_Level2) [2](#_Toc29359_WPSOffice_Level2)

**[三、 docker容器资源配额控制——cpu](#_Toc12348_WPSOffice_Level1)** **[2](#_Toc12348_WPSOffice_Level1)**

[1. docker容器资源配额控制](#_Toc15309_WPSOffice_Level2) [2](#_Toc15309_WPSOffice_Level2)

[2. 了解cpu周期控制](#_Toc30777_WPSOffice_Level2) [3](#_Toc30777_WPSOffice_Level2)

[3. CPU core核心控制](#_Toc17743_WPSOffice_Level2) [4](#_Toc17743_WPSOffice_Level2)

[4. CPU配额控制参数的混合使用](#_Toc14825_WPSOffice_Level2) [6](#_Toc14825_WPSOffice_Level2)

[5. 当容器命令运行结束后，自动删除容器](#_Toc24505_WPSOffice_Level2) [9](#_Toc24505_WPSOffice_Level2)

**[四、 docker容器资源配额控制——内存](#_Toc1672_WPSOffice_Level1)** **[10](#_Toc1672_WPSOffice_Level1)**

[1. 内存](#_Toc24036_WPSOffice_Level2) [10](#_Toc24036_WPSOffice_Level2)

**[五、 docker容器资源配额控制——IO](#_Toc13632_WPSOffice_Level1)** **[10](#_Toc13632_WPSOffice_Level1)**

[1. IO](#_Toc25918_WPSOffice_Level2) [10](#_Toc25918_WPSOffice_Level2)

[2. 为什么阿里云平台上普通云盘的IO为：1000IOPS，为什么这么小](#_Toc6032_WPSOffice_Level2) [10](#_Toc6032_WPSOffice_Level2)

**[六、 docker数据映射](#_Toc16495_WPSOffice_Level1)** **[12](#_Toc16495_WPSOffice_Level1)**

[1. docker数据映射](#_Toc32172_WPSOffice_Level2) [12](#_Toc32172_WPSOffice_Level2)

Part 1 部署docker容器虚拟化平台

1. Docker概述

实验环境：CentOS 7.4-63 64位

1. Docker概述

Docker 是一个开源的应用容器引擎，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，也可以实现虚拟化。容器是完全使用沙盒机制，相互之间不会有任何接口（类似 iPhone 的 app）。几乎没有性能开销,可以很容易地在机器和数据中心中运行。最重要的是,他们不依赖于任何语言、框架或包装系统。

1. 沙盒：

沙盒也叫沙箱，英文sandbox。在计算机领域指一种虚拟技术，且多用于计算机安全技术。安全软件可以先让它在沙盒中运行，如果含有恶意行为，则禁止程序的进一步运行，而这不会对系统造成任何危害。

1. Docker发展

Docker 是 dotCloud 公司开源的一个基于 LXC 的高级容器引擎，源代码托管在 Github 上, 基于 go语言并遵从 Apache2.0 协议开源。

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 container 中，然后发布到任何流行的 Linux 机器上。

1. 形象举例

现在接触的软件是怎么发布的？ windows下的 2016-0ffice.exe 在 xp 下兼容不太好；mk.rpm在 redhat 系列 Linux 上运行，但是不能在其他 linux 版本上运行。

例： /mnt/Packages/vsftpd-3.0.2-10.el7.x86\_64.rpm #这个包是 rhel7，就不能在 rhel6 运

行。

现在软件包必须和系统相关。docker 镜像一次编译，到处运行。

android：linux -> JVM -> java 程序 app

linux -> docker -> 服务做成镜像就可以直接运行起来

windows -> docker -> 服务做成镜像就可以直接运行起来

扩展：

LXC为Linux Container的简写。Linux Container容器是一种内核虚拟化技术，可以提供轻量级的虚拟化，以便隔离进程和资源，而且不需要提供指令解释机制以及全虚拟化的其他复杂性。

LXC主要通过来自kernel的namespace实现每个用户实例之间的相互隔离，通过cgroup实现对资源的配额和度量。

官方网站：<https://www.docker.com/>

docker [ˈdɒkə(r)] 码头工人

注：docker服务相当于鲸鱼，container容器是集装箱

container：集装箱，容器

docker：码头工人

源代码下载

https//github.com/docker/docker

1. docker容器技术和虚拟机对比

相同点：docker容器技术和虚拟机技术，都是虚拟化技术

不同点：

· 虚拟化：

Server(硬件)

Host OS(windows/centos)

Hyperviser(VMWare/KVM)

Guest OS(虚拟机操作系统)

Bins/Libs(库)

Apps

· 容器：

Server（硬件）

Host OS(centos7)

Docker Engine

Bins/Libs(库)

Apps

/\*\* 相比之下，Docker容器技术和VM虚拟化技术之间相差一个GuestOS，占用资源更少，效率更高，Docker使用当前系统内核 \*\*/

物理系统启动时间：10+s

Docker实例启动时间：1~2s

1. Docker架构(单机)

服务器A

Docker Hub(镜像服务器)

Docker client(远程控制客户端)

1.服务器运行Docker引擎(服务)

2.服务器A从镜像服务器上下载(pull)镜像;同理，也可上传

3.服务器A启动docker容器实例

4.通过Docker client对docker容器虚拟化平台进行控制

Image和Container的关系：image可以理解为一个系统镜像，Container是Image在运行时的一个状态。

如果拿虚拟机作类比，Image就是关机状态下的磁盘文件，Container是虚拟机运行时的磁盘文件，包括内存数据。

dockerhub：dockerhub是docker官方的镜像存储站点，其中提供了很多常用的镜像提供用户下载，如ubuntu，centos等系统镜像。通过dockerhub用户也可以发布自己的docker镜像，为此用户需要注册一个账号，在网站上创建一个docker仓库。

1. Docker核心技术
2. Namespace：实现Container的进程，网络，消息，文件系统和主机名的隔离
3. Cgroup：实现对资源的配额和度量

注：Cgroup的配额，可以指定实例使用的cpu个数，内存大小等(就像VmWare虚拟机中的硬件配置参数)

1. Docker特性

文件系统隔离：每个进程容器运行在一个完全独立的根文件系统里。

资源隔离：通过cgroup实现将系统资源，如CPU和内存等可以分配到不同的容器中

网络隔离：每个进程容器运行在自己的网络空间，虚拟接口和IP地址

日志记录：Docker将会收集和记录每个进程容器的标准流(stdout/stderr/stdin)，用于实时检索或批量检索

变更管理：容器文件系统的变更可以提交到新的镜像中，并可重复使用以创建更多的容器。无需使用模板或手动配置

交互式shell：Docker可以分配一个虚拟终端并关联到任何容器的标准输入上，例如运行一个一次性交互shell

优点：

1. 一些优势和VM一样，但不是所有都一样

比VM小，比VM快，Docker容器的尺寸减小相比整个虚拟机大大简化了分步到云和从云分发时间和开销。Docker启动一个容器实例的时间很短，一两秒就可以启动一个实例。

1. 对于在笔记本电脑里，数据中心的虚拟机，以及任何的云上，运行相同的没有变化的应用程序，IT的发布速度更快。

Docker是一个开放的平台，构建，发布和运行分布式应用程序

Docker使应用程序能够快速从组件组装和避免开发和生产环境之间的摩擦

1. 可以在部署公司局域网或虚拟机上使用它
2. 开发人员并不关心具体是哪个Linux系统

使用Docker，开发人员可以根据所有依赖关系，针对他们所选择的操作系统，构建相应的软件。然后在部署时，一切是完全一样的，因为一切都在DockerImage的容器在其上运行。开发人员负责并能够确保所有的相关性得到满足

1. Google，微软，亚马逊，IBM等都支持Docker
2. Docker支持Unix/Linux操作系统，也支持Windows或Mac

缺点和局限性：

1. Docker用于应用程序时是最有用的，但并不包含数据。日志，跟踪和数据库等通常应放在Docker容器外。一个容器的镜像通常都很小，不适合存大量的数据，存储可以通过外部挂载的方式使用。比如使用NFS，ipsan，MFS等，-v映射磁盘分区

一句话：docker只用于计算，存储交给别人

oracle不适合使用docker来运行，太大了，存储的数据太多

1. 部署docker容器化虚拟平台

实验环境准备

1. 实验环境：CentOS 7.4\_64位 IP：192.168.1.100
2. 创建后端盘：

# cd /var/lib/libvirt/images

# qemu-img create -f qcow2 docker.img 30G

1. 创建虚拟路由器

# cd /etc/libvirt/qemu/networks/

# vim docker.xml

<network>

<name>docker</name>

<forward mode='nat'/>

<bridge name='docker' stp='on' delay='0'/>

<ip address='192.168.1.254' netmask='255.255.255.0'>

<dhcp>

<range start='192.168.1.100' end='192.168.1.200'/>

</dhcp>

</ip>

</network>

# virsh net-define docker.xml

# virsh net-start docker

# virsh net-autostart docker

1. 装机

# virt-install --virt-type=kvm --name=docker --vcpus=1 --memory=1024 --location=/ISO/CentOS-7-x86\_64-Everything-1708.iso --disk path=/var/lib/libvirt/images/docker.img,format=qcow2 --network bridge=docker --graphics none --extra-args='console=ttyS0' --force

把docker-rpm.tar.gz上传到 linux 系统上:

# tar xvf docker\_rpm.tar.gz

1. 配置本地yum源
2. 安装docker软件包
3. 启动docker平台

# systemctl start docker

# systemctl enable docker

1. 查看docker信息

# docker info

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Data loop file: /var/lib/docker/devicemapper/devicemapper/data

1. 下载镜像
2. 从Docker Hub中搜索复合条件的Image镜像

# docker search centos

名字 描述 受欢迎程度 是否官方提供 如果OFFICIAL为ok，说明可以放心使用

· 方法 1：从公网 docker hub 拉取（下载）image pull：拉

[root@xuegod63 ~]# docker pull docker.io/centos

#报错了，因为网络的问题。 无法连接到 dockerhub 下载镜像。 如果你的网络没有问题，你可以下载。

解决：换一个 docker 下载地址或者使用阿里云 docker 镜像加速，提升 pull 的速度

要登录容器 Hub 服务 https://cr.console.aliyun.com 的控制台，使用你的支付宝帐号，第

一次登录时，需要设置一个独立的密码，左侧的加速器帮劣页面就会显示为你独立分配的加速地址。

网易镜像：

https://c.163.com/hub#/m/home/ #需要先注册登彔上，才能打开此站点

[root@xuegod63 ~]# vim /etc/docker/daemon.json #改成以下内容

改： {}

为：

{

"registry-mirrors": ["https://e9yneuy4.mirror.aliyuncs.com"]

}

[root@xuegod63 ~]# systemctl daemon-reload

[root@xuegod63 ~]# systemctl restart docker

[root@xuegod63 ~]# docker pull docker.io/centos #再下载，就可以了。

· 方法 2：把之前下载好的 image 镜像导入 image：

把 docker.io-centos.tar 镜像上传到 linux 上

参数： -i " docker.io-centos.tar " 指定载入的镜像归档。

[root@xuegod63 ~]# docker load -i /root/docker.io-centos.tar

· 方法 3：直接下载其他站点的镜像

[root@xuegod63 ~]# docker pull hub.c.163.com/library/tomcat:latest

[root@xuegod63 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/centos latest 1e1148e4cc2c 3 months ago 201.8 MB

1. docker平台基本使用方法
2. 运行一个container并加载镜像centos，运行起来这个实例后，在实例中执行/bin/bash命令

docker常用参数：

run 运行

-i 以交互模式运行容器，通常与-t同时使用

-t 为容器重新分配一个伪输入终端，通常与-i同时使用

[root@docker ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/centos latest 1e1148e4cc2c 3 months ago 201.8 MB

[root@docker ~]# docker run -it docker.io/centos /bin/bash

//启动一个实例，耗时2-3秒

[root@d033e1da9672 /]# ls

//查看实例环境

anaconda-post.log etc lib64 opt run sys var

bin home media proc sbin tmp

dev lib mnt root srv usr

退出容器：

[root@d033e1da9672 /]# exit

1. 在 container 中启动一个长久运行的进程，不断向 stdin 输出hello world 。模拟一个后台运行的服务

docker 常用参数：

-d 后台运行容器，并返回容器 ID；

-c 后面跟待完成的命令

# docker run -d centos:latest -c

# docker run -d centos:latest /bin/bash -c "while tru;do echo hello world ;sleep 1;done"

beb4660936101010099c2538f5e04a111ad661183ecc671e03d90e05ee2ab909

#容器ID

从一个容器中取日志，查看输出的内容

语法：docker logs 容器实例的Name/ID

查看容器ID和名字：

[root@docker ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

beb466093610 docker.io/centos:latest "/bin/sh -c 'while tr" 44 minutes ago Up 44 minutes determined\_tesla

//列出所有容器

[root@docker ~]# docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

beb466093610 docker.io/centos:latest "/bin/sh -c 'while tr" 46 minutes ago Up 45 minutes determined\_tesla

ca04ac61923f docker.io/centos:latest "/bin/bash" 4 hours ago Exited (0) 2 hours ago fervent\_cori

8fe1ee839f10 docker.io/centos:latest "/bin/bash" 5 hours ago Exited (0) 4 hours ago stoic\_pike

25d3cd8abfae docker.io/centos "/bin/bash" 5 hours ago Exited (0) 5 hours ago goofy\_volhard

d033e1da9672 docker.io/centos "/bin/bash" 5 hours ago Exited (0) 5 hours ago evil\_roentgen

//列出所有本地镜像

[root@docker ~]# docker images

//杀死一个正在后台运行的容器

查看要杀死的容器ID：

[root@docker ~]# docker ps

[root@docker ~]# docker kill beb466093610

beb466093610

[root@docker ~]# docker ps

1. 开启路由转发功能

开启网络转发功能：

[root@docker ~]# echo 1 > /proc/sys/net/ipv4/ip\_forward

[root@docker ~]# vim /etc/sysctl.conf

net.ipv4.ip\_forward = 1

[root@docker ~]# sysctl -p

#刷新

net.ipv4.ip\_forward = 1

不开启会报错：IPv4 forwarding is disabled.

1. 启动、停止、重启container容器实例

[root@docker ~]# docker run -d docker.io/centos:latest /bin/sh -c "while true;do echo hello world;sleep 1;done"

[root@docker ~]# docker ps

[root@docker ~]# docker stop caf2c0b4b2f6

[root@docker ~]# docker ps

[root@docker ~]# docker start caf2c0b4b2f6

[root@docker ~]# docker ps

删除指定 container ： rm

[root@docker ~]# docker rm caf2c0b4b2f6

Error response from daemon: You cannot remove a running container caf2c0b4b2f65f61cd02aaf8c659d46012ab9e205b2e2d67dbfec681fdcafd27. Stop the container before attempting removal or use -f

//可以看到真要删除可以使用-f选项

[root@docker ~]# docker rm caf2c0b4b2f6 -f

caf2c0b4b2f6

[root@docker ~]# docker ps

1. docker镜像制作和发布方法
2. Docker Image的两种制作方法
3. docker commit #保存container的当前状态到image后，生成对应的image
4. docker build #使用Dockerfile文件自动化制作image
5. docker commit制作镜像
6. 创建好一个apache工具的容器镜像

[root@docker ~]# docker run -it docker.io/centos:latest /bin/bash

[root@b936e8c4bd07 /]# yum -y install httpd

#在 container 中安装 apache 软件包

[root@b936e8c4bd07 /]# exit

根据容器当前状态做一个image镜像：创建了一个安装了apache工具的centos镜像

语法：docker commit <container ID> <image\_name>

例如：

查看容器ID：

[root@docker ~]# docker ps -a | head -2

[root@docker ~]# docker commit b936e8c4bd07 docker.io/centos:wt\_httpd

sha256:2294903aff84c909c82d5f405465d78eb14c49d1907bbe572189f73c48b99302

[root@docker ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

docker.io/centos wt\_httpd 2294903aff84 24 seconds ago 312.4 MB

docker.io/centos latest 1e1148e4cc2c 3 months ago 201.8 MB

使用新创建的容器镜像生成一台容器实例：

[root@docker ~]# docker run -it docker.io/centos:wt\_httpd /bin/bash

[root@94091517ad61 /]# rpm -q httpd

1. docker build
2. 创建工作目录

[root@docker ~]# mkdir docker\_build

1. 编辑Dockerfile

[root@docker docker\_build]# vim Dockerfile

FROM docker.io/centos:latest

MAINTAINER <yyzh@tedu.cn>

RUN yum -y install httpd

ADD start.sh /usr/local/bin/start.sh

ADD index.html /var/www/html/index.html

#注释：

FROM 基于哪个镜像

MAINTAINER 镜像创建者

RUN 安装软件用

ADD 将文件<src>拷贝到新产生的镜像文件系统对应的路径<dest>。所有拷贝到新景祥中的文件和文件夹权限为0755，uid和gid为0

CMD echo hello world container启动时执行的命令或启动服务，但是一个Dockerfile中只能有一条CMD命令，多条则执行最后一条CMD

1. 创建 start.sh 脚本启劢 httpd 服务和 apache 默认首页 index.html 文件

[root@docker docker\_build]# echo "/usr/sbin/httpd -DFOREGROUND" > start.sh

[root@docker docker\_build]# chmod a+x start.sh

[root@docker docker\_build]# echo "docker image build test" > index.html

[root@docker docker\_build]# ls

Dockerfile index.html start.sh

1. 使用build来创建新的image

语法：docker build

[root@docker docker\_build]# docker build -t docker.io/centos:http ./

Sending build context to Docker daemon 4.096 kB

Step 1 : FROM docker.io/centos:latest

---> 1e1148e4cc2c

Step 2 : MAINTAINER <yyzh@tedu.cn>

---> Running in c75499c5d3cd

---> c8dd20981c4f

Removing intermediate container c75499c5d3cd

Step 3 : RUN yum -y install httpd

---> Running in e223319b45b0

Loaded plugins: fastestmirror, ovl

Determining fastest mirrors

\* base: mirrors.163.com

\* extras: mirrors.163.com

\* updates: mirrors.163.com

Resolving Dependencies

--> Running transaction check

---> Package httpd.x86\_64 0:2.4.6-88.el7.centos will be installed

--> Processing Dependency: httpd-tools = 2.4.6-88.el7.centos for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Processing Dependency: system-logos >= 7.92.1-1 for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Processing Dependency: /etc/mime.types for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Processing Dependency: libaprutil-1.so.0()(64bit) for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Processing Dependency: libapr-1.so.0()(64bit) for package: httpd-2.4.6-88.el7.centos.x86\_64

--> Running transaction check

---> Package apr.x86\_64 0:1.4.8-3.el7\_4.1 will be installed

---> Package apr-util.x86\_64 0:1.5.2-6.el7 will be installed

---> Package centos-logos.noarch 0:70.0.6-3.el7.centos will be installed

---> Package httpd-tools.x86\_64 0:2.4.6-88.el7.centos will be installed

---> Package mailcap.noarch 0:2.1.41-2.el7 will be installed

--> Finished Dependency Resolution

Dependencies Resolved

================================================================================

Package Arch Version Repository Size

================================================================================

Installing:

httpd x86\_64 2.4.6-88.el7.centos base 2.7 M

Installing for dependencies:

apr x86\_64 1.4.8-3.el7\_4.1 base 103 k

apr-util x86\_64 1.5.2-6.el7 base 92 k

centos-logos noarch 70.0.6-3.el7.centos base 21 M

httpd-tools x86\_64 2.4.6-88.el7.centos base 90 k

mailcap noarch 2.1.41-2.el7 base 31 k

Transaction Summary

================================================================================

Install 1 Package (+5 Dependent packages)

Total download size: 24 M

Installed size: 31 M

Downloading packages:

warning: /var/cache/yum/x86\_64/7/base/packages/apr-util-1.5.2-6.el7.x86\_64.rpm: Header V3 RSA/SHA256 Signature, key ID f4a80eb5: NOKEY

Public key for apr-util-1.5.2-6.el7.x86\_64.rpm is not installed

--------------------------------------------------------------------------------

Total 869 kB/s | 24 MB 00:28

Retrieving key from file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

Importing GPG key 0xF4A80EB5:

Userid : "CentOS-7 Key (CentOS 7 Official Signing Key) <security@centos.org>"

Fingerprint: 6341 ab27 53d7 8a78 a7c2 7bb1 24c6 a8a7 f4a8 0eb5

Package : centos-release-7-6.1810.2.el7.centos.x86\_64 (@CentOS)

From : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

Installing : apr-1.4.8-3.el7\_4.1.x86\_64 1/6

Installing : apr-util-1.5.2-6.el7.x86\_64 2/6

Installing : httpd-tools-2.4.6-88.el7.centos.x86\_64 3/6

Installing : centos-logos-70.0.6-3.el7.centos.noarch 4/6

Installing : mailcap-2.1.41-2.el7.noarch 5/6

Installing : httpd-2.4.6-88.el7.centos.x86\_64 6/6

Verifying : mailcap-2.1.41-2.el7.noarch 1/6

Verifying : apr-util-1.5.2-6.el7.x86\_64 2/6

Verifying : httpd-tools-2.4.6-88.el7.centos.x86\_64 3/6

Verifying : httpd-2.4.6-88.el7.centos.x86\_64 4/6

Verifying : apr-1.4.8-3.el7\_4.1.x86\_64 5/6

Verifying : centos-logos-70.0.6-3.el7.centos.noarch 6/6

Installed:

httpd.x86\_64 0:2.4.6-88.el7.centos

Dependency Installed:

apr.x86\_64 0:1.4.8-3.el7\_4.1

apr-util.x86\_64 0:1.5.2-6.el7

centos-logos.noarch 0:70.0.6-3.el7.centos

httpd-tools.x86\_64 0:2.4.6-88.el7.centos

mailcap.noarch 0:2.1.41-2.el7

Complete!

---> 64b68e26cbd0

Removing intermediate container e223319b45b0

Step 4 : ADD start.sh /usr/local/bin/start.sh

---> 3508c13cb72e

Removing intermediate container c44ce552f937

Step 5 : ADD index.html /var/www/html/index.html

---> 397c67ca580a

Removing intermediate container d58725d334ec

Successfully built 397c67ca580a

docker镜像=应用/程序+库

1. Docker镜像的发布

· 方法1：Save Image To TarBall

· 方法2：Push Image To Docker Hub

1. 保存Image到tar包

· 语法：dockersave -o 导出的镜像名/tar 本地镜像名：镜像标签

[root@docker docker\_build]# docker save -o docker.io-centos-httpd-docker-image.tar docker.io/centos:http

[root@docker docker\_build]# ll -h

-rw------- 1 root root 307M 3月 14 15:13 docker.io-centos-httpd-docker-image.tar

1. 使用导入本地镜像：

[root@docker ~]# docker rmi docker.io/centos:http

[root@docker ~]# docker load -i docker\_build/docker.io-centos-httpd-docker-image.tar

· 方法2：Push Image To Docker Hub 发布到外网

1. Signup on docker hub & create repo 注册一个帐号

https://hub.docker.com/

1. Login to docker hub

# docker login -u userabc -p abc123 -e userab@gmail.com

1. Push image to docker hub #上传镜像

# docker push centos:httpd

1. Pull image from docker hub #下载镜像

# docker pull userabc/centos:httpd # 用户名/镜像名

1. Container容器端口映射
2. 启动container

[root@docker ~]# docker run -d -p 80:80 docker.io/centos:http /bin/bash -c /usr/local/bin/start.sh

1. 访问正在运行的container容器实例

docker exec -it <container ID | name> /bin/bash

Part2 docker容器命名和资源配额控制

前期准备：

· 还原快照到已安装好的docker的状态

# systemctl start docker

# docker images

1. docker容器命名和重命名
2. 查看容器的名字：

# docker ps -a

1. 命名和重命名
2. 容器命名语法

docker run -d --name 容器实例名 容器镜像名 要执行的命令

1. 容器重命名语法

docker rename 旧容器名 新容器名

1. 运行一个名字为docker1的容器

# docker run -d docker0 docker.io/centos:latest /bin/bash

//此docker不会挂在后台，在后台执行/bin/bash后直接退出

[root@docker ~]# docker run -itd --name docker1 docker.io/centos:latest /bin/bash

//相当于在后台创建一个虚拟终端，容器会一直挂在后台

[root@docker ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ce61b37bb58b docker.io/centos:latest "/bin/bash" 4 minutes ago Up 4 minutes docker1

1. 将docker1容器重命名

[root@docker ~]# docker rename docker1 docker2

[root@docker ~]# docker rename docker{2,1}

1. 创建docker容器实例时指定主机名
2. 语法：

docker run -it --name 容器名 -h 指定主机名 镜像 /bin/bash

[root@docker ~]# docker run -it --name docker2 -h docker2 docker.io/centos:latest /bin/bash

[root@docker2 /]# hostname

docker2

1. docker容器资源配额控制——cpu
2. docker容器资源配额控制
3. 启动docker容器时，指定CPU，内存，硬盘性能等的硬件资源使用份额

· Docker通过cgroup来控制容器使用的资源配额，包括CPU、内存、硬盘三大方面，基本覆盖了常见的资源配额和使用量控制

· cgroup概述：

· cgroup是Control Groups的缩写，是Linux内核提供的一种可以限制、记录、隔离进程组所使用的物理资源(如cpu、memory、磁盘IO等)的机制，被LXC、docker等多个项目用于实现进程资源的控制。cgroup将任意进程进行分组化管理的Linux内核功能。cgroup本身是提供将进程进行分组化管理的功能和接口的基础结构，IO或内存的分配控制等具体的资源管理功能是通过这个功能来实现的。

· 为什么要进行硬件配额

1. 当多个容器运行时，防止某容器把所有的硬件都占用了(比如一台被黑的容器)

例 1：给容器实例分配 512 权重的 cpu 使用份额

[root@xuegod63 ~]# docker run --help | grep cpu-shares

-c, --cpu-shares int CPU shares (relative weight)

\* cpu 配额参数：

-c, --cpu-shares int CPU shares (relative weight) 在创建容器时指定容器所使用的 CPU 份额值。

cpu-shares 的值不能保证可以获得 1 个 vcpu 或者多少 GHz 的 CPU 资源，仅仅只是一个弹性的加权值。

\* 默认情况下，每个 docker 容器的 cpu 份额都是 1024。单独一个容器的份额是没有意义的，只有在同时运行多个容器时，容器的 cpu 加权的效果才能体现出来。

1. 两个容器A、B的cpu份额分别为1000和500,1000+500>1024，是否超出限制？

- 并没有，A使用1024的2/3，B使用1024的1/3

- cgroups 只在容器分配的的资源紧缺时，也就是说在需要对容器使用的资源迚行限制时，才会生效。因此，无法单纯根据某个容器的 cpu 份额来确定有多少 cpu 资源分配给它，资源分配结果取决于同时运行的其他容器的 cpu 分配和容器中进程运行情况。

例1：给容器实例分配512权重 的cpu使用份额

参数：--cpu-shares 512

[root@docker01 ~]# docker run -it --cpu-shares 512 centos /bin/bash

[root@c06f30b4c11c /]# cat /sys/fs/cgroup/cpu/cpu.shares

512

稍后启动多个容器进行测试，单个容器看不出来

1. 了解cpu周期控制
2. docker提供了--cpu-period(周期)、--cpu-quota两个参数控制容器可以分配到的CPU时钟周期。

· cpu-period是用来指定容器对CPU的使用要在多长时间内做一次重新分配。

//指定周期

· cpu-quota是用来指定在这个周期内，最多可以有多少时间片段用来跑这个容器。

//指定在这个周期中使用多少时间片

· 跟--cpu-shares不同，--cpu-period和--cpu-quota是指定一个绝对值，而且没有弹性在里面，容器对CPU资源的使用绝对不会超过配置的值。

· cpu-period和cpu-quota的单位为微秒(μs)，cpu-period的最小值为1000微秒，最大值为1秒(106μs)，默认值为0.1s(100000μs)。cpu-quota的默认值为-1，表示不做控制。

1s=1000ms,1ms=1000μs

例 1：如果容器进程需要每 1 秒使用单个 CPU 的 0.2 秒时间，可以将 cpu-period 设置为 1000000（即 1 秒），cpu-quota 设置为 200000（0.2 秒）。

[root@docker01 ~]# docker run -it --cpu-period 1000000 --cpu-quota 200000 centos /bin/bash

[root@10905b569ec2 /]# cat /sys/fs/cgroup/cpu/cpu.cfs\_period\_us

1000000

[root@10905b569ec2 /]# cat /sys/fs/cgroup/cpu/cpu.cfs\_quota\_us

200000

1. CPU core核心控制
2. 参数：--cpuset可以绑定CPU

对多核 CPU 的服务器，docker 还可以控制容器运行限定使用哪些 cpu 内核和内存节点，即使用--cpuset-cpus 和--cpuset-mems 参数。对具有 NUMA 拓扑（具有多 CPU、多内存节点）的服务器尤其有用，可以对需要高性能计算的容器迚行性能最优的配置。如果服务器只有一个内存节点，则cpuset-mems 的配置基本上不会有明显效果。

1. 扩展：

· 服务器架构一般分：SMP、NUMA、MPP体系结构

· 从系统结构来看，目前的商用服务器大体可以分为三类：

1. 即对称多处理器结构(SMP：Symmetric Multi-Processor)例：x86服务器，双路服务器。主板上有两个物理cpu
2. 非一致存储访问结构：(NUMA：Non-Uniform Memory Access)例：IBM小型机 、pSeries690
3. 海量并行处理结构(MPP：Massive ParallelProcessing)：例：大型机

做运维有好处：软、硬、网络

学完Linux学开发，再往技术主管走

1. taskset命令

· taskset设定

taskset 设定 cpu 亲和力，taskset 能够将一个或多个进程绑定到一个或多个处理器上运行。

· 参数：

[root@docker01 ~]# ps aux|grep sshd

root 981 0.0 0.2 105996 4084 ? Ss 22:32 0:00 /usr/sbin/sshd -D

[root@docker01 ~]# taskset -cp 0 981

pid 981's current affinity list: 0

/\* 案例中cpu设置为1,2，因为建立虚拟机时只给了1个cpu所以这里设置为cpu0

将981进程绑定在cpu1和2上

affinity [əˈfɪnəti] 密切关系

\*/

· 例2：查看ID为1的进程在哪个cpu上运行

[root@xuegod63 ~]# taskset -cp 1

pid 1's current affinity list: 0-3

RHEL中进程ID为1的进程是：systemd;RHEL6中进程ID为1的进程是：init

· 例 3：设置 nginx cpu 亲和力

1. 在 conf/nginx.conf 中，有如下一行：

worker\_processes 1;

1. 这是用来配置nginx启动几个工作进程的，默认为 1。而 nginx还支持一个名为worker\_cpu\_affinity的配置项，也就是说，nginx 可以为每个工作进程绑定 CPU。如下配置：

worker\_processes 4;

worker\_cpu\_affinity 0001 0010 0100 1000;

这里 0001 0010 0100 1000 是掩码，分别代表第 1、2、3、4 颗 cpu 核心。

1. 重启 nginx 后，4 个工作进程就可以各自用各自的 CPU 了。

· 例 4：物理机一共有 16 个核心，创建的容器只能用 0、1、2 这三个核心

[root@docker01 ~]# docker run -it --name cpu1 --cpuset-cpus 0 centos

//这里可以设置cpu0-2

[root@207a2fdc0dbb /]# cat /sys/fs/cgroup/cpuset/cpuset.cpus

[root@207a2fdc0dbb /]# cat /sys/fs/cgroup/cpuset/cpuset.cpus

0

[root@docker01 ~]# docker exec cpu1 taskset -cp 1

pid 1's current affinity list: 0

[root@207a2fdc0dbb /]# taskset -cp 1

pid 1's current affinity list: 0

1. CPU配额控制参数的混合使用
2. cpu-shares控制只发生在容器竞争同一个cpu的时间片时，如果通过cpuset-cpus指定容器A使用cpu0，容器B用cpu1，在主机上只有这两个容器使用对应内核的情况时，他们各自占用全部的内核资源，cpu-shares没有明显的效果。
3. cpu-period、cpu-quota这两个参数一般联合使用，在单核情况或者通过cpuset-cpus强制使容器使用一个cpu内核的情况下，即使cpu-quota超过cpu-period，也不会使容器使用更多的CPU资源。
4. cpuset-cpus、cpuset-mems只在多核、多内存节点上的服务器上有效，并且必须与实际的物理配置匹配，否则也无法达到资源控制的目的。

· 例 1：测试 cpuset-cpus 和 cpu-shares 混合使用运行效果，就需要一个压缩力测试工具 stress 来让容器实例把 cpu 跑满。

如何把 cpu 跑满？ 如何把 4 核心的 cpu 中第一和第三核心跑满？

1. 扩展：stress 命令

概述：linux 系统压力测试软件 Stress 。 stress 可以测试 Linux 系统 cpu/menory/IO/disk 的负载

下载页：

<http://people.seas.harvard.edu/~apw/stress/>

1. 方法 1：

[root@xuegod63 ~]# yum install -y epel-release

[root@xuegod63 ~]# yum install stress -y

1. 方法2：

[root@xuegod63 ~]# yum install -y epel-release

[root@xuegod63 ~]# yum install stress -y

[root@docker01 ~]# mkdir -p software/stress

[root@docker01 ~]# cd software/stress/

[root@docker01 ~]# yum -y install wget

[root@docker01 ~]# wget http://people.seas.harvard.edu/~apw/stress/stress-1.0.4.tar.gz

[root@docker01 ~]# tar xvf stress-1.0.4.tar.gz

[root@docker01 ~]# cd stress-1.0.4/

[root@docker01 ~]# yum -y install gcc

[root@docker01 ~]# make -j4 && make install

stress 参数解释

-? 显示帮助信息

-v 显示版本号

-q 不显示运行信息

-n 显示已完成的指令情况

-t --timeout N 指定运行 N 秒后停止

--backoff N 等待 N 微妙后开始运行

-c 产生 n 个进程 每个进程都反复不停的计算随机数的平方根，测试 cpu

-i 产生 n 个进程 每个进程反复调用 sync()，sync()用于将内存上的内容写到硬盘上，测试 io

-m --vm n 产生 n 个进程,每个进程不断调用内存分配 malloc 和内存释放 free 函数 ，测试内存

--vm-bytes B 指定 malloc 时内存的字节数 （默认 256MB）

--vm-hang N 指定在 free 栈的秒数

-d --hadd n 产生 n 个执行 write 和 unlink 函数的进程

-hadd-bytes B 指定写的字节数

--hadd-noclean 不 unlink

注：时间单位可以为秒 s，分 m，小时 h，天 d，年 y，文件大小单位可以为 K，M，G

· 例1：产生两个cpu进程，2个io进程，20s后停止运行

[root@docker01 ~]# stress -c 2 -i 2 --verbose --timeout 20s

//如果执行时间为分钟，改20s为1m

· 例2：测试 cpuset-cpus 和 cpu-shares 混合使用运行效果，就需要一个压缩力测试工具 stress 来让容器实例把cpu跑满。当跑满后，会不会去其他cpu上运行。如果没有在其他cpu上运行，说明cgroup资源限制成功。

· 实例：创建两个容器实例:docker10 和 docker20。 让 docker10 和 docker20 只运行在 cpu0和 cpu1 上，最终测试一下 docker10 和 docker20 使用 cpu 的百分比。

1. 创建两个容器实例：

[root@docker01 ~]# docker run -idt --name docker10 --cpuset-cpus 0,1 --cpu-shares 512 centos /bin/bash

[root@docker01 ~]# docker run -idt --name docker20 --cpuset-cpus 0,1 --cpu-shares 1024 centos /bin/bash

1. 进入docker10，使用stress测试进程是不是只在cpu0,1上运行：

[root@docker01 ~]# docker exec docker10 /bin/bash

[root@d1a431815090 /]# yum install -y epel-release #安装 epel 扩展源

[root@d1a431815090 /]# yum install stress -y

1. 在物理机上使用top按1快捷键查看每个cpu的使用情况

cpu0和cpu1跑满

1. 进入docker20，使用stress测试进程是不是只在cpu0,1上运行，且docker20上运行的stress使用cpu百分比是docker10的两倍

[root@docker01 ~]# docker exec -it docker20 /bin/bash

[root@d1a431815090 /]# yum install -y epel-release #安装 epel 扩展源

[root@d1a431815090 /]# yum install stress -y

[root@f24e75bca5c0 /]# stree -c 2 -v -t 10m

两个容器只在cpu0,1上运行且docker20是docker10使用cpu的两倍，说明--cpu-shares限制资源成功

1. 当容器命令运行结束后，自动删除容器
2. rm命令

· 简介

--rm 当容器命令运行结束后，自动删除容器，自动释放资源

· 用处

- 当容器命令运行结束后，自动删除容器

· 应用场景：在某些环境下，可能需要大量的新建docker实例，然后仅仅运行几秒钟，就彻底删除。如运行单元测试或弹性云计算。

- 例：阿里云要模拟双11的压力，需要快速创建1万台虚拟机，运行stress，运行1个小时，1小时候自动删除;或者运行ab命令，拼命访问tmall.com首页，运行1个小时，1小时后自动删除。

[root@docker01 ~]# docker run -it --rm --name mk centos sleep 5

[root@docker01 ~]# docker ps | grep mk

34f31b11245f centos "sleep 5" 1 seconds ago Up Less than a second mk

执行结束后

[root@docker01 ~]# docker ps | grep mk

//容器mk自动删除了

1. docker容器资源配额控制——内存
2. 内存
3. Docker提供参数-m，--memory=""限制容器的内存使用量

例1：允许容器使用的内存上限为128M：

[root@docker01 ~]# docker run -it -m 128m centos

[root@035a4656cb27 /]# cat /sys/fs/cgroup/memory/memory.limit\_in\_bytes

134217728

注意：也可以使用stress进行测试

echo 3 > /proc/sys/vm/drop\_caches

1. docker容器资源配额控制——IO
2. IO

[root@docker01 ~]# docker run --help | grep write-b

--device-write-bps value Limit write rate (bytes per second) to a device (default [])

//限制此设备上的写速度，单位可以使kb,mb或者gb

[root@docker01 ~]# docker run --help | grep read-b

--device-read-bps value Limit read rate (bytes per second) from a device (default [])

//限制此设备上的读速度，单位可以使kb,mb或者gb

1. 为什么阿里云平台上普通云盘的IO为：1000IOPS，为什么这么小
2. 原因

· 1存储给2000台云主机使用，需要控制一下，防止某台云主机吃光你的磁盘I/O资源

1. 案例

· 防止某个Docker容器吃光你的磁盘I/O资源

例1：容器对硬盘的最高写入速度设定为1MB/s

--device参数：将主机设备添加到容器

[root@docker01 ~]# docker run -it -v /var/webroot/:/var/www/html --device /dev/sda:/dev/sda --device-write-bps /dev/sda:1mb centos /bin/bash

[root@d727737ac6bc /]# time dd if=/dev/sda of=/var/www/html/test02.out bs=1M count=10

10+0 records in

10+0 records out

10485760 bytes (10 MB) copied, 0.0052847 s, 2.0 GB/s

real 0m0.006s

user 0m0.000s

sys 0m0.006s

[root@d727737ac6bc /]# time dd if=/dev/sda of=/test01.out bs=1M count=50 oflag=direct,nonblock

50+0 records in

50+0 records out

52428800 bytes (52 MB) copied, 0.061107 s, 858 MB/s

real 0m0.062s

user 0m0.000s

sys 0m0.020s

[root@d727737ac6bc /]# time dd if=/dev/sda of=/var/www/html/test01.out bs=1M count=50 oflag=direct,nonblock

50+0 records in

50+0 records out

52428800 bytes (52 MB) copied, 50.0011 s, 1.0 MB/s

real 0m50.002s

user 0m0.000s

sys 0m0.018s

注：dd参数

direct：读写数据采用直接IO方式;

nonblock：读写数据采用非阻塞IO方式

可以看到，只有在满足条件{ (1) 目录为/var/www/html/ ; (2) oflag=direct,nonblock跳过内存 ; (3) 两者同时满足 ; }的情况下读写速度才是限定的1M/s

例2：在ubuntu系统中测试

1. 虚拟机第一次启动时

[root@docker01 ~]# docker run -it --rm --device /dev/sda:/dev/sda --device-read-bps /dev/sda:1mb --device-write-bps /dev/sda:1mb ubuntu bash

root@363a19654857:/# time dd if=/dev/sda of=/test.out bs=1M count=30 oflag=direct,nonblock

30+0 records in

30+0 records out

31457280 bytes (31 MB, 30 MiB) copied, 29.5047 s, 1.1 MB/s

real 0m29.506s

user 0m0.000s

sys 0m0.147s

1. 生成一个文件之后再次测试

root@363a19654857:/# time dd if=/dev/sda of=/test01.out bs=1M count=30 oflag=direct,nonblock

30+0 records in

30+0 records out

31457280 bytes (31 MB, 30 MiB) copied, 0.206045 s, 153 MB/s

real 0m0.208s

user 0m0.002s

sys 0m0.009s

1. 使用文件映射之后测试

[root@docker01 ~]# docker run -it -v /var/webroot/:/var/www/html --rm --device /dev/sda:/dev/sda --device-read-bps /dev/sda:1mb --device-write-bps /dev/sda:1mb ubuntu bash

root@34cbd4575753:/# time dd if=/dev/sda of=/var/www/html/test.out bs=1M count=30 oflag=direct,nonblock

30+0 records in

30+0 records out

31457280 bytes (31 MB, 30 MiB) copied, 30.0023 s, 1.0 MB/s

/\* 可以看出，ubuntu系统中也需要满足以上条件 \*/

1. docker数据映射
2. docker数据映射
3. docker用来做计算，存储外挂
4. 语法

docker run -itd -v src:/dst centos bash

-v 用来指定挂载目录，冒号：前面的/src为物理机本地目录，后面/dst为容器里的目录

1. 案例

例1：把物理机上的/var/webroot映射到docker实例的/var/www/html

· 好处：这样当docker坏了，数据还在物理机上，再使用apache镜像启动一个docker就可以了，数据不会丢失。

mkdir -p /var/webroot

docker run -it --name web1 -h web1 -v /var/www/html:/var/www/html centos bash

echo "I am a docker!" > /var/www/html/index.html

· 在物理机查看：

[root@docker01 ~]#cat /var/webroot/index.html

Part3 配置docker静态IP地址-配置docker私有仓库

1. 实验环境
2. 一个还原到之前安装了docker的虚拟机快照
3. 创建docker静态化IP
4. Docker的4种网络模式
5. Docker有以下4种网络模式

· host模式

- 使用--net=host指定

· container模式

- 使用--net=container:NAME\_or\_ID指定

· none模式

- 使用--net=none指定

· bridge模式

- 使用--net=bridge指定，默认设置

· 默认选择bridge的情况下，容器启动后会通过DHCP获取一个地址，这可能不是我们想要的，在centos7系统上，docker环境下可以使用pipework脚本对容器分配固定IP(这个IP可以是和物理机相同网段的IP)

/\*

\* docker默认是bridge(--net=bridge)模式，相当于VMware中NAT模式

\* docker环境下可以使用pipework脚本对容器分配固定IP，相当于VMware中桥接模式

\* pipework有个缺陷，容器重启后IP设置会自动消失，需要重新设置

\*/

1. 配置桥接网络
2. 优点和缺点

· 优点

桥接本地物理网络的目的，是为了局域网内用户方便访问docker实例中服务，不需要各种端口映射即可访问服务

· 缺点

这样做违背了docker容器的安全隔离的原则，在工作中酌情选择

1. 创建桥接设备
2. 安装包
3. 把eth0绑定到br0桥设备上

[root@docker02 ~]# cp network/ifcfg-ens33{,.bak}

[root@docker02 ~]# vim ifcfg-ens33

TYPE=Ethernet

PROXY\_METHOD=none

BROWSER\_ONLY=no

BOOTPROTO=dhcp

DEFROUTE=yes

IPV4\_FAILURE\_FATAL=no

NAME=ens33

DEVICE=ens33

ONBOOT=yes

BRIDGE=br0

[root@docker02 ~]# vim ifcfg-br0

DEVICE=br0

NAME=br0

NM\_CONTROLLED=yes

ONBOOT=yes

TYPE=Bridge

BOOTPROTO=none

IPADDR=192.168.1.129

PREFIX=24

GATEWAY=192.168.1.254

DNS1=192.168.1.254

DNS2=114.114.114.114

[root@docker02 ~]# systemctl restart network

[root@docker02 ~]# ifconfig ens33 | head -2

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

ether 00:0c:29:bd:46:da txqueuelen 1000 (Ethernet)

[root@docker02 ~]# ifconfig br0 | head -2

br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.1.129 netmask 255.255.255.0 broadcast 192.168.1.255

1. 下载pipework包

· 地址：

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

· 下载并解压

[root@docker02 ~]# wget <https://github.com/jpetazzo/pipework/archive/master.zip>

[root@docker02 ~]# unzip master.zip

[root@docker02 ~]# ls

anaconda-ks.cfg master.zip network pipework-master

[root@docker02 ~]# cp pipework-master/pipework /usr/local/bin/

1. 使用静态 IP 启动一个 docker 实例
2. 以none模式，使用--net=none启动一个容器，并且开启docker特权模式
3. 特权功能启动

--privileged=true #允许开启特权功能

· 大约在docekr0.6版本以后，privileged被引入docker。使用该参数，container内的root拥有真正的root权限。否则，container内的root只是外部的一个普通用户权限

· 使用privileged启动的容器，可以看到很多host上的设备，并且可以执行mount。甚至允许你在docker容器中启动docker容器。

· 不启用privileged，容器中root用户不能mount

[root@docker02 ~]# docker run -it --privileged 192.168.1.130:5000/centos bash

/\* 开启特权模式可以做什么：

1. 可以看到很多设备文件 ls /dev
2. 可以挂载目录 mount -o bind /etc /opt
3. 可以挂载分区 mount /dev/sda1 /opt

\*/

1. 配置容器网络
2. 启动一个容器实例

[root@docker02 ~]# docker run -itd --net=none --privileged 192.168.1.130:5000/centos bash

1. 尝试挂载

[root@docker02 ~]# docker exec -it 2b bash

[root@2ba622894e43 /]# mount -o bind /etc /opt

1. 配置网络

[root@docker02 ~]# pipework br0 2b 192.168.1.145/24@192.168.1.254

1. 查看配置

[root@2ba622894e43 /]# vi /etc/yum.repos.d/yum.repo

[CentOS]

name=CentOS7\_Base

baseurl=ftp://192.168.1.130/CentOS7

enabled=1

gpgcheck=0

[root@2ba622894e43 /]# yum -y install net-tools

[root@2ba622894e43 /]# ifconfig eth1 | head -2

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.1.145 netmask 255.255.255.0 broadcast 192.168.1.255

1. 总结
2. 创建一个 br0 桥接设备
3. 下载 pipework 包并安装
4. 安装并运行 docker
5. 导入 centos docker 镜像
6. 启动一个 docker 实例 注意加参数： --net=none --privileged=true
7. 使用 pipework 给 docker 实例配置 IP
8. 创建docker私有化仓库
9. 使用官方提供的工具配置私有化仓库
10. 私有仓库介绍

· 有时候使用Docker Hub这样的公共仓库可能不方便(有时候无法访问)，用户可以创建一个本地仓库供私人使用，这里使用官方提供的工具docker-registry来配置私有仓库。

1. 官方工具

· docker-registry是官方提供的工具，可以用于构建私有的镜像仓库

1. 私有仓库的好处
2. 节约带宽
3. 可以自己定制系统
4. docker-registry这个工具是一个镜像，直接下载并使用registry镜像启动docker实例就可以了
5. 有了docekr以后，所有软件不再以office.exe或lrzsz.rpm的形式发布，而是以docker镜像发布。只需要下载docker镜像并运行一个docker实例。
6. 实验环境
7. docker私有仓库地址：192.168.1.130
8. docker服务器地址：192.168.1.129，docker02会使用docker01上的docker私有仓库来pull/push镜像
9. 实验过程
10. 配置docker01为docker私有仓库
11. 不能关闭防火墙，因为docker后期端口转发需要用到firewalld

[root@docker01 ~]# systemctl start firewalld

//开启防火墙，后面需要使用5000端口进行通信

1. 关闭selinux

[root@docker01 ~]# vim /etc/selinux/config

SELINUX=disabled

[root@docker01 ~]# reboot

1. 配置docker01为docker私有仓库服务端
2. 启动docker

[root@docker01 ~]# systemctl start docker

1. 拉取registry'镜像

· 方法1：导入本地镜像

[root@docker01 ~]# docker load -i registry.tar

· 方法2：在线导入镜像(比较慢)

[root@docker01 ~]# docker pull registry

/\* 出现error ... EOF是因为访问不了国外的网址导致的 \*/

1. 查看registry镜像

[root@docker01 ~]# docker images

扩展：.io域名

.io是[british indian ocean territory]英属印度洋领地的简写

1. 从Docker Hub上拉取一个镜像测试

· 方法1：从本地导入

上传busybox镜像到Linux系统上，作为测试镜像

[root@docker01 ~]# docker load -i busybox

[root@docker01 ~]# docker images

· busybos概述

- busybox是一个集成了一百多个最常用Linux命令和工具的软件。susybox包含了一些简单的工具，例如ls，cat和echo等，还包含了一些更大、更复杂的工具，例如grep、find、mount以及telnet。有些人将busybox成为Linux工具里的瑞士军刀。简单来说，busybox像一个大工具箱，它集成压缩了Linux的许多工具和命令，也包含了Android系统的自带shell。

1. 给docker镜像打标签(创建一个硬链接)

· docker tag命令

- 语法：docker tag 镜像名:标签 私有仓库地址:端口/镜像名:标签

[root@docker01 ~]# docker tag busybox:latest 192.168.1.130:5000/busybox:latest

[root@docker01 ~]# docker images | grep 192.168.1.130:5000

1. 修改docker配置文件，指定私有仓库url

· registry 记录，登记，注册

· 需要安装docker-common-1.12.6-11.el7.centos.x86\_64包，否则没有配置文件

[root@xuegod63 ~]# rpm -qf /etc/sysconfig/docker

docker-common-1.12.6-11.el7.centos.x86\_64

· 修改配置

- 方法1：

[root@docker01 ~]# vim /etc/sysconfig/docker

//第四行后面加上insecure-registry 192.168.1.130:5000

OPTIONS='--selinux-enabled --log-driver=journald --signature-verification=false --insecure-registry 192.168.1.130:5000'

--insecure-registry不安全的注册。即通信使用http协议，如果使用安全的通信，需要使用https

- 方法2：

[root@docker02 ~]# vim /etc/docker/daemon.json

{

"insecure-registries":["192.168.1.130:5000"]

}

1. 使用registry镜像搭建一个私有仓库

· 使用registry镜像搭建一个私有仓库。registry镜像中已经把搭建私有库程序安装好了，只需要使用registry镜像运行一个docker实例就可以了。

· registry运行的实例的默认存储路径为/var/lib/registry，是个临时目录，容器关闭文件丢失，所以使用-v参数，制定本地持久的路径。

[root@docker01 ~]# docker run -d -p 5000:5000 -v /opt/registry:/var/lib/registry registry:latest

[root@docker01 ~]# ss -antulp | grep 5000

tcp LISTEN 0 128 :::5000 :::\* users:(("docker-proxy-cu",pid=4016,fd=4))

/\* 搭建成功，私有仓库已经启动 \*/

1. 上传镜像

[root@docker01 ~]# docker push 192.168.1.130:5000/busybox:latest

1. 客户端测试
2. 修改docker配置文件，指定docker镜像加速节点为私有仓库地址

[root@docker01 ~]# vim /etc/sysconfig/docker

//第四行后面加上insecure-registry 192.168.1.130:5000

OPTIONS='--selinux-enabled --log-driver=journald --signature-verification=false --insecure-registry 192.168.1.130:5000'

1. 下载刚才上传的镜像

[root@docker02 ~]# docker pull 192.168.1.130:5000/busybox:latest

[root@docker02 ~]# docker images

1. 运行镜像

[root@docker02 ~]# docker run --rm 192.168.1.130:5000/busybox echo "hello world"

1. 使用阿里云私有仓库存储自己家的docker镜像
2. 登录阿里云，选择创建我的容器镜像
3. 仓库名字，命名空间，仓库状态，性质，权限，地址
4. 使用Docker加速器：修改daemon文件/etc/docker/daemon来使用加速器

{

"registry-mirrors":["https://..."]

}

1. 镜像的使用

docker log in --username=支付宝账号 registry.cn

密码时开通服务时设置的密码