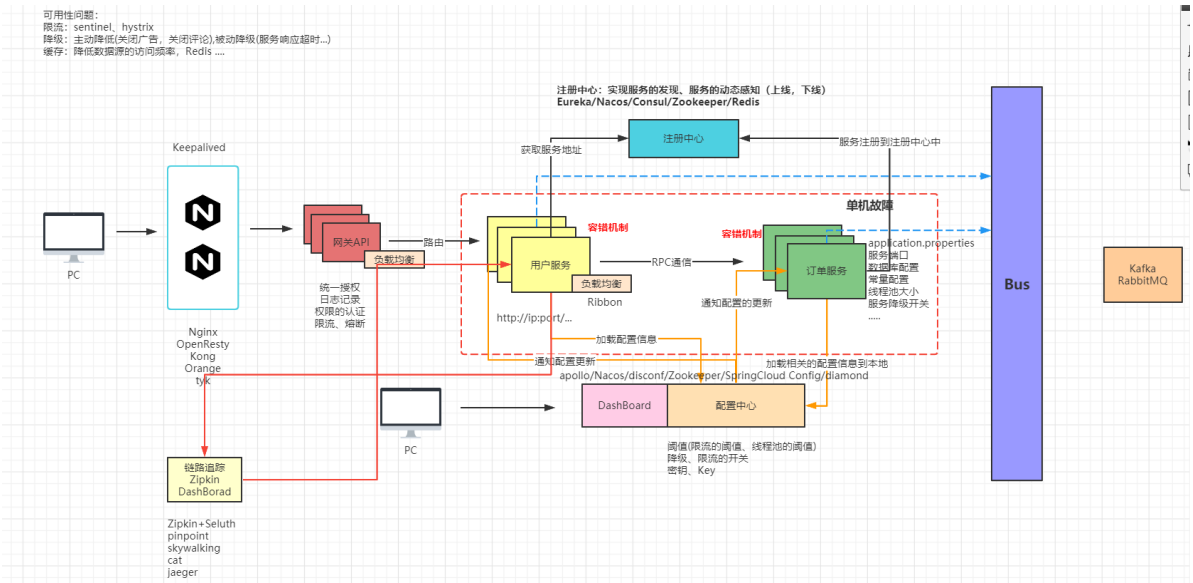
Docker

一、Docker基础篇

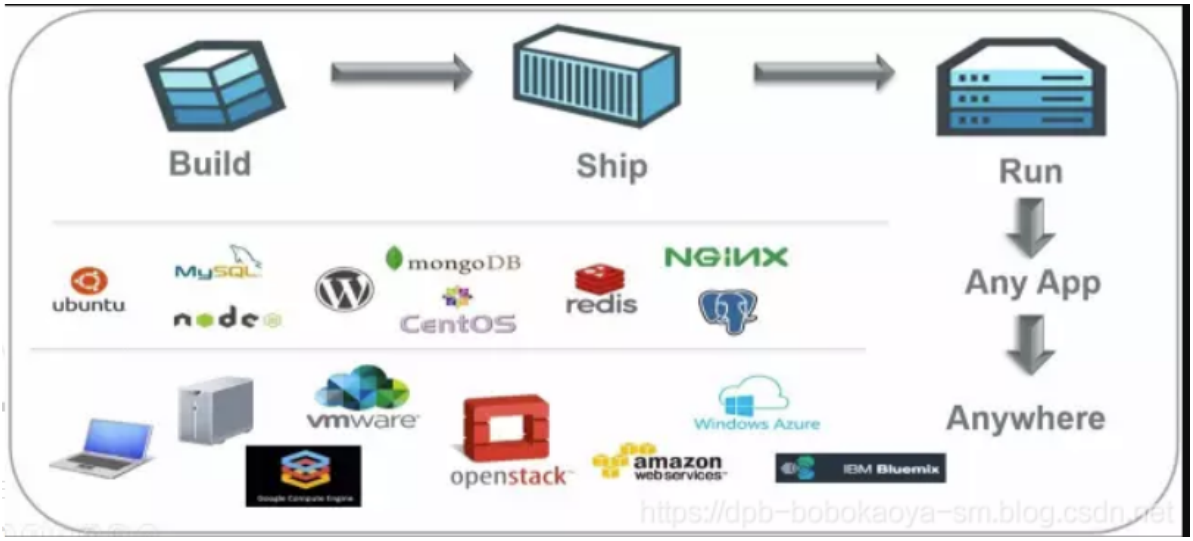
1.Docker的介绍与安装



1.1 什么是Docker

Docker是基于**Go**语言实现的云开源项目。

Docker的主要目标是Build，Ship and Run Any App,Anywhere，也就是通过对应用组件的封装、分发、部署、运行等生命周期的管理，使用户的APP（可以是一个WEB应用或数据库应用等等）及其运行环境能够做到一次封装，到处运行。



Linux 容器技术的出现就解决了这样一个问题，而 Docker 就是在它的基础上发展过来的。将应用运行在 Docker 容器上面，而 Docker 容器在任何操作系统上都是一致的，这就实现了跨平台、跨服务器。只需要一次配置好环境，换到别的机子上就可以一键部署好，大大简化了操作

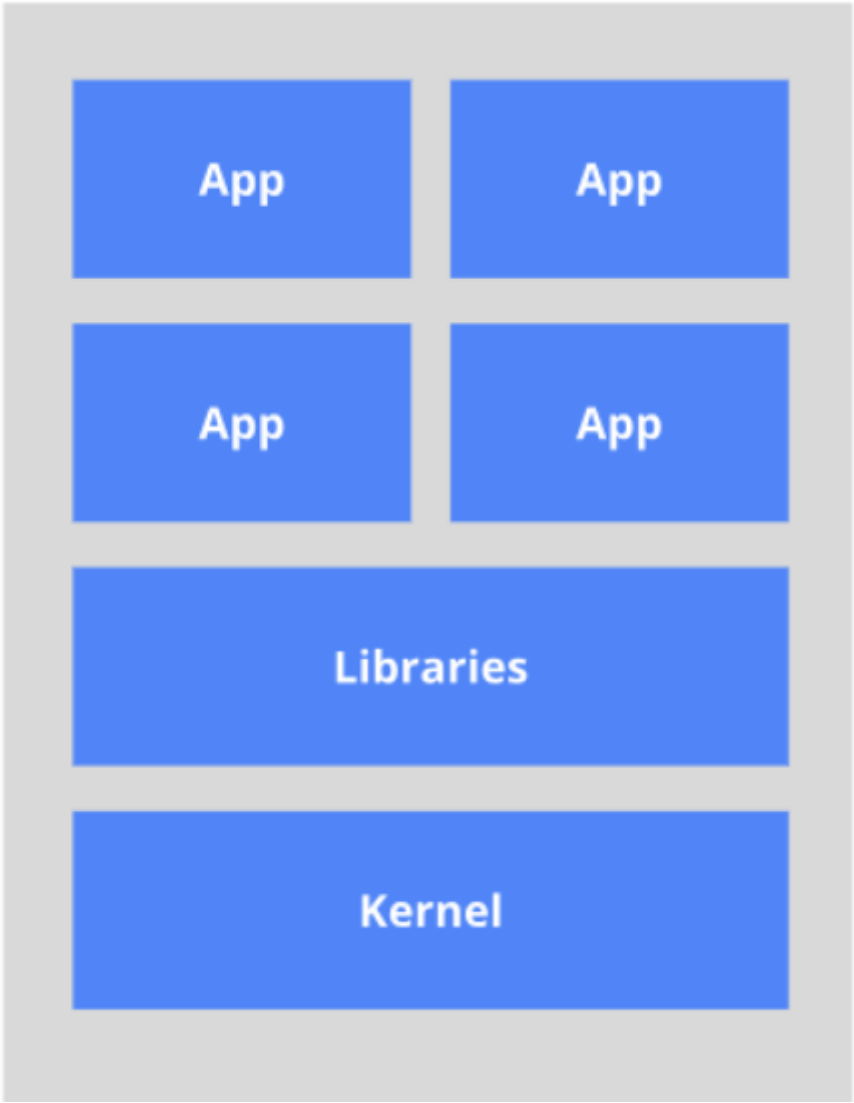
> 解决了运行环境和配置问题软件容器，方便做持续集成并有助于整体发布的容器虚拟化技术

1.2 Docker能干什么?

1.2.1以前的虚拟化技术

虚拟机（virtual machine）就是带环境安装的一种解决方案。

它可以在一种操作系统里面运行另一种操作系统，比如在Windows 系统里面运行Linux 系统。应用程序对此毫无感知，因为虚拟机看上去跟真实系统一模一样，而对于底层系统来说，虚拟机就是一个普通文件，不需要了就删掉，对其他部分毫无影响。这类虚拟机完美的运行了另一套系统，能够使应用程序，操作系统和硬件三者之间的逻辑不变。



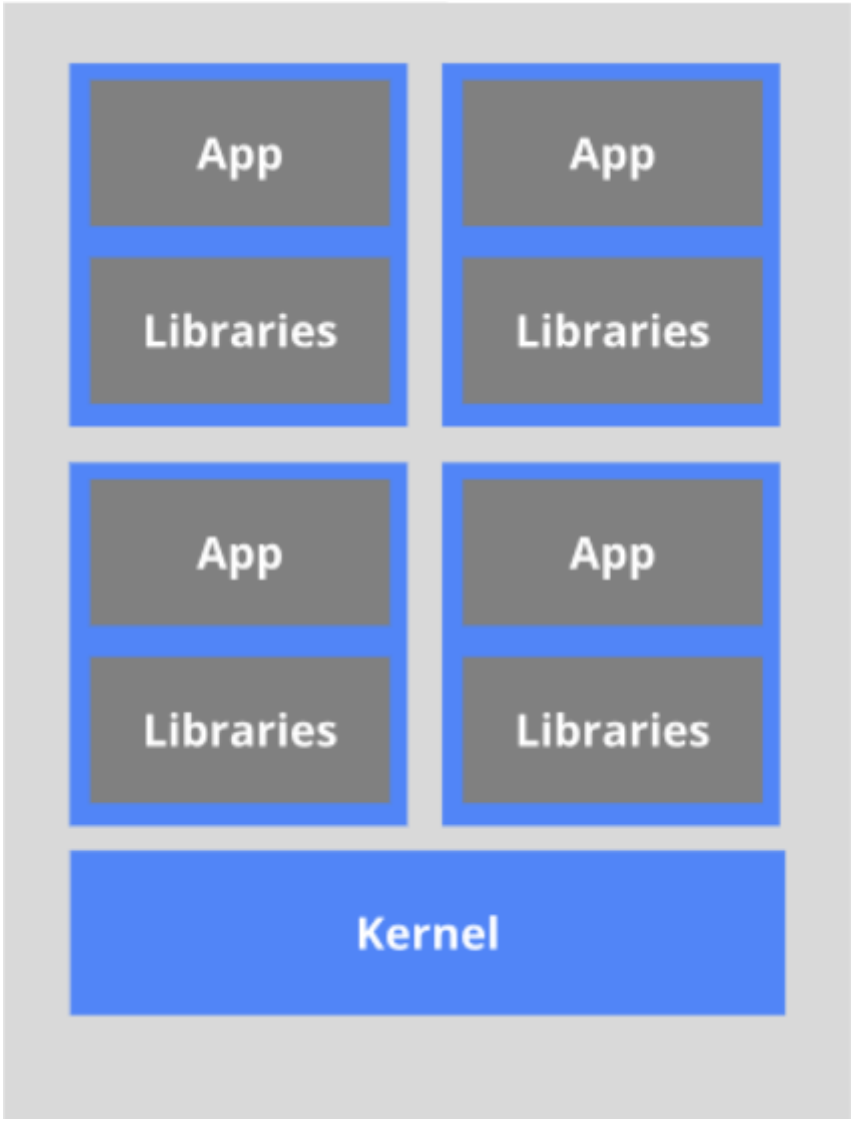
虚拟机的缺点:

1. 资源占用多
2. 冗余步骤多
3. 启动慢

1.2.2 容器虚拟化技术

由于前面虚拟机存在这些缺点，Linux 发展出了另一种虚拟化技术：Linux 容器（Linux Containers，缩写为 LXC）。

Linux 容器不是模拟一个完整的操作系统，而是对进程进行隔离。有了容器，就可以将软件运行所需的所有资源打包到一个隔离的容器中。容器与虚拟机不同，不需要捆绑一整套操作系统，只需要软件工作所需的库资源和设置。系统因此而变得高效轻量并保证部署在任何环境中的软件都能始终如一地运行。



比较了 Docker 和传统虚拟化方式的不同之处：

1. 传统虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统，在该系统上再运行所需应用进程；
2. 而容器内的应用进程直接运行于宿主的内核，容器内没有自己的内核，而且也没有进行硬件虚拟。因此容器要比传统虚拟机更为轻便。
3. 每个容器之间互相隔离，每个容器有自己的文件系统 ，容器之间进程不会相互影响，能区分计算资源。

1.2.3 实际的运行

Docker作为开发人员需要掌握，作为`运维人员`必须掌握。
一次构建，随处运行

1. 更快速的应用交付和部署
2. 更便捷的升级和扩缩容
3. 更简单的系统运维
4. 更高效的计算资源利用

1.3 相关资源

官网: http://www.docker.com

仓库: https://hub.docker.com

1.4 Docker安装

1.4.1 Docker安装的前提环境

CentOS Docker 安装
Docker支持以下的CentOS版本：
CentOS 7 (64-bit) 8
CentOS 6.5 (64-bit) 或更高的版本

前提条件
目前，CentOS 仅发行版本中的内核支持 Docker。
Docker 运行在 CentOS 7 上，要求系统为64位、系统内核版本为 3.10 以上。
Docker 运行在 CentOS-6.5 或更高的版本的 CentOS 上，要求系统为64位、系统内核版本为 2.6.32-431 或者更高版本。
查看自己的内核
uname命令用于打印当前系统相关信息（内核版本号、硬件架构、主机名称和操作系统类型等）。

1.4.2 Vagrant虚拟机环境

Docker安装在虚拟机上，我们可以通过VMWare来安装，但是通过VMWare安装大家经常会碰到网络ip连接问题，为了减少额外的环境因素影响，Docker内容的讲解我们会通过VirtualBox结合Vagrant来安装虚拟机

VirtualBox官网：https://www.virtualbox.org/

Vagrant官网：https://www.vagrantup.com/

Vagrant镜像仓库：https://app.vagrantup.com/boxes/search

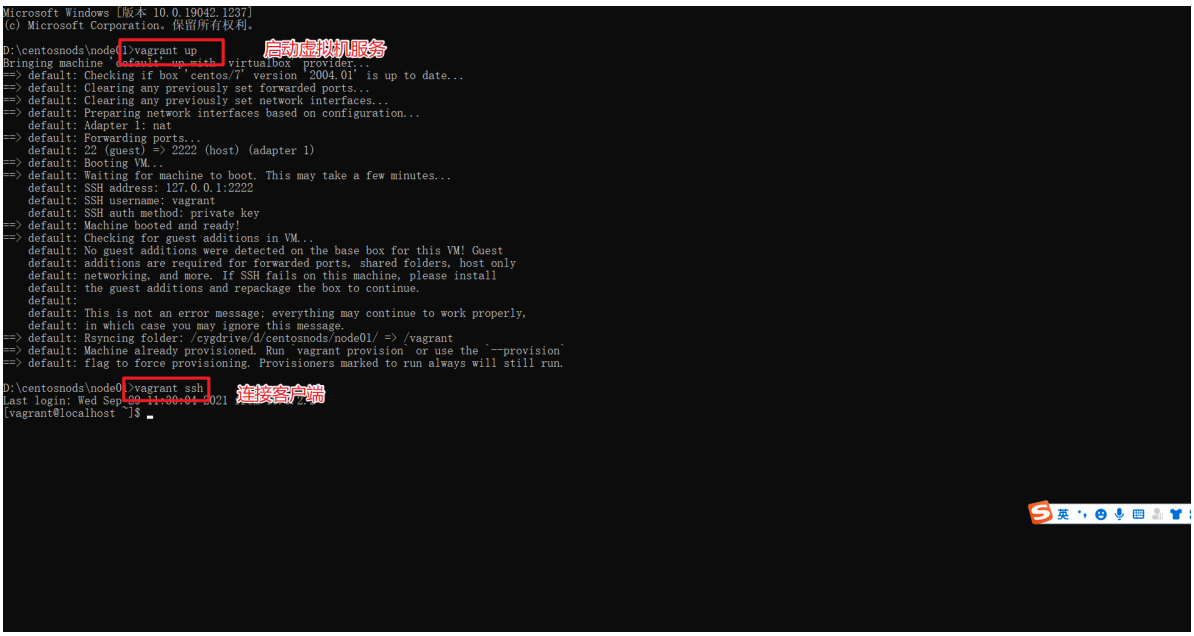
安装VirtualBox和Vagrant，傻瓜式安装。安装完成后需要重启计算机。

在cmd命令窗口输入`vagrant`命令弹出如下内容表示`vagrant`安装成功

```
C:\Users\dpb>vagrant
Usage: vagrant [options] <command> [<args>]

    -v, --version                    Print the version and exit.
    -h, --help                       Print this help.

Common commands:
     box             manages boxes: installation, removal, etc.
     cloud           manages everything related to Vagrant Cloud
     destroy         stops and deletes all traces of the vagrant machine
     global-status   outputs status Vagrant environments for this user
     halt            stops the vagrant machine
     help            shows the help for a subcommand
     init            initializes a new Vagrant environment by creating a Vagrantfile
     login
     package         packages a running vagrant environment into a box
     plugin          manages plugins: install, uninstall, update, etc.
     port            displays information about guest port mappings
     powershell      connects to machine via powershell remoting
     provision       provisions the vagrant machine
     push            deploys code in this environment to a configured destination
     rdp             connects to machine via RDP
     reload          restarts vagrant machine, loads new Vagrantfile configuration
     resume          resume a suspended vagrant machine
     snapshot        manages snapshots: saving, restoring, etc.
     ssh             connects to machine via SSH
     ssh-config      outputs OpenSSH valid configuration to connect to the machine
     status          outputs status of the vagrant machine
     suspend         suspends the machine
     up              starts and provisions the vagrant environment
     upload          upload to machine via communicator
     validate        validates the Vagrantfile
     version         prints current and latest Vagrant version
     winrm           executes commands on a machine via WinRM
     winrm-config    outputs WinRM configuration to connect to the machine

For help on any individual command run `vagrant COMMAND -h`

Additional subcommands are available, but are either more advanced
or not commonly used. To see all subcommands, run the command
`vagrant list-commands`.
```

通过Vagrant安装虚拟机：

- 创建一个空的目录，cmd切换到该目录中，然后执行`vagrant init centos/7`会创建Vagrantfile文件
- 执行`vagrant up`第一次执行的时候会远程下相关的镜像文件，并启动虚拟机。
- 连接虚拟机通过`vagrant ssh`连接，默认的账号密码是： vagrant vagrant



网络配置：

找到对应的Vagrantfile文件

```
# Disable automatic box update checking. If you disable this, then
# boxes will only be checked for updates when the user runs
# `vagrant box outdated`. This is not recommended.
# config.vm.box_check_update = false

# Create a forwarded port mapping which allows access to a specific port
# within the machine from a port on the host machine. In the example below,
# accessing "localhost:8080" will access port 80 on the guest machine.
# NOTE: This will enable public access to the opened port
# config.vm.network "forwarded_port", guest: 80, host: 8080

# Create a forwarded port mapping which allows access to a specific port
# within the machine from a port on the host machine and only allow access
# via 127.0.0.1 to disable public access
# config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip: "127.0.0.1"

# Create a private network, which allows host-only access to the machine
# using a specific IP.
  config.vm.network "private_network", ip: "192.168.33.10"

# Create a public network, which generally matched to bridged network.
# Bridged networks make the machine appear as another physical device on
# your network.
# config.vm.network "public_network"

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.
# config.vm.synced_folder "../data", "/vagrant_data"

# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.
# Example for VirtualBox:
#
# config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
#
#   # Customize the amount of memory on the VM:
#   vb.memory = "1024"
# end
#
```

放开这段代码 ip就是我们自己要配置的ip地址，
注意网段要和主机分配的一致

查看当前主机给虚拟机分配的网关的网段：

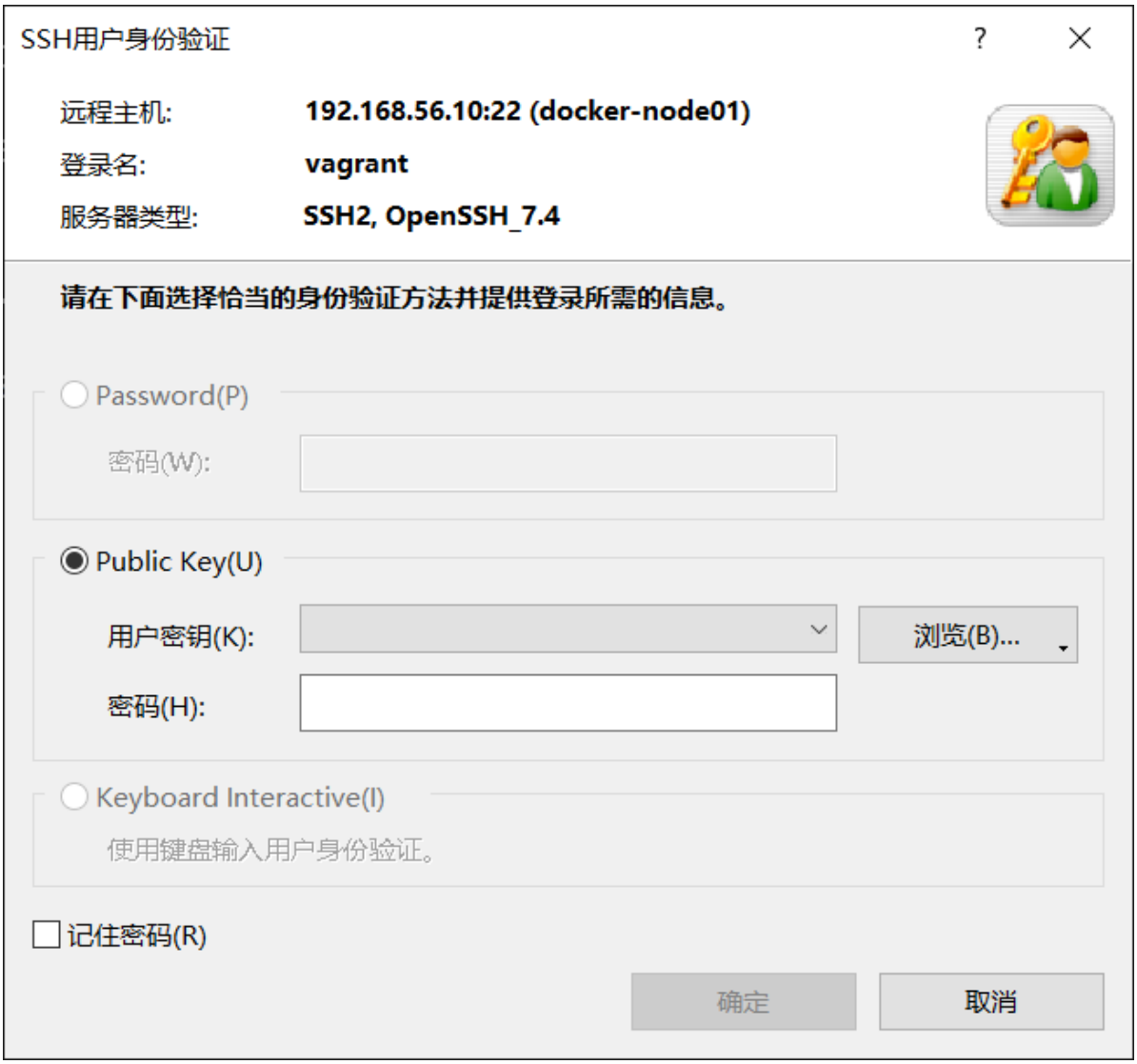
```
Microsoft Windows [版本 10.0.19042.1237]
(c) Microsoft Corporation。保留所有权利。

C:\Users\dpb>ipconfig

Windows IP 配置


以太网适配器 以太网:

   连接特定的 DNS 后缀 . . . . . . . :
   本地链接 IPv6 地址. . . . . . . . : fe80::e9dd:c0df:abbe:8b3f%21
   IPv4 地址 . . . . . . . . . . . . : 192.168.0.102
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 192.168.0.1

以太网适配器 VirtualBox Host-Only Network:

   连接特定的 DNS 后缀 . . . . . . . :
   本地链接 IPv6 地址. . . . . . . . : fe80::bd90:fe22:d068:649c%4
   IPv4 地址 . . . . . . . . . . . . : 192.168.56.1
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . :

无线局域网适配器 WLAN:

   媒体状态 . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :

无线局域网适配器 本地连接* 1:

   媒体状态 . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :

无线局域网适配器 本地连接* 2:

   媒体状态 . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :

以太网适配器 VMware Network Adapter VMnet1:

   连接特定的 DNS 后缀 . . . . . . . :
   本地链接 IPv6 地址. . . . . . . . : fe80::8565:ea36:23ea:b69f%7
   IPv4 地址 . . . . . . . . . . . . : 192.168.226.1
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . :

以太网适配器 VMware Network Adapter VMnet8:

   连接特定的 DNS 后缀 . . . . . . . :
```

网段是 56

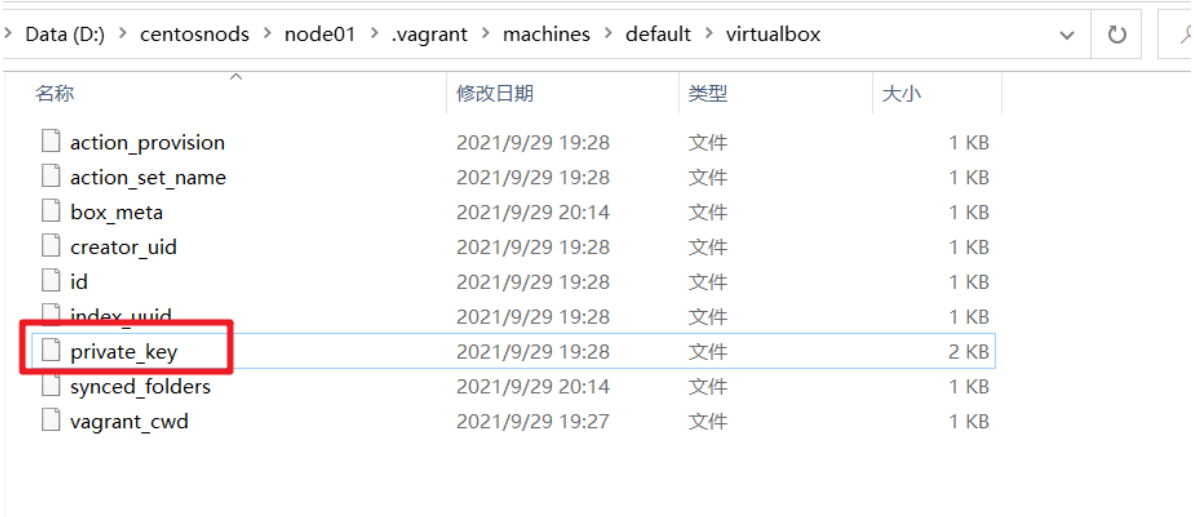
所以修改后的ip为：192.168.56.10

```
# Disable automatic box update checking. If you disable this, then
# boxes will only be checked for updates when the user runs
# `vagrant box outdated`. This is not recommended.
# config.vm.box_check_update = false

# Create a forwarded port mapping which allows access to a specific port
# within the machine from a port on the host machine. In the example below,
# accessing "localhost:8080" will access port 80 on the guest machine.
# NOTE: This will enable public access to the opened port
# config.vm.network "forwarded_port", guest: 80, host: 8080

# Create a forwarded port mapping which allows access to a specific port
# within the machine from a port on the host machine and only allow access
# via 127.0.0.1 to disable public access
# config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip: "127.0.0.1"

# Create a private network, which allows host-only access to the machine
# using a specific IP.
  config.vm.network "private_network", ip: "192.168.56.10"

# Create a public network, which generally matched to bridged network.
# Bridged networks make the machine appear as another physical device on
# your network.
# config.vm.network "public_network"

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.
# config.vm.synced_folder "../data", "/vagrant_data"

# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.
# Example for VirtualBox:
#
# config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
#
#   # Customize the amount of memory on the VM:
#   vb.memory = "1024"
# end
#
```

重启测试：需要提供私钥



私钥地址：



1.4.3 Docker安装

参考官方文档安装：https://docs.docker.com/engine/install/centos/

卸载原有的环境：

```
sudo yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-engine
```

安装对应的依赖环境和镜像地址

```
sudo yum install -y yum-utils
sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
```

安装过慢设置镜像

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

直接安装docker CE

```
sudo yum install -y docker-ce docker-ce-cli containerd.io
```

```
 docker-ce                       x86_64         3:20.10.8-3.el7                 docker-ce-stable          23 M
 docker-ce-cli                   x86_64         1:20.10.8-3.el7                 docker-ce-stable          29 M
Installing for dependencies:
 audit-libs-python               x86_64         2.8.5-4.el7                     base                      76 k
 checkpolicy                     x86_64         2.5-8.el7                       base                     295 k
 container-selinux               noarch         2:2.119.2-1.911c772.el7_8       extras                    40 k
 docker-ce-rootless-extras       x86_64         20.10.8-3.el7                   docker-ce-stable         8.0 M
 docker-scan-plugin              x86_64         0.8.0-3.el7                     docker-ce-stable         4.2 M
 fuse-overlayfs                  x86_64         0.7.2-6.el7_8                   extras                    54 k
 fuse3-libs                      x86_64         3.6.1-4.el7                     extras                    82 k
 libcgroup                       x86_64         0.41-21.el7                     base                      66 k
 libsemanage-python              x86_64         2.5-14.el7                      base                     113 k
 policycoreutils-python          x86_64         2.5-34.el7                      base                     457 k
 python-IPy                      noarch         0.75-6.el7                      base                      32 k
 setools-libs                    x86_64         3.3.8-4.el7                     base                     620 k
 slirp4netns                     x86_64         0.4.3-4.el7_8                   extras                    81 k

Transaction Summary
================================================================================================================
Install  3 Packages (+13 Dependent packages)

Total download size: 96 M
Installed size: 386 M
Is this ok [y/d/N]: y
Downloading packages:
(1/16): container-selinux-2.119.2-1.911c772.el7_8.noarch.rpm                              |  40 kB  00:00:00
(2/16): audit-libs-python-2.8.5-4.el7.x86_64.rpm                                          |  76 kB  00:00:00
(3/16): checkpolicy-2.5-8.el7.x86_64.rpm                                                  | 295 kB  00:00:00
(4/16): containerd.io-1.4.9-3.1.el7.x86_64.rp 1% [-                       ]  20 kB/s | 1.6 MB  01:19:34 ETA
```

```
  Verifying  : docker-ce-rootless-extras-20.10.8-3.el7.x86_64                                 3/16
  Verifying  : audit-libs-python-2.8.5-4.el7.x86_64                                           4/16
  Verifying  : python-IPy-0.75-6.el7.noarch                                                   5/16
  Verifying  : docker-scan-plugin-0.8.0-3.el7.x86_64                                          6/16
  Verifying  : libsemanage-python-2.5-14.el7.x86_64                                           7/16
  Verifying  : slirp4netns-0.4.3-4.el7_8.x86_64                                               8/16
  Verifying  : 2:container-selinux-2.119.2-1.911c772.el7_8.noarch                             9/16
  Verifying  : containerd.io-1.4.9-3.1.el7.x86_64                                            10/16
  Verifying  : policycoreutils-python-2.5-34.el7.x86_64                                      11/16
  Verifying  : 1:docker-ce-cli-20.10.8-3.el7.x86_64                                          12/16
  Verifying  : 3:docker-ce-20.10.8-3.el7.x86_64                                              13/16
  Verifying  : setools-libs-3.3.8-4.el7.x86_64                                               14/16
  Verifying  : fuse-overlayfs-0.7.2-6.el7_8.x86_64                                           15/16
  Verifying  : libcgroup-0.41-21.el7.x86_64                                                  16/16

Installed:
  containerd.io.x86_64 0:1.4.9-3.1.el7     docker-ce.x86_64 3:20.10.8-3.el7     docker-ce-cli.x86_64 1:20.10.8-3.el7

Dependency Installed:
  audit-libs-python.x86_64 0:2.8.5-4.el7                 checkpolicy.x86_64 0:2.5-8.el7
  container-selinux.noarch 2:2.119.2-1.911c772.el7_8     docker-ce-rootless-extras.x86_64 0:20.10.8-3.el7
  docker-scan-plugin.x86_64 0:0.8.0-3.el7                fuse-overlayfs.x86_64 0:0.7.2-6.el7_8
  fuse3-libs.x86_64 0:3.6.1-4.el7                        libcgroup.x86_64 0:0.41-21.el7
  libsemanage-python.x86_64 0:2.5-14.el7                 policycoreutils-python.x86_64 0:2.5-34.el7
  python-IPy.noarch 0:0.75-6.el7                         setools-libs.x86_64 0:3.3.8-4.el7
  slirp4netns.x86_64 0:0.4.3-4.el7_8
```

表示安装成功

启动docker服务

```
sudo systemctl start docker
```

查看docker的版本

```
sudo docker version
```

```
[root@localhost ~]# systemctl start docker
[root@localhost ~]# docker version
Client: Docker Engine - Community
 Version:           20.10.8
 API version:       1.41
 Go version:        go1.16.6
 Git commit:        3967b7d
 Built:             Fri Jul 30 19:55:49 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.8
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.6
  Git commit:       75249d8
  Built:            Fri Jul 30 19:54:13 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.9
  GitCommit:        e25210fe30a0a703442421b0f60afac609f950a3
 runc:
  Version:          1.0.1
  GitCommit:        v1.0.1-0-g4144b63
 docker-init:
```

补充：通过官方的镜像地址下载docker会比较慢，

配置阿里云的镜像地址：

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

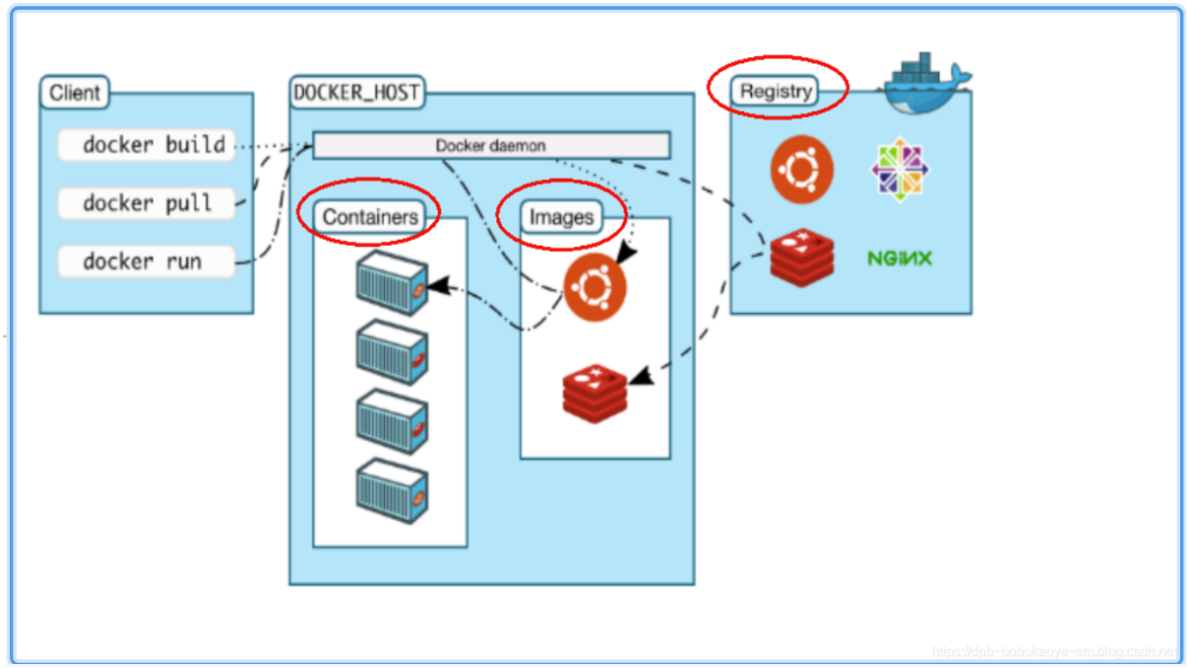
yum更新下即可：

```
yum makecache fast
```

开机启动docker

```
sudo systemctl enable docker
```

1.5 Docker的基本组成



1.5.1 镜像(image)

Docker 镜像（Image）就是一个只读的模板。镜像可以用来创建 Docker 容器，一个镜像可以创建很多容器。

docker	面向对象
容器	对象
镜像	类

1.5.2 容器(container)

Docker 利用容器（Container）独立运行的一个或一组应用。容器是用镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。可以把容器看做是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。容器的定义和镜像几乎一模一样，也是一堆层的统一视角，唯一区别在于容器的最上面那一层是可读可写的。

1.5.3 仓库(repository)

仓库（Repository）是集中存放镜像文件的场所。

仓库(Repository)和仓库注册服务器（Registry）是有区别的。仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签（tag）。

仓库分为公开仓库（Public）和私有仓库（Private）两种形式。

最大的公开仓库是 Docker Hub(https://hub.docker.com/)，存放了数量庞大的镜像供用户下载。国内的公开仓库包括阿里云 、网易云 等

1.5.4 总结

image 文件生成的容器实例，本身也是一个文件，称为镜像文件。

一个容器运行一种服务，当我们需要的时候，就可以通过docker客户端创建一个对应的运行实例，也就是我们的容器

至于仓储，就是放了一堆镜像的地方，我们可以把镜像发布到仓储中，需要的时候从仓储中拉下来就可以了。

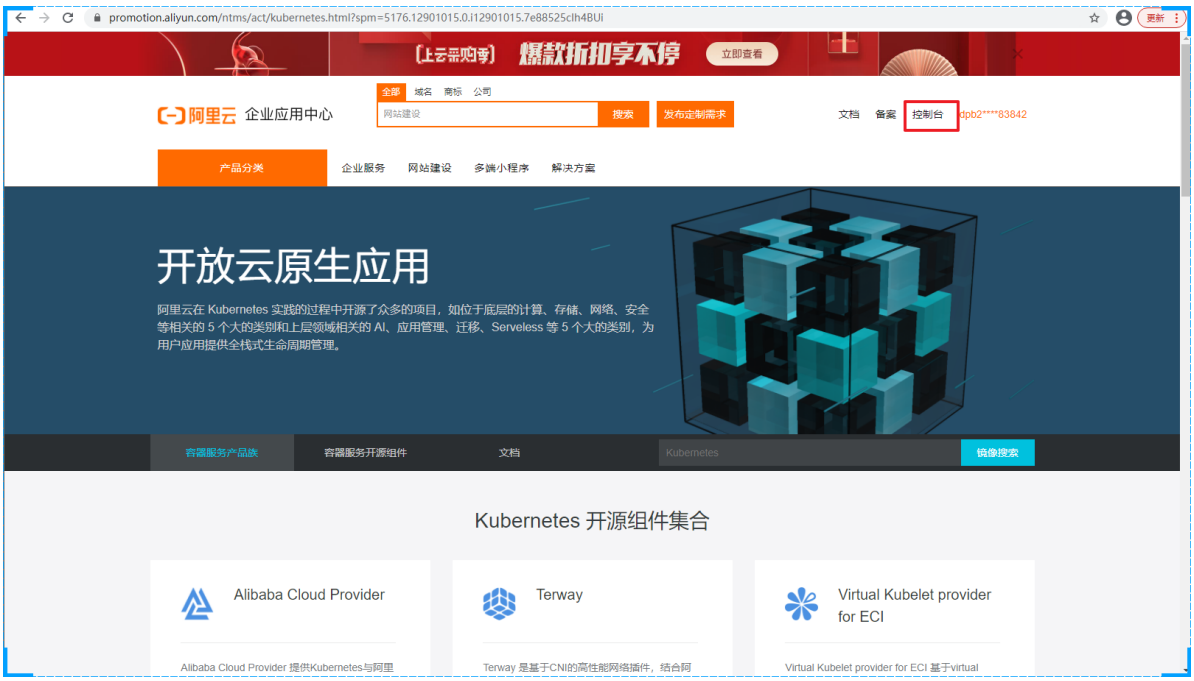
hello-world案例演示：

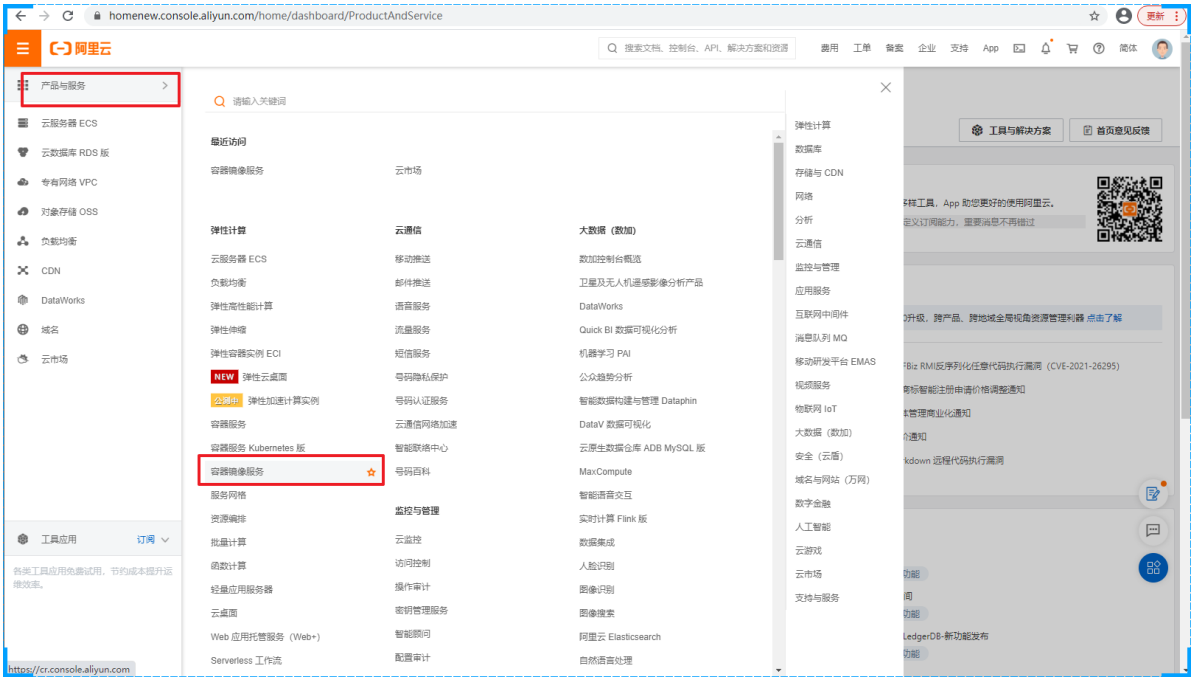
```
[root@localhost ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:393b81f0ea5a98a7335d7ad44be96fe76ca8eb2eaa76950eb8c989ebf2b78ec0
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[root@localhost ~]#
```

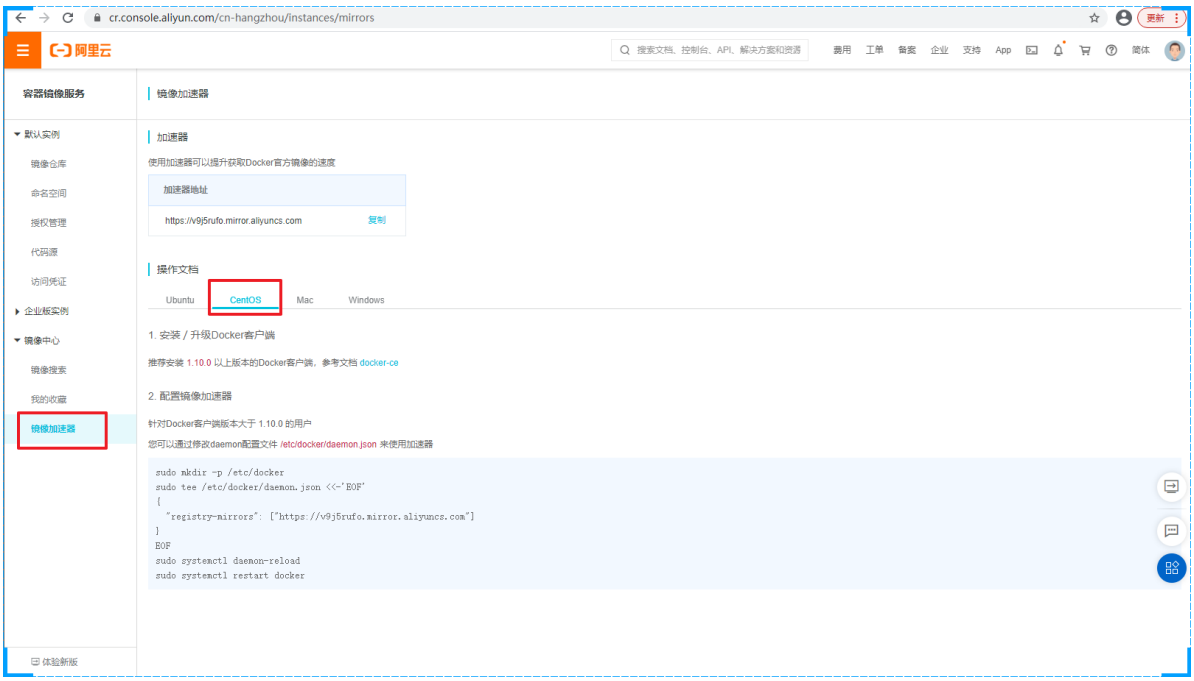
1.6 阿里云镜像加速

默认访问的仓库是在国外所以访问速度是没法保证的。为了更好的体验，我们可以配置阿里云的镜像加速

http://dev.aliyun.com/search.html


promotion.aliyun.com/ntms/act/kubernetes.html?spm=5176.12901015.0.i12901015.7e88525clh4BUi
【上云采购季】 爆款折扣享不停
立即查看
阿里云 企业应用中心
全部 域名 商标 公司
网站建设
搜索
发布定制需求
文档 备案 控制台
产品分类
企业服务 网站建设 多端小程序 解决方案
开放云原生应用
阿里云在 Kubernetes 实践的过程中开源了众多的项目，如位于底层的计算、存储、网络、安全等相关的 5 个大的类别和上层领域相关的 AI、应用管理、迁移、Serverless 等 5 个大的类别，为用户应用提供全栈式生命周期管理。
容器服务产品族
容器服务开源组件
文档
Kubernetes
镜像搜索
Kubernetes 开源组件集合
Alibaba Cloud Provider
Alibaba Cloud Provider 提供Kubernetes与阿里
Terway
Terway 是基于CNI的高性能网络插件，结合阿
Virtual Kubelet provider for ECI
Virtual Kubelet provider for ECI 基于virtual


homenew.console.aliyun.com/home/dashboard/ProductAndService
阿里云
产品与服务
云服务器 ECS
云数据库 RDS 版
专有网络 VPC
对象存储 OSS
负载均衡
CDN
DataWorks
域名
云市场
工具应用
最近访问
容器镜像服务
云市场
弹性计算
云服务器 ECS
负载均衡
弹性高性能计算
弹性伸缩
弹性容器实例 ECI
NEW 弹性云桌面
公测中 弹性加速计算实例
容器服务
容器服务 Kubernetes 版
容器镜像服务
服务网格
资源编排
批量计算
函数计算
轻量应用服务器
云桌面
Web 应用托管服务（Web+）
Serverless 工作流
云通信
移动推送
邮件推送
语音服务
流量服务
短信服务
号码隐私保护
号码认证服务
云通信网络加速
智能联络中心
号码百科
监控与管理
云监控
访问控制
操作审计
密钥管理服务
智能顾问
配置审计
大数据（数加）
数加控制台概览
卫星及无人机遥感影像分析产品
DataWorks
Quick BI 数据可视化分析
机器学习 PAI
公众趋势分析
智能数据构建与管理 Dataphin
DataV 数据可视化
云原生数据仓库 ADB MySQL 版
MaxCompute
智能语音交互
实时计算 Flink 版
数据集成
人脸识别
图像识别
图像搜索
阿里云 Elasticsearch
自然语言处理
弹性计算
数据库
存储与 CDN
网络
分析
云通信
监控与管理
应用服务
互联网中间件
消息队列 MQ
移动研发平台 EMAS
视频服务
物联网 IoT
大数据（数加）
安全（云盾）
域名与网站（万网）
数字金融
人工智能
云游戏
云市场
支持与服务
工具与解决方案
首页意见反馈
https://cr.console.aliyun.com



按照官网提示，执行对应的操作即可

```
[root@bobo01 ~]# sudo mkdir -p /etc/docker
[root@bobo01 ~]# cd /etc/docker/
[root@bobo01 docker]# ll
总用量 4
-rw-------. 1 root root 244 3月  22 11:39 key.json
[root@bobo01 docker]# vim daemon.json
[root@bobo01 docker]# sudo systemctl daemon-reload
[root@bobo01 docker]# sudo systemctl restart docker
[root@bobo01 docker]# ll
总用量 8
-rw-r--r--. 1 root root  67 3月  22 11:48 daemon.json
-rw-------. 1 root root 244 3月  22 11:39 key.json
[root@bobo01 docker]# cat daemon.json
{
  "registry-mirrors": ["https://v9j5rufo.mirror.aliyuncs.com"]
}
[root@bobo01 docker]#
```

1.7 Docker卸载

```
systemctl stop docker
yum -y remov docker-ce
rm -rf /var/lib/docker
# 重启服务
sudo systemctl restart docker
```

2.Docker的常用命令

2.1 帮助命令

命令	说明
docker version	查看docker的版本信息
docker info	查看docker详细的信息
docker --help	docker的帮助命令，可以查看到相关的其他命令

docker version

```
[root@localhost ~]# docker version
Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea
 Built:             Wed Nov 13 07:25:41 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.5
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.12
  Git commit:       633a0ea
  Built:            Wed Nov 13 07:24:18 2019
  OS/Arch:          linux/amd64
  Experimental:     false
```

docker info

```
[root@work02-node ~]# docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Build with BuildKit (Docker Inc., v0.6.3-docker)
  scan: Docker Scan (Docker Inc., v0.8.0)

Server:
 Containers: 1
  Running: 1
  Paused: 0
  Stopped: 0
 Images: 3
 Server Version: 20.10.9
 Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: active
  NodeID: 4zo3mxtrckeorjsitx1udefu2
  Is Manager: false
```

docker --help

```
[root@work02-node ~]# docker help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/root/.dock
  -c, --context string     Name of the context to use to connect to the daemon (
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/root/.d
      --tlscert string     Path to TLS certificate file (default "/root/.docker/
      --tlskey string      Path to TLS key file (default "/root/.docker/key.pem"
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  app*        Docker App (Docker Inc., v0.9.1-beta3)
  builder     Manage builds
  buildx*     Build with BuildKit (Docker Inc., v0.6.3-docker)
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  scan*       Docker Scan (Docker Inc., v0.8.0)
```

2.2 镜像命令

镜像命令	说明
docker images	列出本地主机上的镜像
docker search 镜像名称	从 docker hub 上搜索镜像
docker pull 镜像名称	从docker hub 上下载镜像
docker rmi 镜像名称	删除本地镜像

2.2.1 docker images

```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              fce289e99eb9        11 months ago       1.84kB
[root@localhost ~]#
```

镜像表格信息说明

选项	说明
REPOSITORY	表示镜像的仓库源
TAG	镜像的标签
IMAGE ID	镜像ID
CREATED	镜像创建时间
SIZE	镜像大小

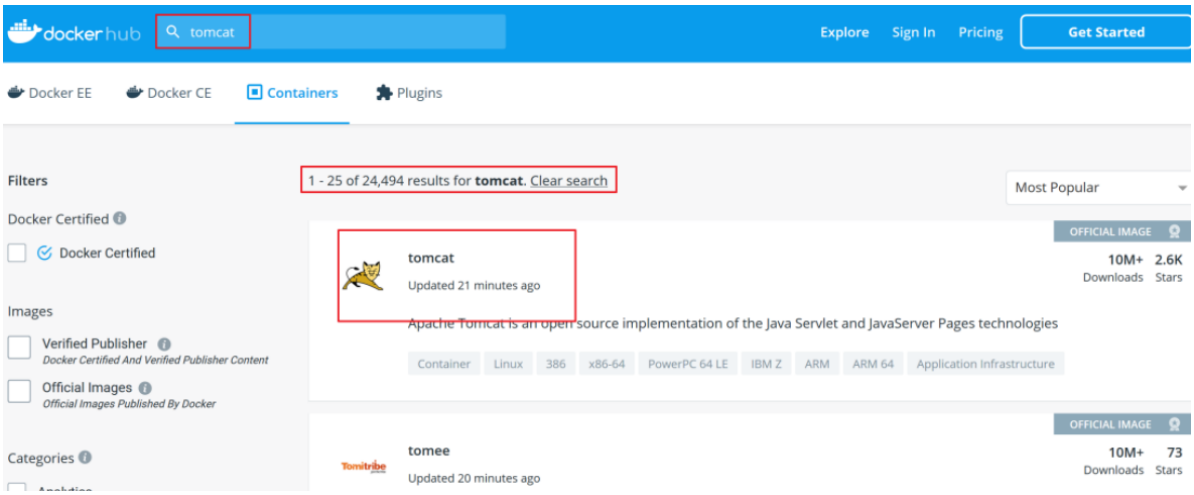
参数	说明
-a	列出本地所有的镜像
-q	只显示镜像ID
--digests	显示镜像的摘要信息
--no-trunc	显示完整的镜像信息

```
[root@bobo01 ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
hello-world   latest    d1165f221234   2 weeks ago   13.3kB
[root@bobo01 ~]#
[root@bobo01 ~]# docker images -a
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
hello-world   latest    d1165f221234   2 weeks ago   13.3kB
[root@bobo01 ~]# docker images -q
d1165f221234
[root@bobo01 ~]# docker images -qa
d1165f221234
[root@bobo01 ~]# docker images --digests
REPOSITORY    TAG       DIGEST
                IMAGE ID       CREATED       SIZE
```

```
hello-world   latest
sha256:308866a43596e83578c7dfa15e27a73011bdd402185a84c5cd7f32a88b501a24
d1165f221234   2 weeks ago   13.3kB
[root@bobo01 ~]# docker images --no-trunc
REPOSITORY    TAG       IMAGE ID
                CREATED       SIZE
hello-world   latest
sha256:d1165f2212346b2bab48cb01c1e39ee8ad1be46b87873d9ca7a4e434980a7726   2
weeks ago   13.3kB
```

2.2.2 docker search

docker hub是Docker的在线仓库，我们可以通过docker search 在上面搜索我们需要的镜像



参数名称	描述
--no-trunc	显示完整的描述信息
--limit	分页显示
-f	过滤条件 docker search -f STARS=5 tomcat

2.2.3 Docker pull

从Docker hub 上下载镜像文件

```
[root@work02-node ~]# docker search tomcat
NAME                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
tomcat                    Apache Tomcat is an open source implementati…   3155      [OK]
tomee                     Apache TomEE is an all-Apache Java EE certif…   93        [OK]
dordoka/tomcat            Ubuntu 14.04, Oracle JDK 8 and Tomcat 8 base…   58                   [OK]
consol/tomcat-7.0         Tomcat 7.0.57, 8080, "admin/admin"              18                   [OK]
cloudesire/tomcat         Tomcat server, 6/7/8                            15                   [OK]
aallam/tomcat-mysql       Debian, Oracle JDK, Tomcat & MySQL              13                   [OK]
arm32v7/tomcat            Apache Tomcat is an open source implementati…   11
andreptb/tomcat           Debian Jessie based image with Apache Tomcat…   10                   [OK]
rightctrl/tomcat          CentOS , Oracle Java, tomcat application ssl…   7                    [OK]
arm64v8/tomcat            Apache Tomcat is an open source implementati…   6
unidata/tomcat-docker     Security-hardened Tomcat Docker container.      5                    [OK]
amd64/tomcat              Apache Tomcat is an open source implementati…   3
fabric8/tomcat-8          Fabric8 Tomcat 8 Image                          2                    [OK]
cfje/tomcat-resource      Tomcat Concourse Resource                       2
jelastic/tomcat           An image of the Tomcat Java application serv…   2
oobsri/tomcat8            Testing CI Jobs with different names.           2
i386/tomcat               Apache Tomcat is an open source implementati…   1
picoded/tomcat7           tomcat7 with jre8 and MANAGER_USER / MANAGER…   1                    [OK]
camptocamp/tomcat-logback Docker image for tomcat with logback integra…   1                    [OK]
ppc64le/tomcat            Apache Tomcat is an open source implementati…   1
99taxis/tomcat7           Tomcat7                                         1                    [OK]
chenyufeng/tomcat-centos  tomcat基于centos6的镜像                                1                         [OK]
s390x/tomcat              Apache Tomcat is an open source implementati…   0
softwareplant/tomcat      Tomcat images for jira cloud testing            0                    [OK]
```

等待下载完成即可

```
[root@work02-node ~]# docker pull tomcat
Using default tag: latest
latest: Pulling from library/tomcat
bb7d5a84853b: Extracting [==================>                                ]  22.84MB/54.92MB
f02b617c6a8c: Download complete
d32e17419b7e: Download complete
c9d2d81226a4: Downloading [=========>                                          ]   10.5MB/54.57MB
fab4960f9cd2: Downloading [==============================>                     ]  3.277MB/5.42MB
da1c1e7baf6d: Download complete
79b231561270: Waiting
7d337880d8b4: Waiting
2df65a31be06: Waiting
10cbf519de23: Waiting
```

2.2.4 Docker rmi

删除方式	命令
删除单个	docker rmi -f 镜像ID
删除多个	docker rmi -f 镜像1:TAG 镜像2:TAG
删除全部	docker rmi -f $(docker images -qa)

-f 表示强制删除

```
[root@localhost ~]# docker rmi -f tomcat hello-world
Untagged: tomcat:latest
Untagged: tomcat@sha256:68355b27adee5fc76c23e3d3cb994bd2733f05aa8e2c070a61346e16eed308ac
Deleted: sha256:6fa48e04772109a282efcc0eb42211b44d50c27156cd5f3bae77754882d67bb6
Deleted: sha256:62d023e77b1252586eed9b494feef7d3064fa6840be3371c81cb512440566406
Deleted: sha256:a106a7fb50963e543f857e3116c161cbfa57cfcd91ccf2117b159a8e5af2a4cb
Deleted: sha256:ab0e4bc9eb374158707db581ef55a732332cfd3fcc505baf22490e901eda185f
Deleted: sha256:b35dd16822b6808f5f8bd5da55f3241ff5cade5eb7d25d1440df90209ca159b8
Deleted: sha256:97200aa46a2ef0247e91462cc37b0bfa10b167da4a9fce0d54d598f70edb06d5
Deleted: sha256:7f3e89512c144c481c815517998f69b6063f33f36d6946ef1ec6cda6c42cd6e5
Deleted: sha256:23b818b261d944ed79f99d8b96ff917835f68b296a0002e094b3db2555e26f12
Deleted: sha256:8b595b250565c6ab1163e71f4104587188518185719929ce0429e16fdfb76b1e
Deleted: sha256:d692181dfebae7a739bdcbd55ce40348dcedecaa741d2978933ecc830955c44c
Deleted: sha256:e4b20fcc48f4a225fd29ce3b3686cc51042ce1f076d88195b3705b5bb2f38c3d
Untagged: hello-world:latest
Untagged: hello-world@sha256:4fe721ccc2e8dc7362278a29dc660d833570ec2682f4e4194f4ee23e415e1064
Deleted: sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc1741675f587e
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

2.3 容器命令

有镜像才能创建容器。

```
[root@localhost ~]# docker pull centos
Using default tag: latest
latest: Pulling from library/centos
729ec3a6ada3: Pull complete
Digest: sha256:f94c1d992c193b3dc09e297ffd54d8a4f1dc946c37cbeceb26d35ce1647f88d9
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
tomcat              latest              6fa48e047721        11 days ago         507MB
centos              latest              0f3e07c0138f        2 months ago        220MB
hello-world         latest              fce289e99eb9        11 months ago       1.84kB
```

2.3.1 创建并启动容器

创建并启动一个容器的命令

```
docker run [OPTIONS] IMAGE [COMMAND]
```

OPTIONS中的一些参数

options	说明
--name	"容器新名字": 为容器指定一个名称
-d	后台运行容器，并返回容器ID，也即启动守护式容器
`-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`
-P:	随机端口映射
-p	指定端口映射，有以下四种格式 ip:hostPort:containerPort ip::containerPort `hostPort:containerPort` containerPort

交互式的容器

```
docker run -it centos /bin/bash
```

```
[root@localhost ~]# docker run -it centos /bin/bash
[root@31d1187850f0 /]# pwd
/
```

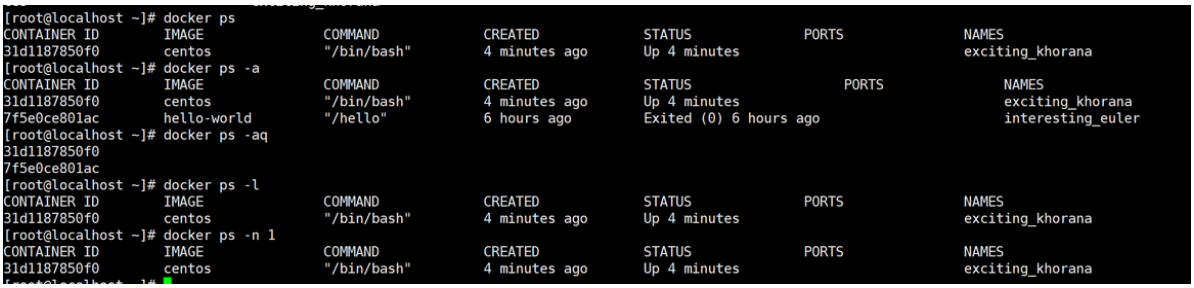
2.3.2 列举运行的容器

我们要查看当前正在运行的容器有哪些，可以通过ps 命令来查看

```
docker ps [OPTIONS]
```

OPTONS可用的参数

OPTIONS	说明
-a	列出当前所有正在运行的容器+历史上运行过的
-l	显示最近创建的容器。
-n	显示最近n个创建的容器。
-q	静默模式，只显示容器编号。
--no-trunc	不截断输出。



2.3.3 退出容器命令

我们启动了一个容器后，如何退出容器

退出方式	说明
exit	容器停止退出
ctrl+p+q	容器不停止退出

2.3.4 启动容器

```
docker start 容器ID或者容器名称
```

2.3.5 重启容器

```
docker restart 容器id或者名称
```

2.3.6 停止容器

```
docker stop 容器ID或者名称
```

还可以通过强制停止方式处理

```
docker kill 容器ID或者名称
```

2.3.7 删除容器

有时候容器使用完成就没有作用了，我们想要删除掉容器，这时可以通过rm命令

```
docker rm 容器ID
docker rm -f $(docker ps -qa)
docker ps -a -q | xargs docker rm
```

2.4 其他命令

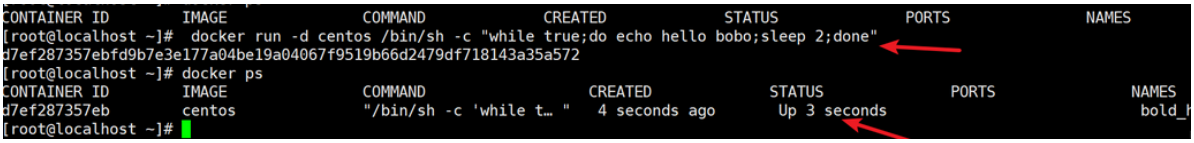
2.4.1 守护式容器

```
docker run -d 容器名称
```

我们通过 docker ps -a 可以看到刚刚启动的容器已经退出了

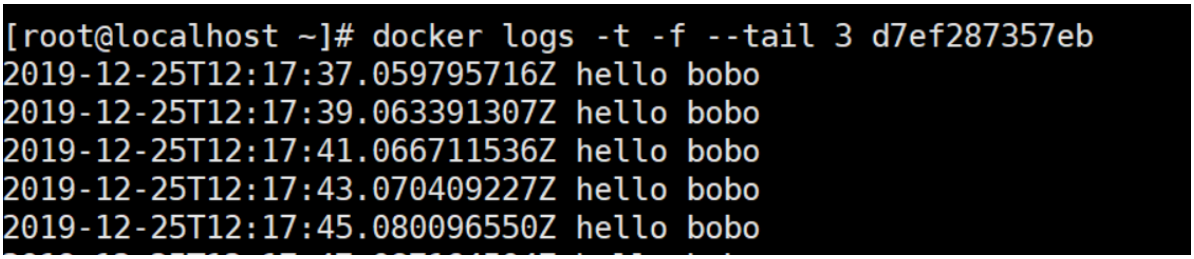
为了让守护式容器能够一直执行，我们可以在启动容器后在后台运行一个循环的脚本

```
docker run -d centos /bin/bash -c 'while true;do echo hello bobo;sleep 2;done'
```



查看我们运行的日志

```
docker logs -t -f --tail 3 容器ID
```



查看容器中运行的进程

```
docker top 容器ID
```

2.4.2 查看容器细节

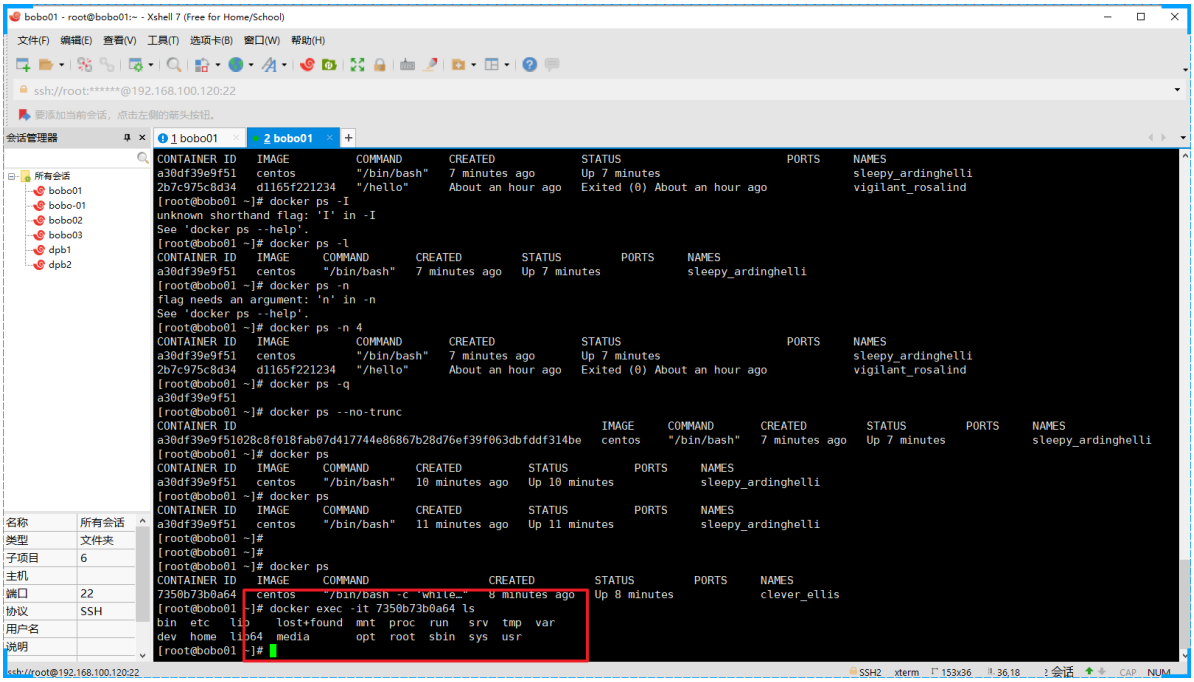
我们想要查看容器的细节可以通过inspect命令

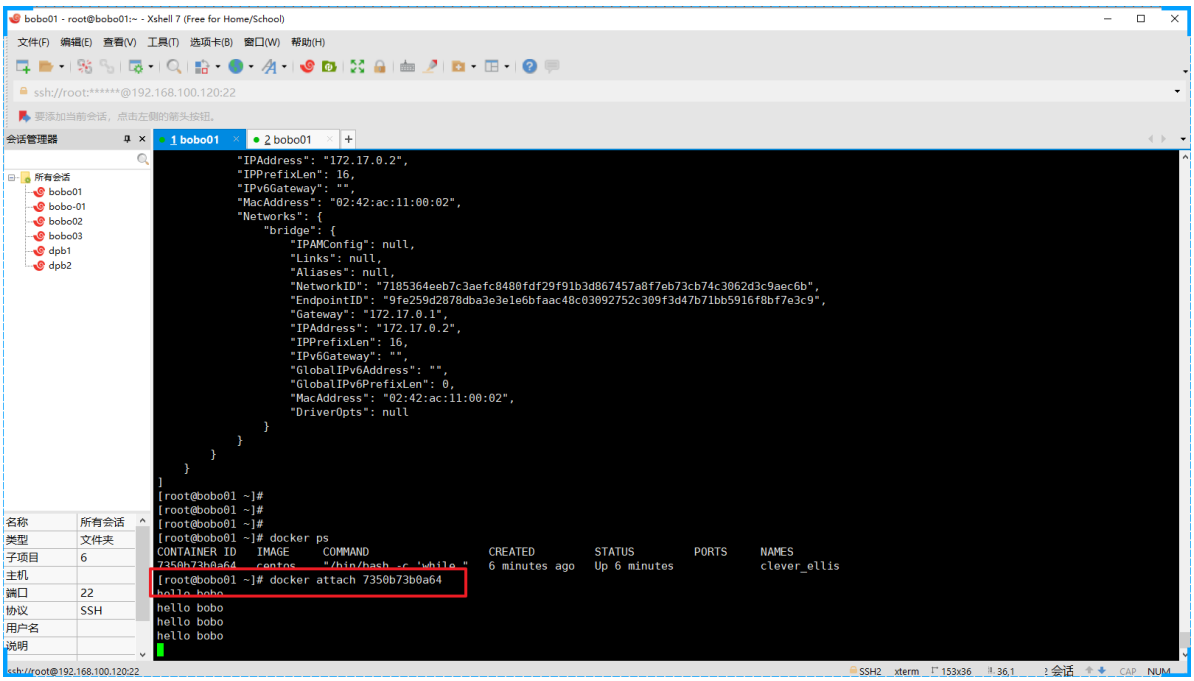
```
docker inspect 容器ID
```

```
bang-steep /usr/bin/steep 2
[root@localhost ~]# docker inspect d7ef287357eb
[
    {
        "Id": "d7ef287357ebfd9b7e3e177a04be19a04067f9519b66d2479df718143a35a572",
        "Created": "2019-12-25T12:14:45.137655843Z",
        "Path": "/bin/sh",
        "Args": [
            "-c",
            "while true;do echo hello bobo;sleep 2;done"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 9469,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2019-12-25T12:14:46.185318785Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:0f3e07c0138fbe05abcb7a9cc7d63d9bd4c980c3f61fea5efa32e7c4217ef4da",
        "ResolvConfPath": "/var/lib/docker/containers/d7ef287357ebfd9b7e3e177a04be19a04067f9519b66d2479df718143a35a572/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/d7ef287357ebfd9b7e3e177a04be19a04067f9519b66d2479df718143a35a572/hostname",
        "HostsPath": "/var/lib/docker/containers/d7ef287357ebfd9b7e3e177a04be19a04067f9519b66d2479df718143a35a572/hosts",
        "LogPath": "/var/lib/docker/containers/d7ef287357ebfd9b7e3e177a04be19a04067f9519b66d2479df718143a35a572/d7ef287357ebfd9b7e3e17
```

2.4.3 进入运行的容器

进入方式	说明
exec	在容器中打开新的终端,并且可以启动新的进程
attach	直接进入容器启动命令的终端，不会启动新的进程





2.4.4 文件复制

我们有时需要从容器中拷贝内容到宿主机中

```
docker cp 容器ID:容器内路径  目的地路径
```

3.Docker镜像文件介绍

3.1 镜像是什么

首先我们来看看镜像到底是什么？虽然前面有介绍过`镜像`和`容器`,但也不是特别的深入。

> 镜像是一种轻量级、可执行的独立软件包，用来打包软件运行环境和基于运行环境开发的软件，它包含运行某个软件所需的所有内容，包括代码、运行时、库、环境变量和配置文件。

3.1.1 UnionFS

UnionFS（联合文件系统）：Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

3.1.2 镜像加载原理

Docker镜像加载原理：

Docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS。

Bootfs(boot file system)主要包含Bootloader和Kernel, Bootloader主要是引导加载Kernel, Linux刚启动时会加载Bootfs文件系统，在Docker镜像的最底层是bootfs。这一层与我们典型的Linux/Unix系统是一样的，包含Boot加载器和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权已由bootfs转交给内核，此时系统也会卸载bootfs。

Rootfs (root file system)，在Bootfs之上。包含的就是典型 Linux 系统中的 /dev, /proc, /bin, /etc 等标准目录和文件。Rootfs就是各种不同的操作系统发行版，比如Ubuntu，Centos等等。

3.1.3 分层的镜像

其实我们前面在 pull 文件的时候比如 Tomcat，在pull界面我们就可以看到下载的文件是一层层的。

3.1.4 分层结构的特点

其实我们也会考虑docker为什么会才用这种分层的结果，它有什么好处呢？最大的一个好处就是共享资源

比如：有多个镜像都从相同的 base 镜像构建而来，那么宿主机只需在磁盘上保存一份base镜像，同时内存中也只需加载一份 base 镜像，就可以为所有容器服务了。而且镜像的每一层都可以被共享。

3.2 镜像的特点

大家需要注意，Docker镜像都是只读的，当容器启动时，一个新的可写层被加载到镜像的顶部，这一层通常被称为容器层,容器层之下的都叫镜像层。

3.3 镜像操作

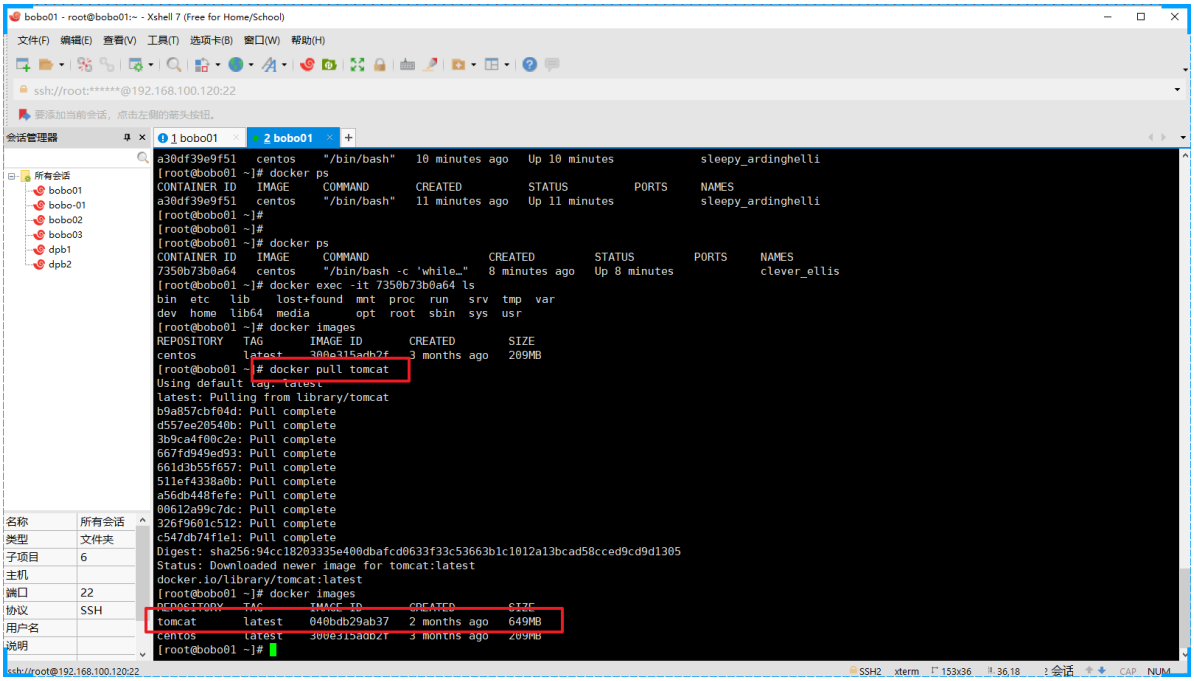
我们现在已经掌握了从docker hub上获取相关镜像，然后运行容器，并作出我们自己的处理，但有时候我们需要将我们自己的容器制作为对应的镜像，以便后面继续使用，这时我们就需要用到docker commit ...命令了，这节我们就通过案例来介绍下 docker commit ...命令的使用

```
docker commit -m="要提交的描述信息" -a="作者" 容器ID 要创建的目标镜像名:[标签名]
```

操作案例

我们通过tomcat镜像来创建容器后操作处理,然后将容器制作为新的镜像然后我们通过新的镜像来制作容器来演示这个效果，有点绕，我们直接通过案例来说。

3.3.1 下载tomcat镜像

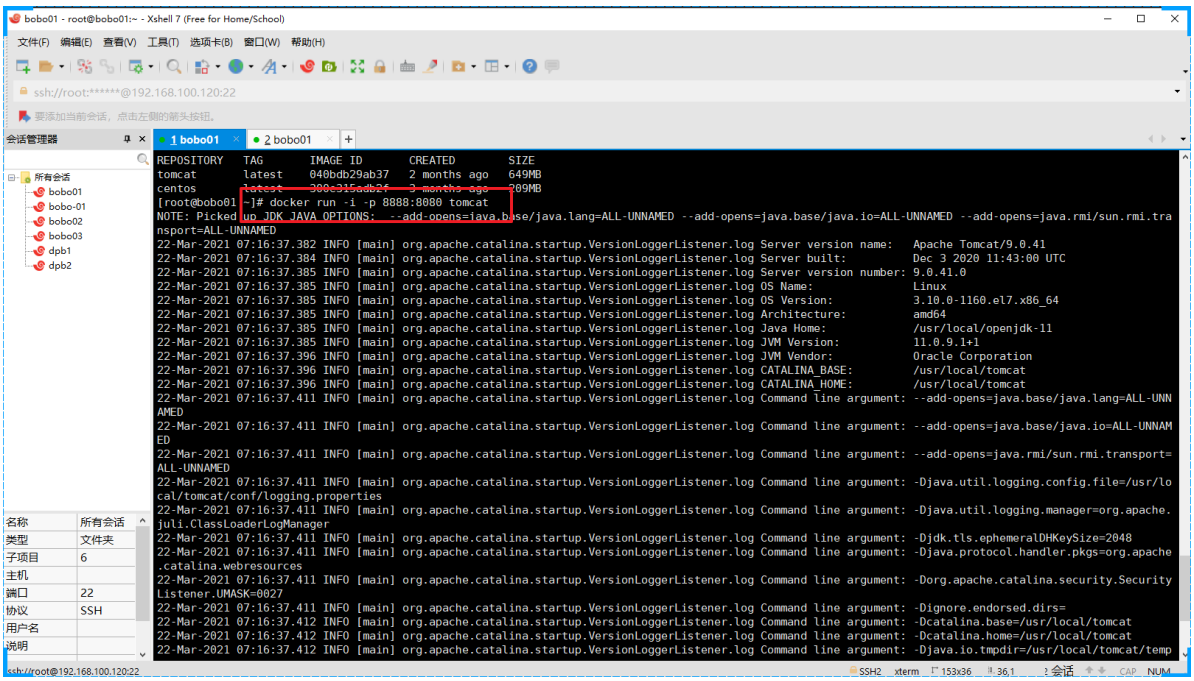


3.3.2 创建容器并启动

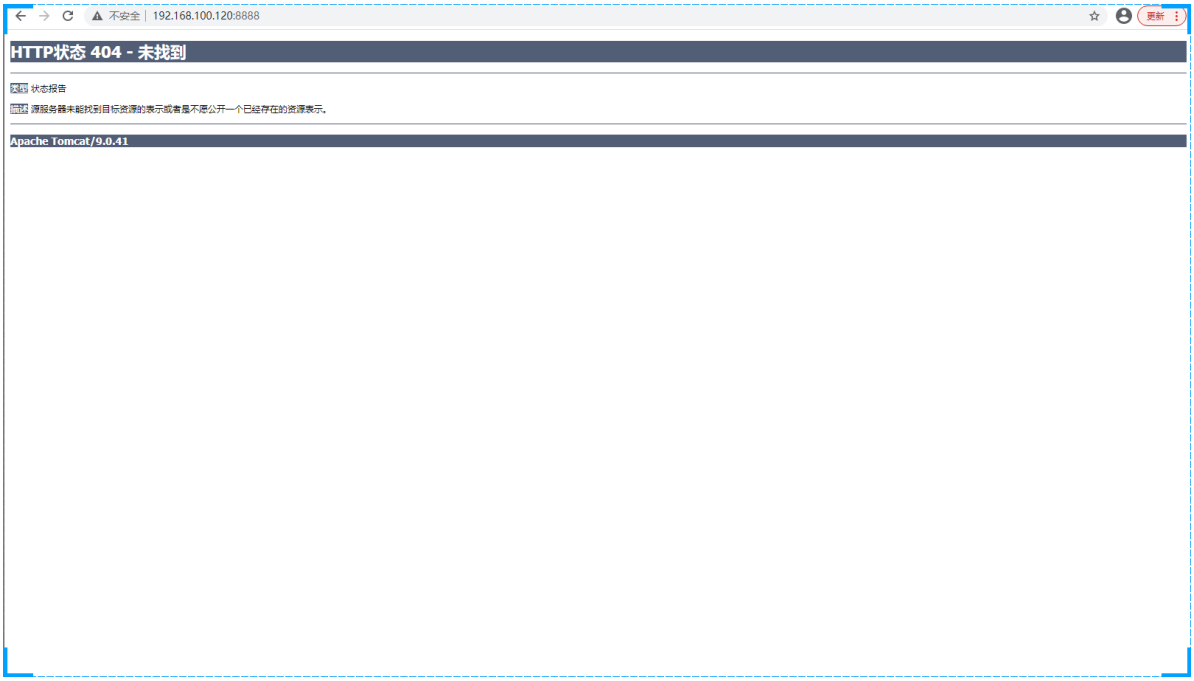
```
docker run -it -p 8888:8080 tomcat
```

参数

参数	说明
-p	主机端口:docker容器端口
-P	随机分配端口
-i	交互
-t	终端



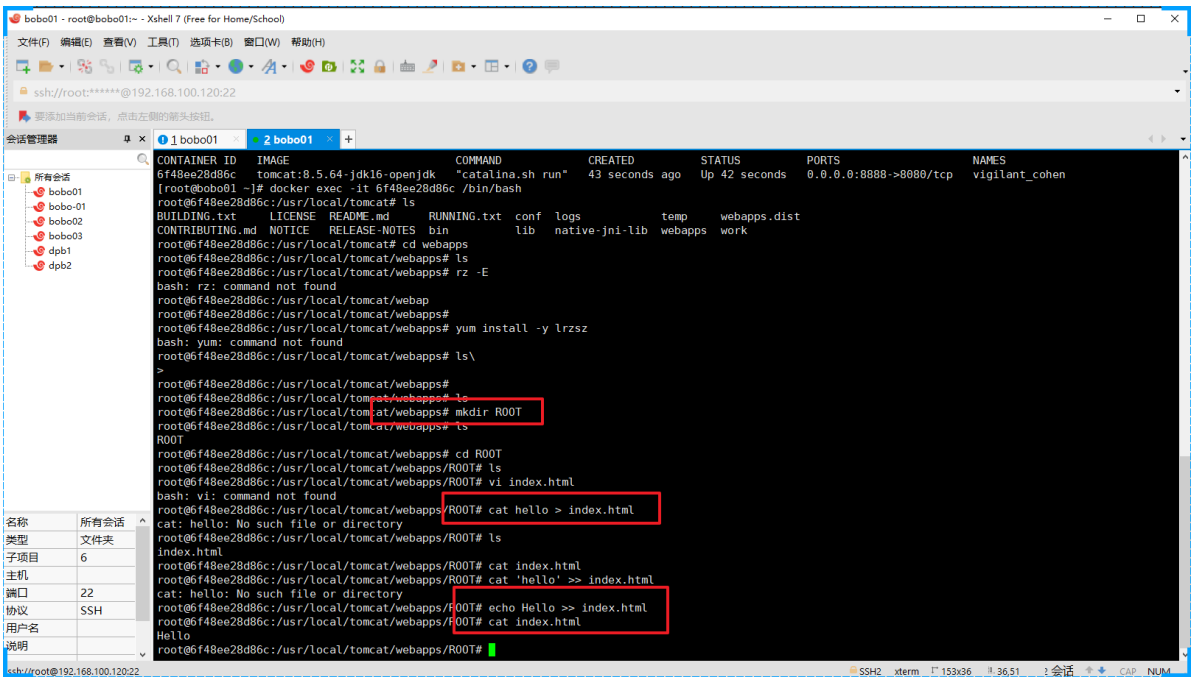
404

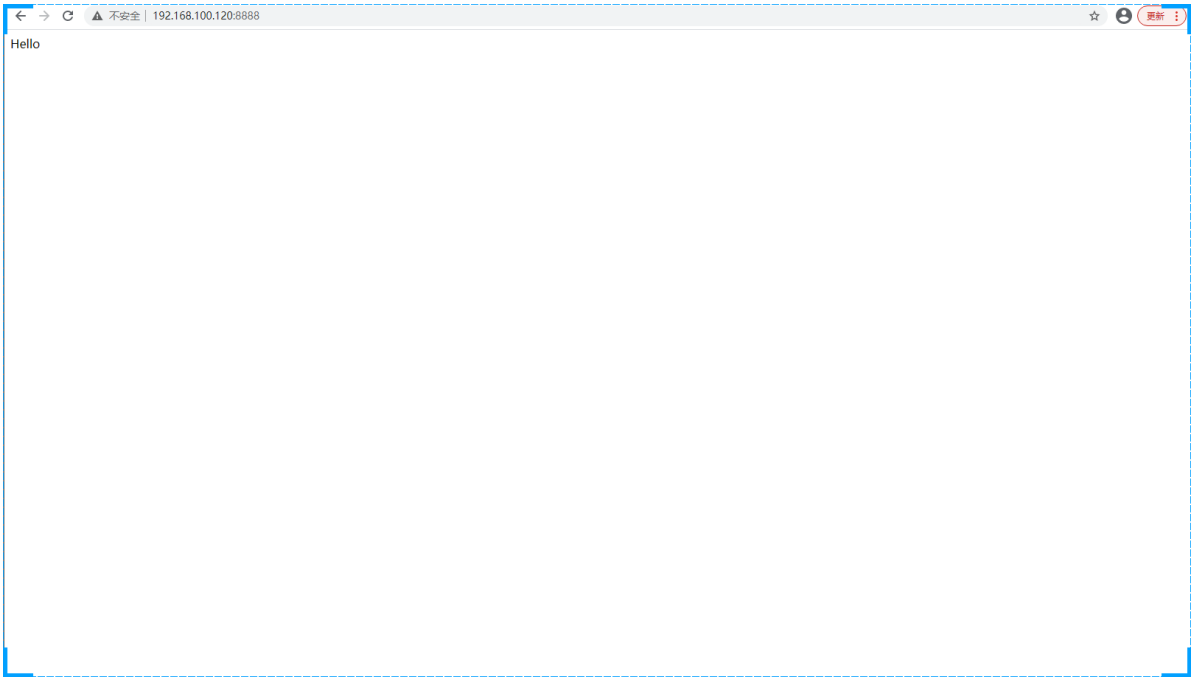


3.3.3 修改容器

我们发现启动的容器中没有要访问的资源，那么我们自己创建对应的资源即可

```
docker exec -it 容器ID /bin/bash
```

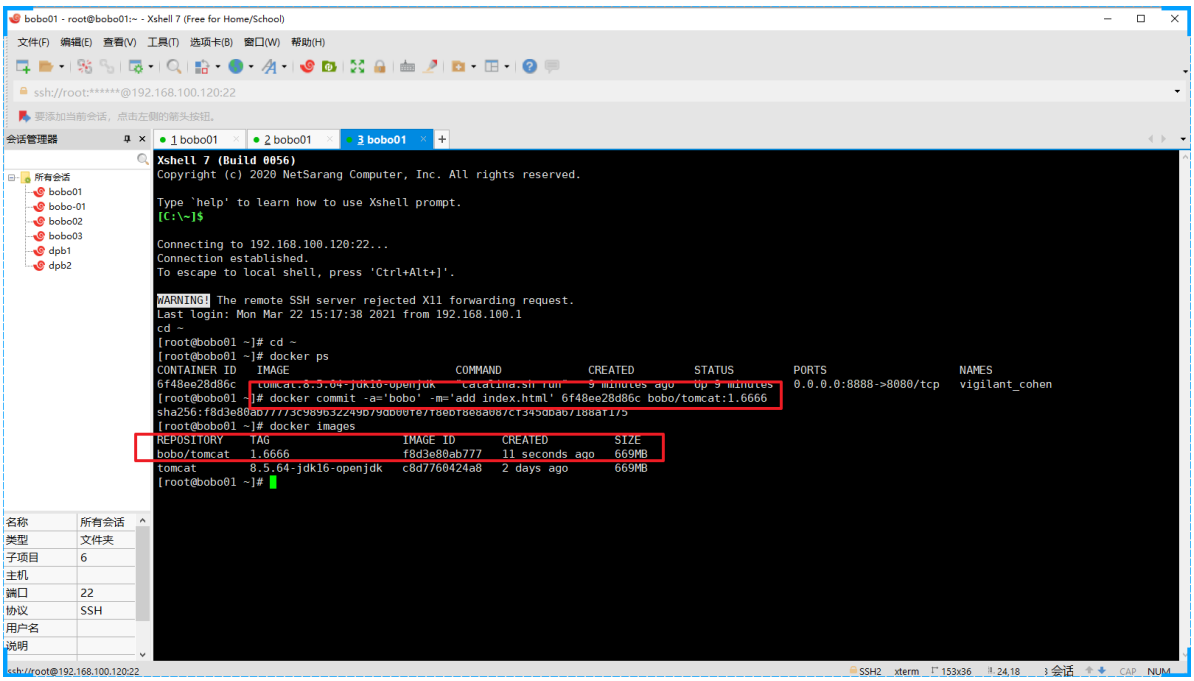




3.3.4 创建镜像

我们现在的容器和下载的有区别了，我们可以在这个基础上来创建新的镜像

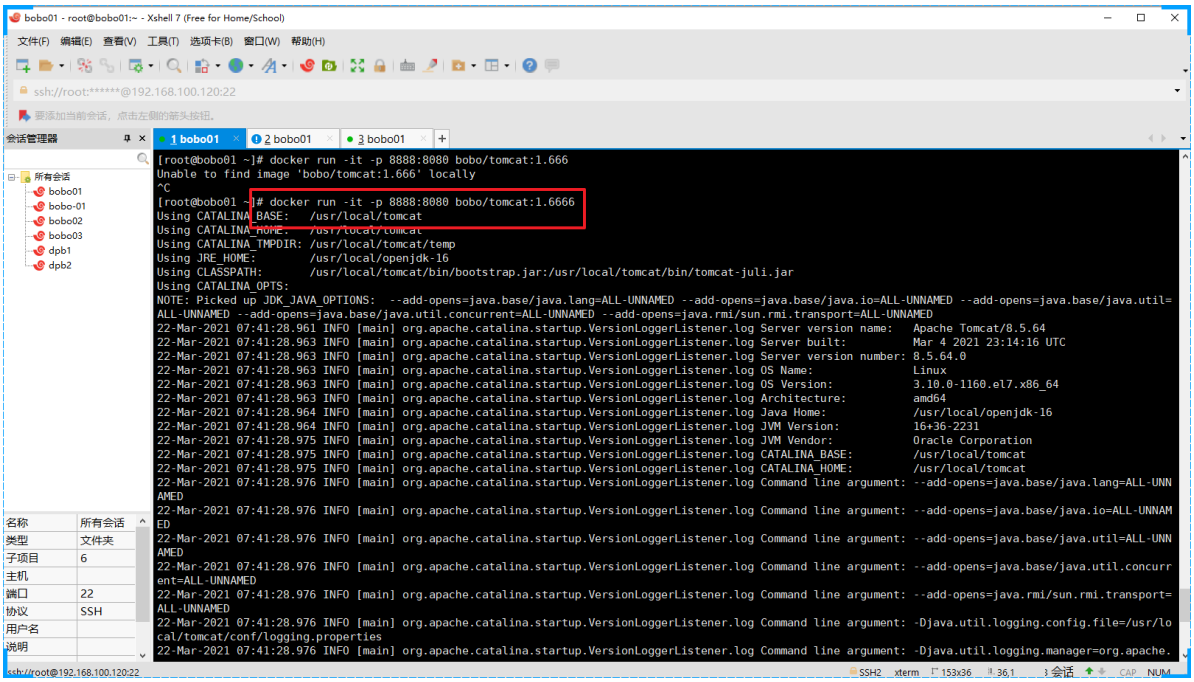
```
docker commit -a='bobo' -m='add index.html' 容器ID bobo/tomcat:1.666
```

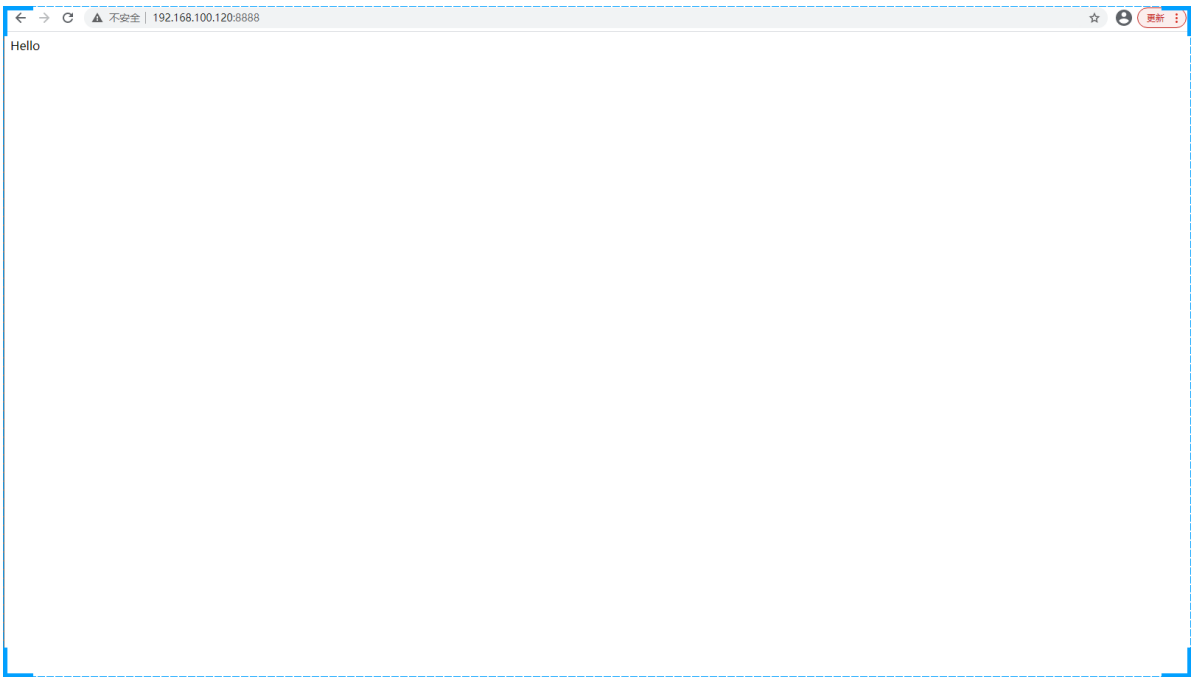


3.3.5 启动新的镜像

现在我们可以通过我们自己新创建的镜像文件来创建并启动容器了

```
docker run -it -p 8888:8080 bobo/tomcat:1.6666
```





4.Docker数据卷

4.1 数据卷

前面我们介绍了镜像和容器，通过镜像我们可以启动多个容器，但是我们发现当我们的容器停止获取删除后，我们在容器中的应用的一些数据也丢失了，这时为了解决容器的数据持久化，我们需要通过容器数据卷来解决这个问题

4.1.1 数据卷是什么

Docker容器产生的数据，如果不通过docker commit生成新的镜像，使得数据做为镜像的一部分保存下来，那么当容器删除后，数据自然也就没有了。为了能保存数据在docker中我们使用卷。简单来说，容器卷就相当于Redis中持久化方式的RDB和AOF。

4.1.2 解决了什么问题

卷就是目录或文件，存在于一个或多个容器中，由docker挂载到容器，但不属于联合文件系统，因此能够绕过Union File System提供一些用于持续存储或共享数据的特性：

卷的设计目的就是数据的持久化，完全独立于容器的生存周期，因此Docker不会在容器删除时删除其挂载的数据卷

特点：

1. 数据卷可在容器之间共享或重用数据
2. 卷中的更改可以直接生效
3. 数据卷中的更改不会包含在镜像的更新中
4. 数据卷的生命周期一直持续到没有容器使用它为止

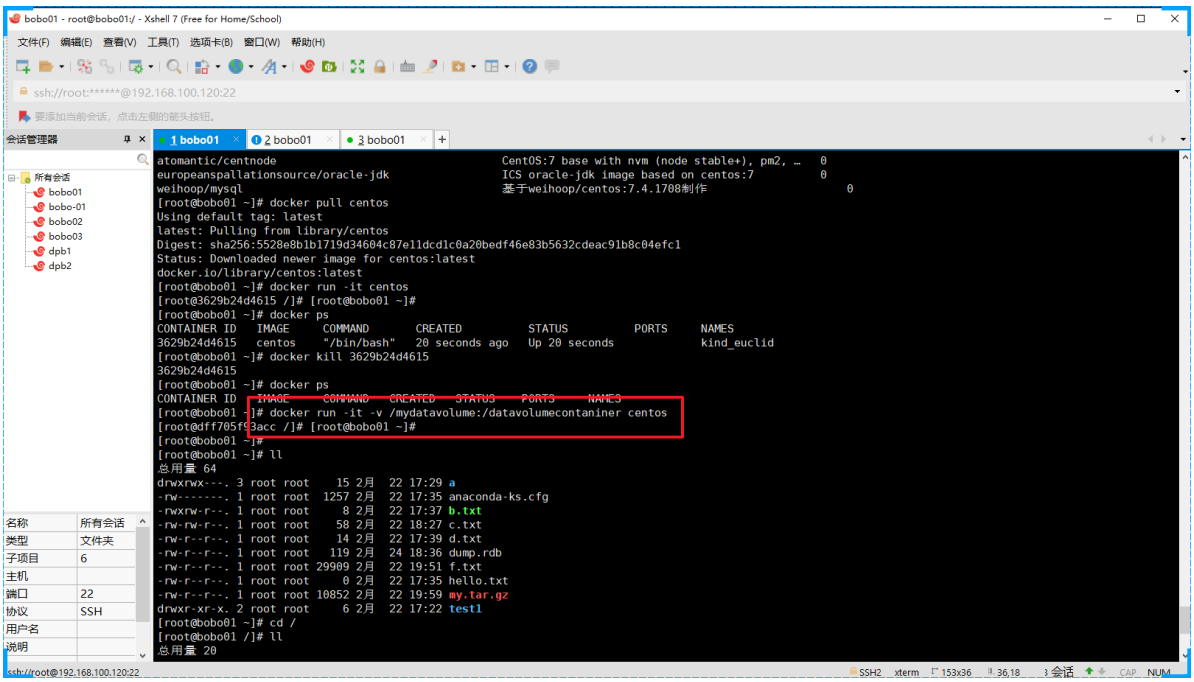
> 持久化，容器间继承和共享数据

4.1.3 数据卷使用

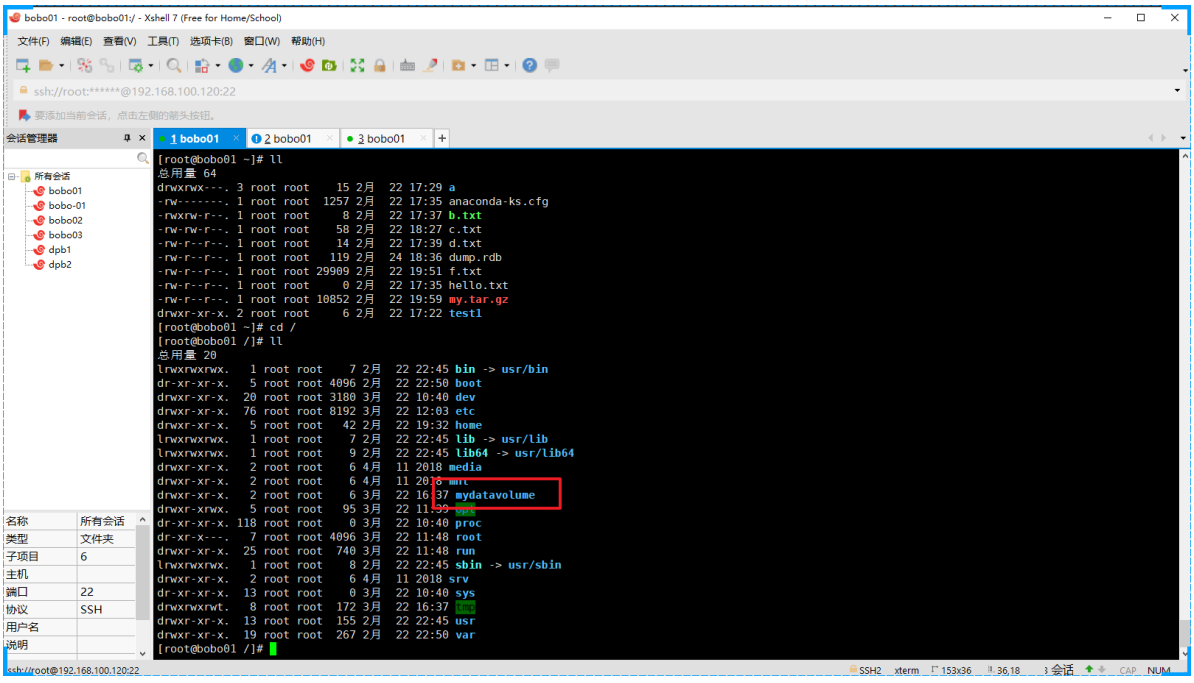
4.1.3.1 直接添加

运行一个centos容器

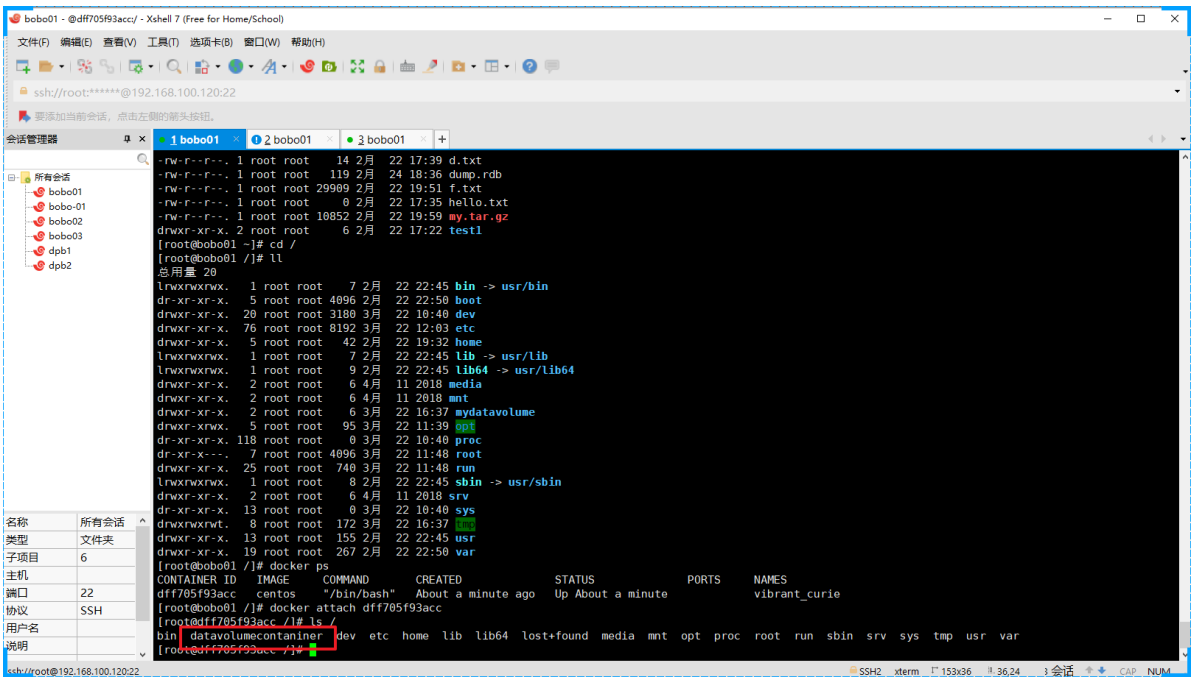
```
docker run -it -v /宿主机绝对路径:/容器内目录 镜像名
```



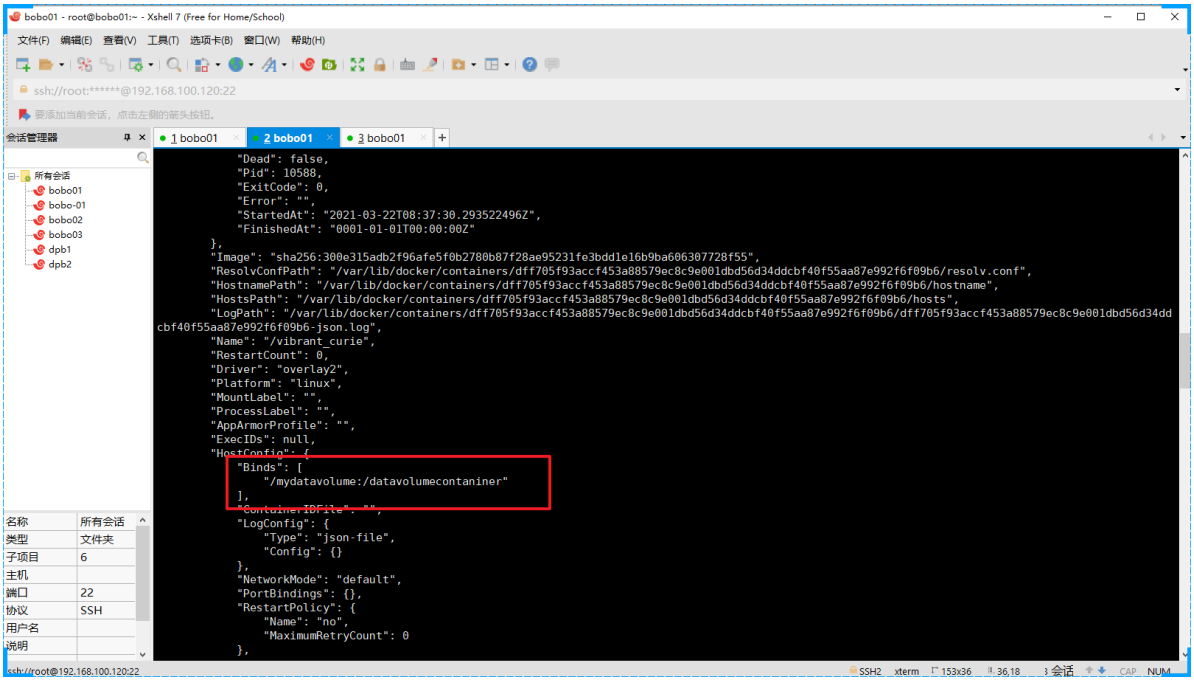
在宿主机的根目录下会多出对应的文件夹



然后在容器的根目录下也会出现对应的文件夹

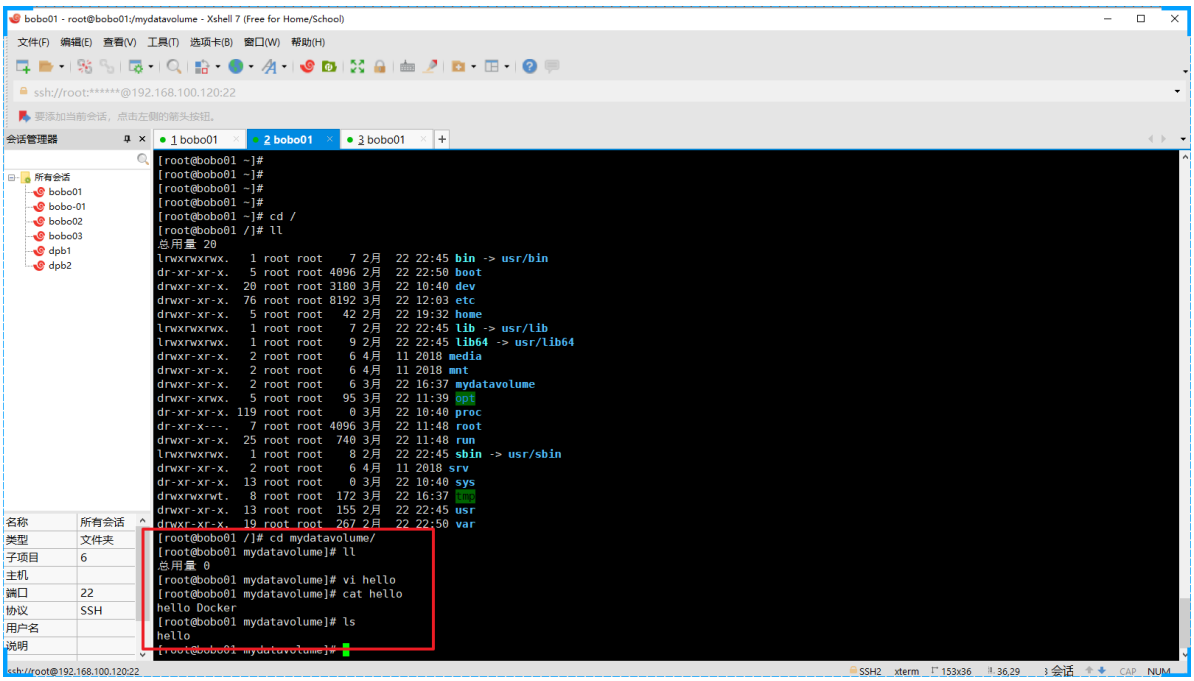


通过inspect命令可以查询容器的详情

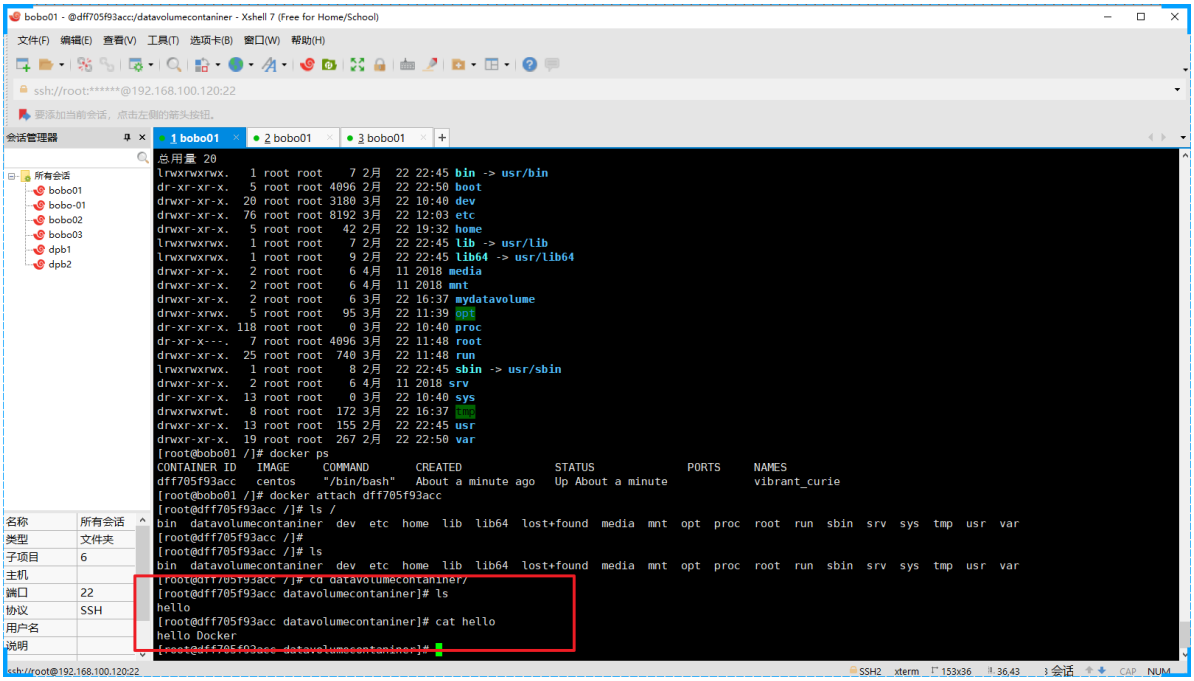


数据共享的操作

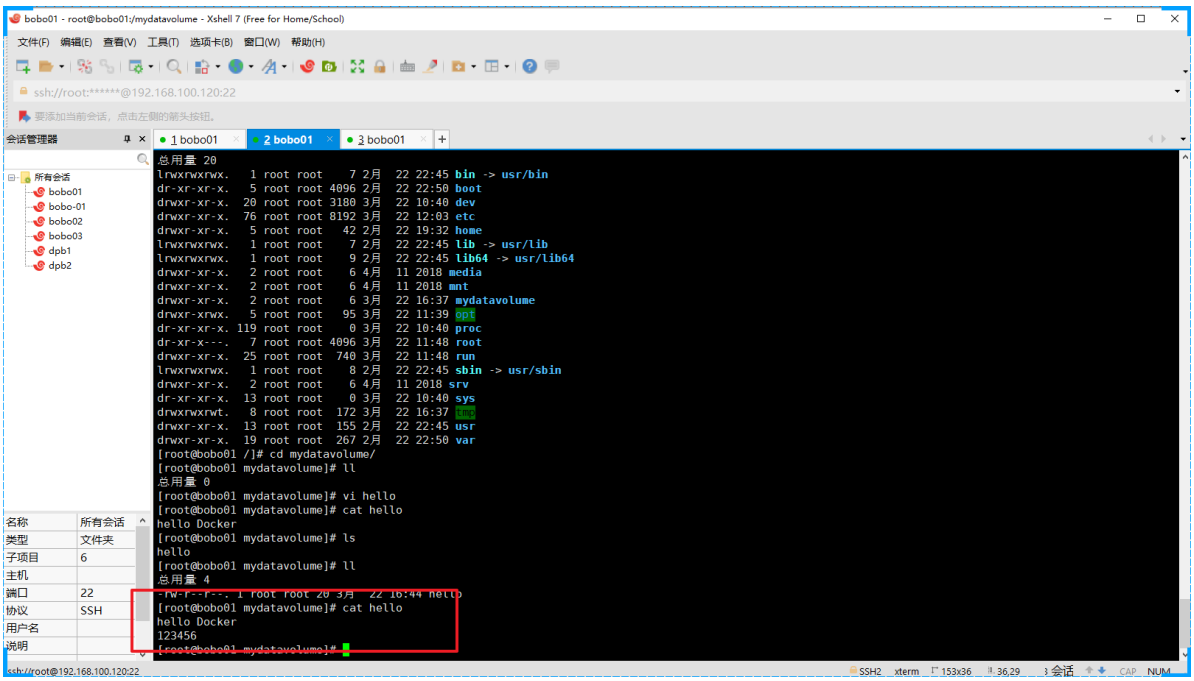
宿主机添加对应的文件



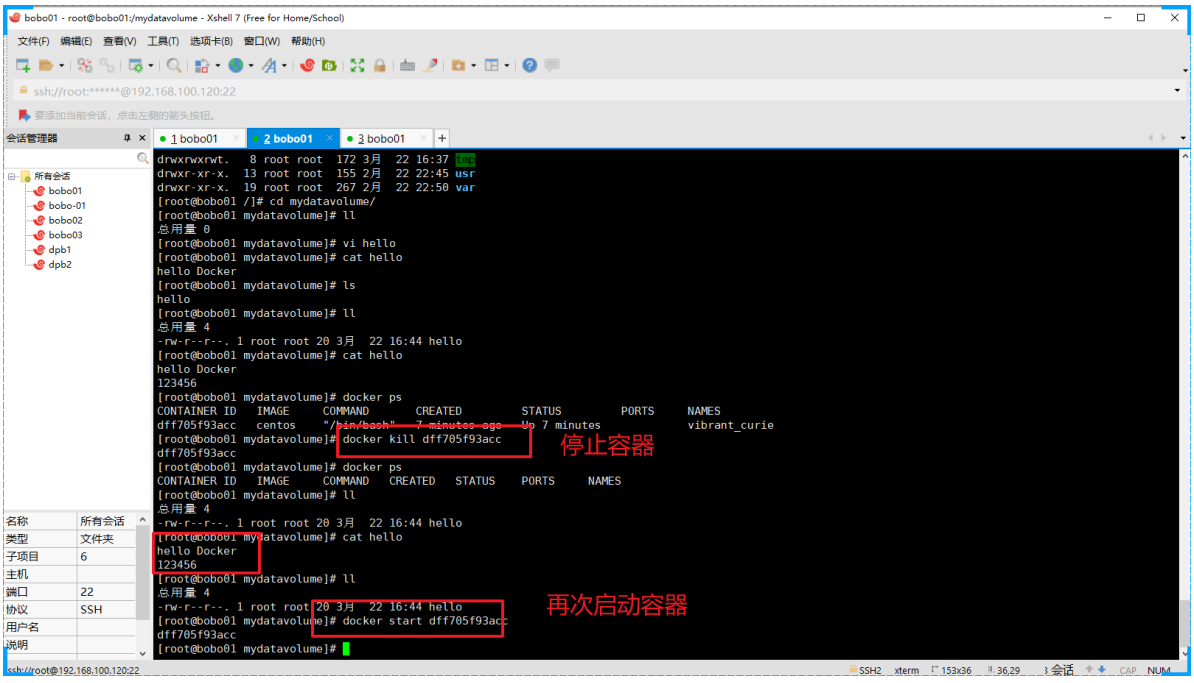
容器中查看

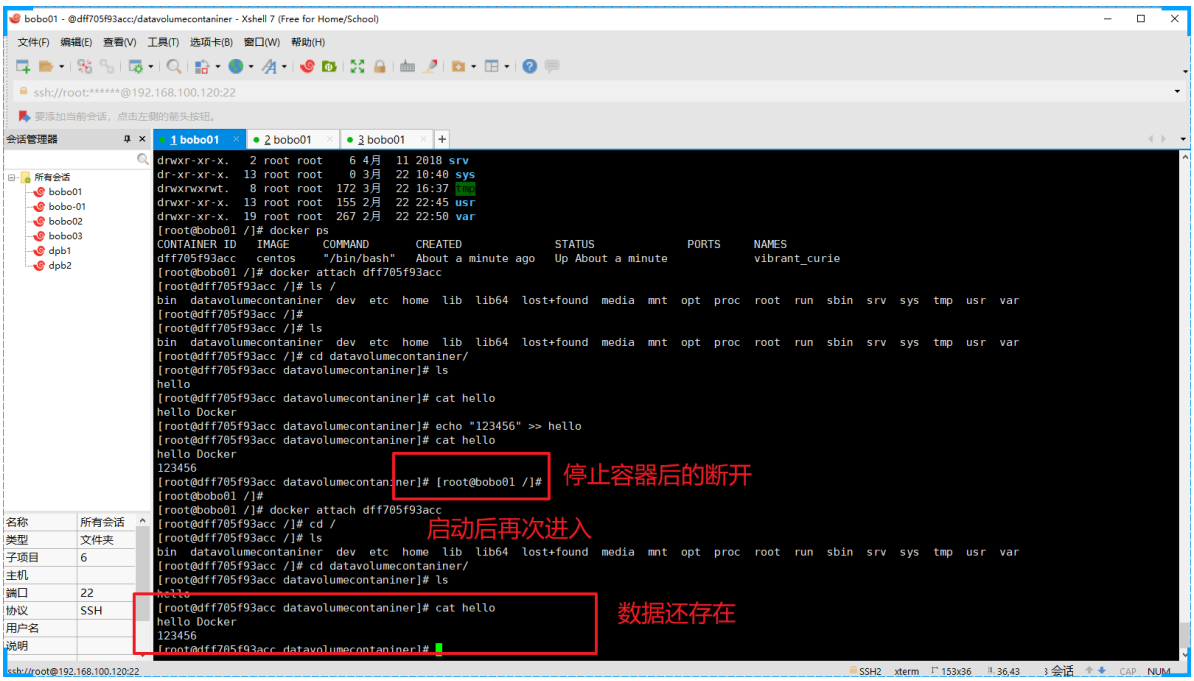


容器中可以同步看到，然后在容器中修改数据

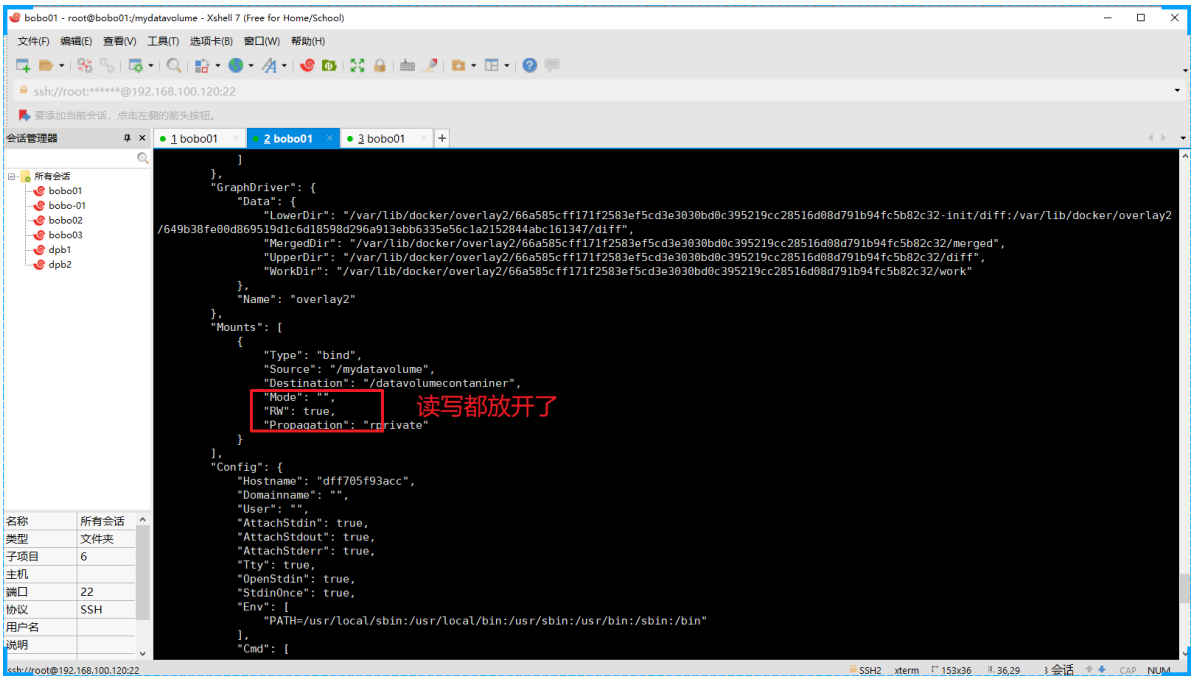


停止掉容器后，数据依然存在



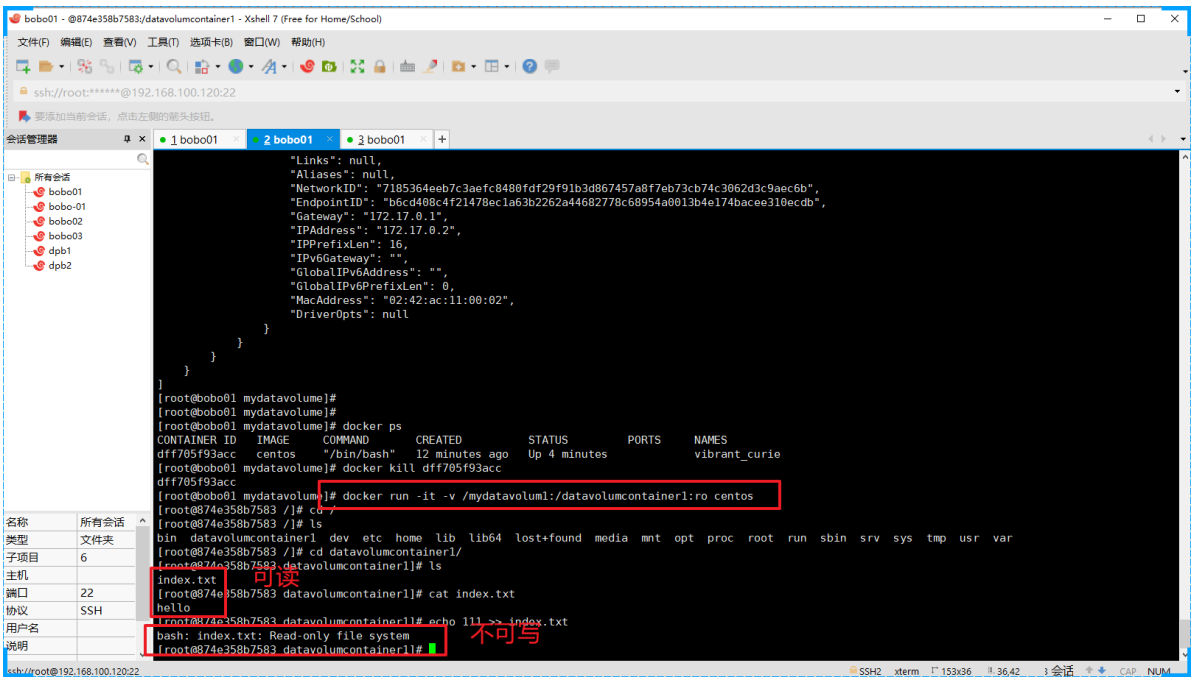


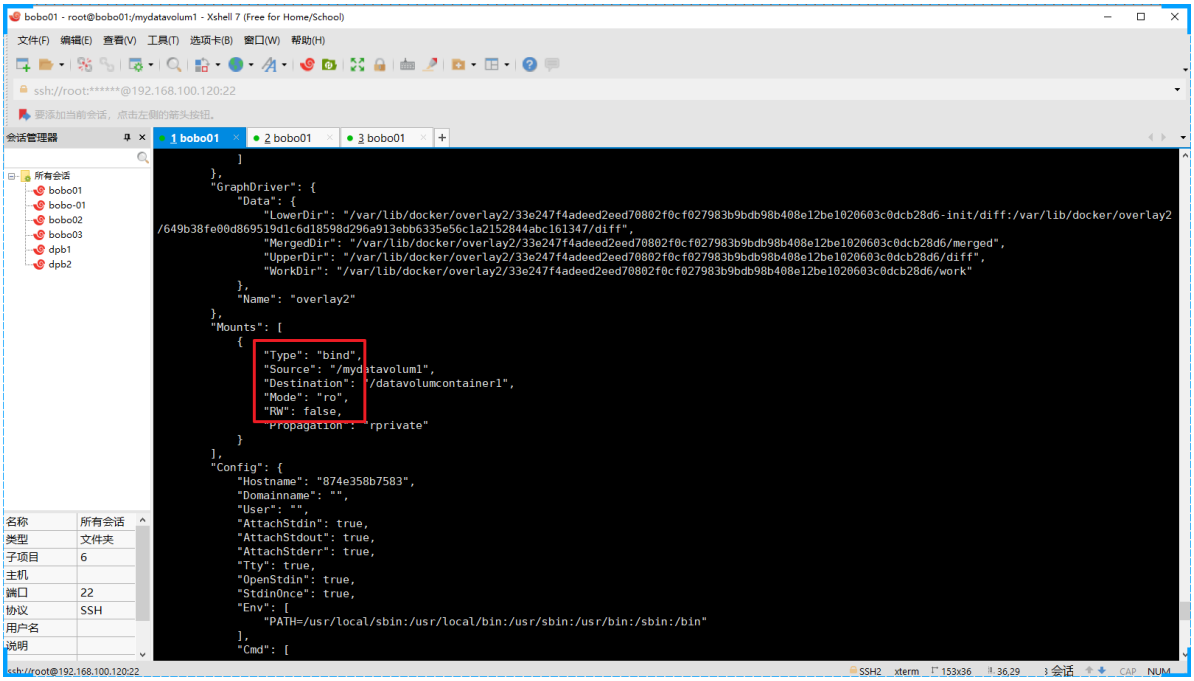
权限控制：不允许在容器中修改



修改权限

```
docker run -it -v /宿主机绝对路径:/容器目录:ro 镜像名
```





4.1.3.2 DockerFiler添加

宿主机跟目录下创建一个mydocker，并在该目录下创建一个文件，内容如下

```
# volume test

FROM centos

VOLUME ["/dataVolumeContainer1","/dataVolumeContainer2"]

CMD echo "finished,--------success1"

CMD /bin/bash
```

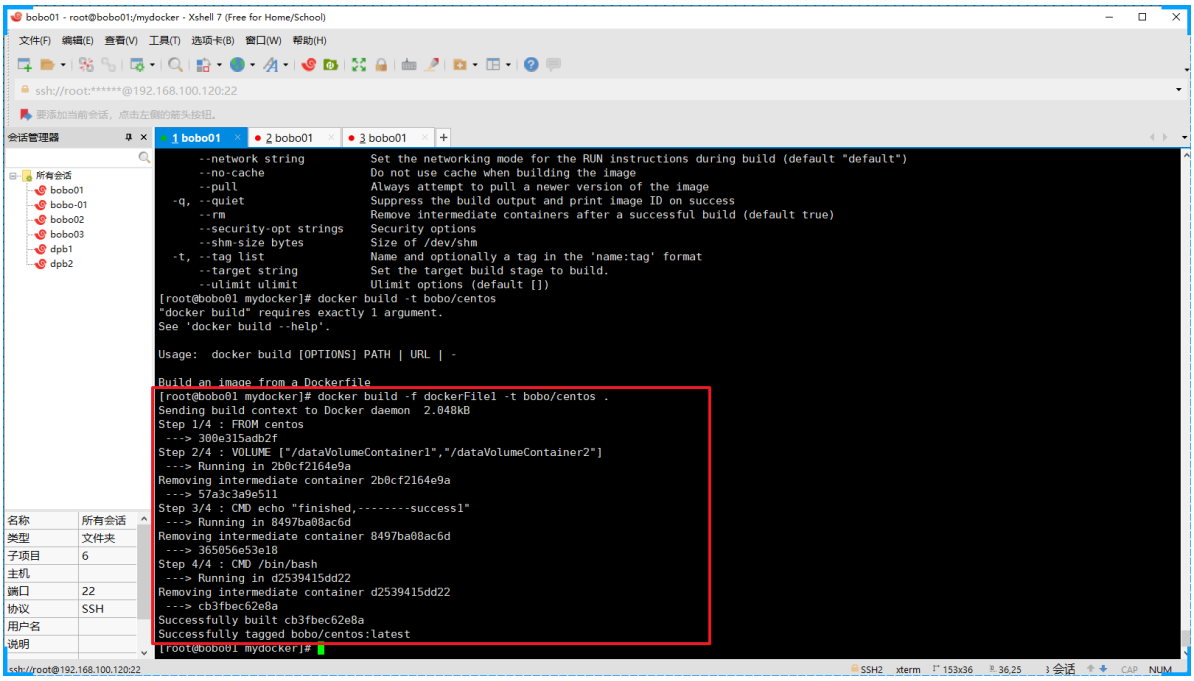
根据这个DockerFile构建我们的镜像文件

```
docker build -f dockerFile1 -t bobo/centos .
```

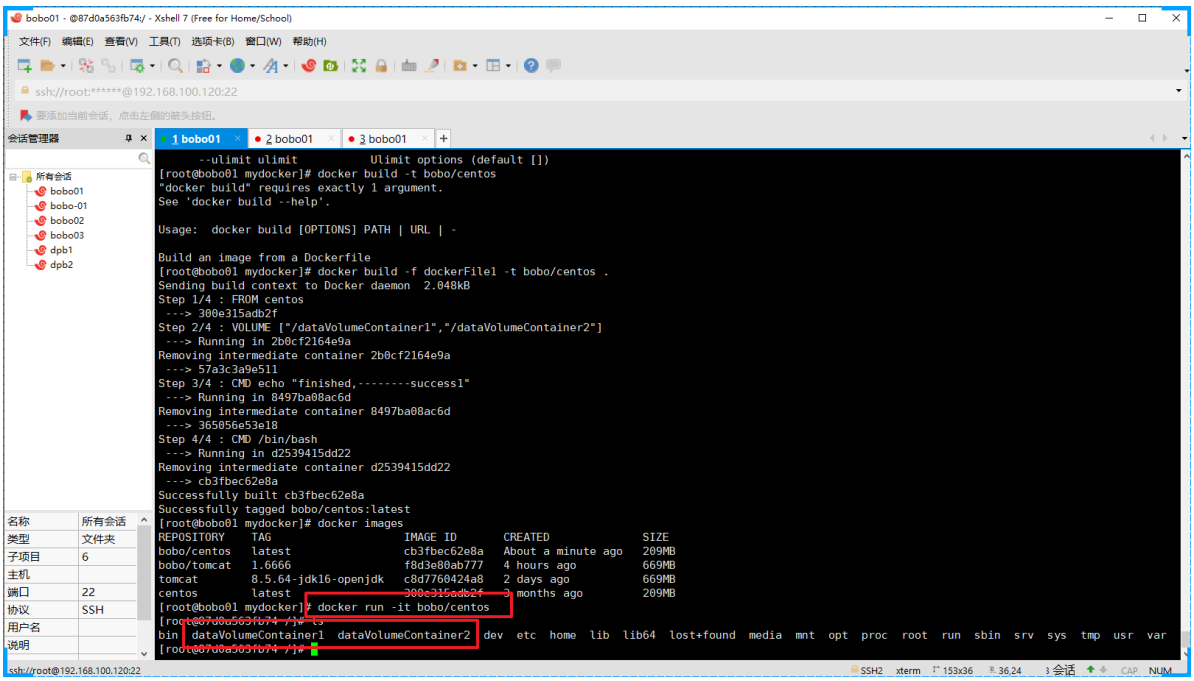
-f DockerFile文件的路径

-t 标签

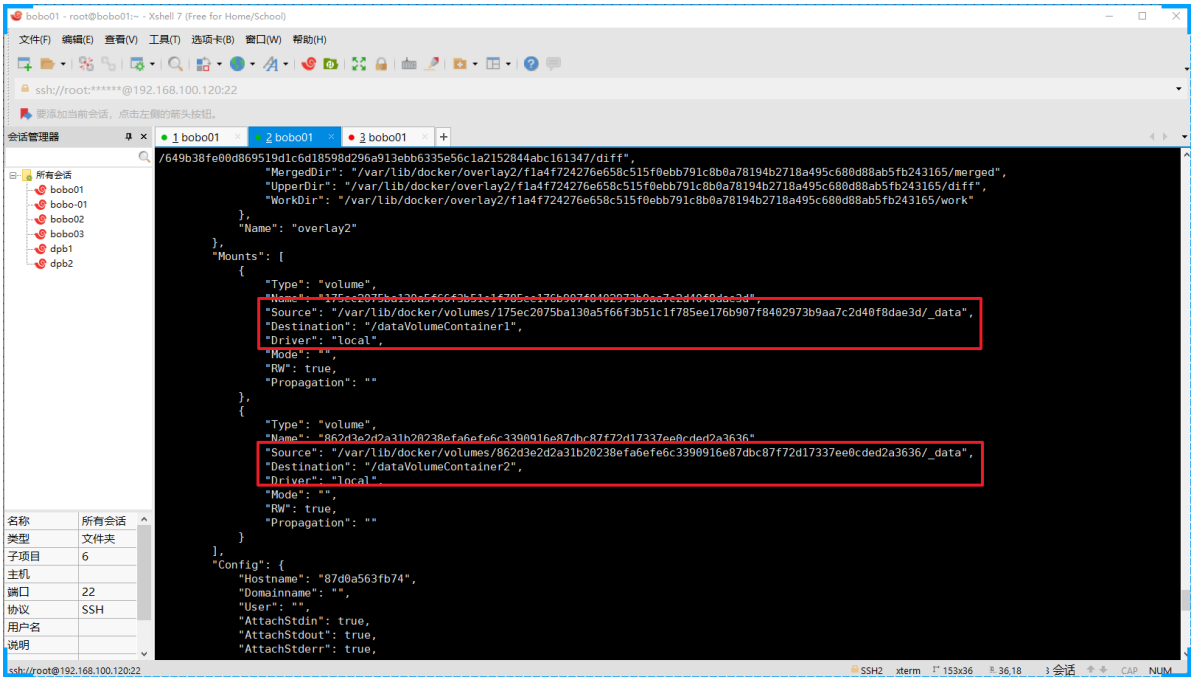
. 当前路径



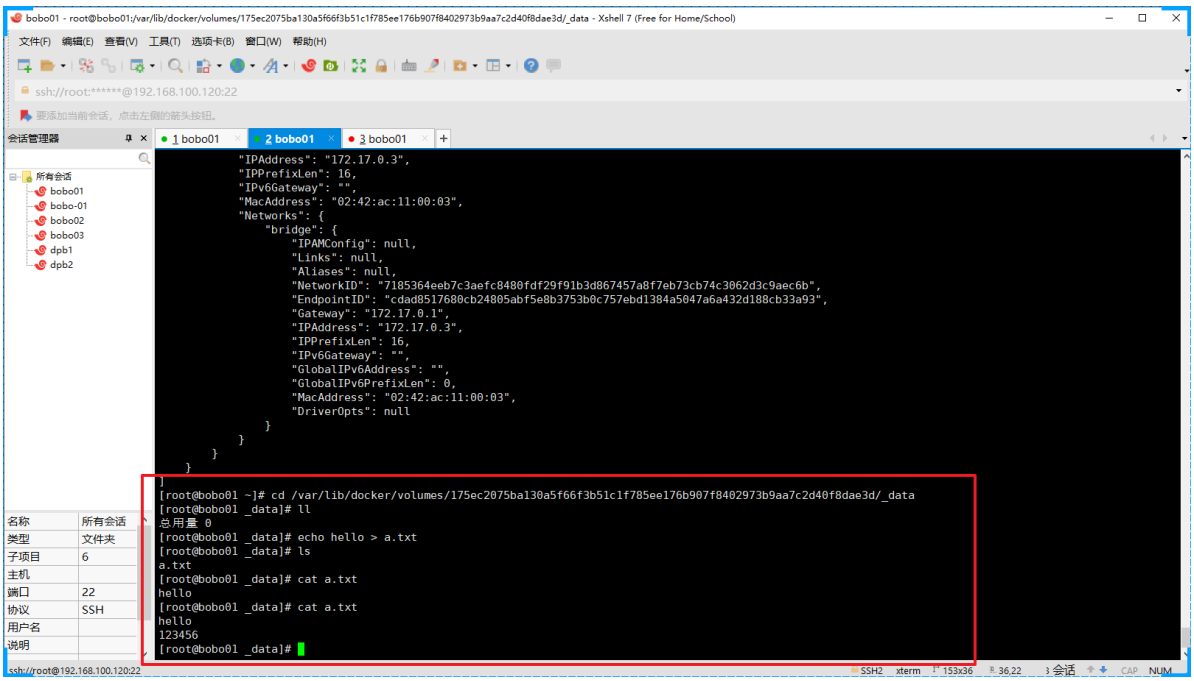
根据新创建的镜像文件创建一个容器，启动后我们可以看到在容器中创建的有对应的目录

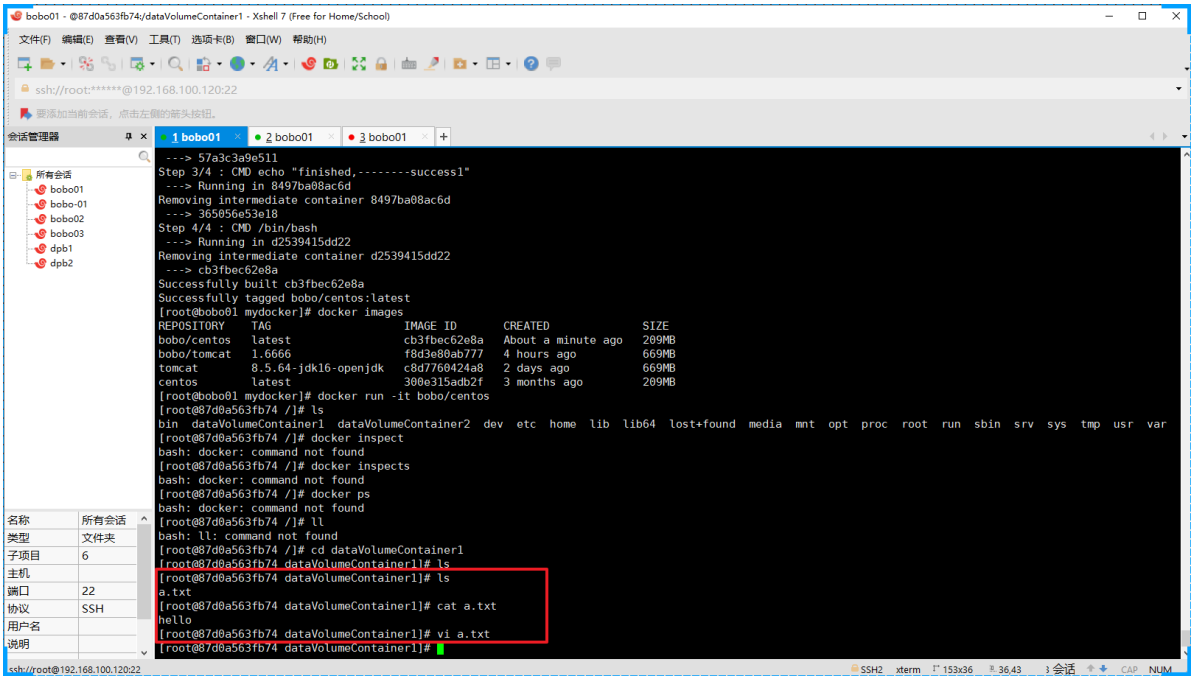


这两个目录和宿主机的映射目录在哪呢？这时我们可以通过 inspect命令查看



验证就只需要在宿主机中创建文件，然后再到容器对应的文件夹中查看即可



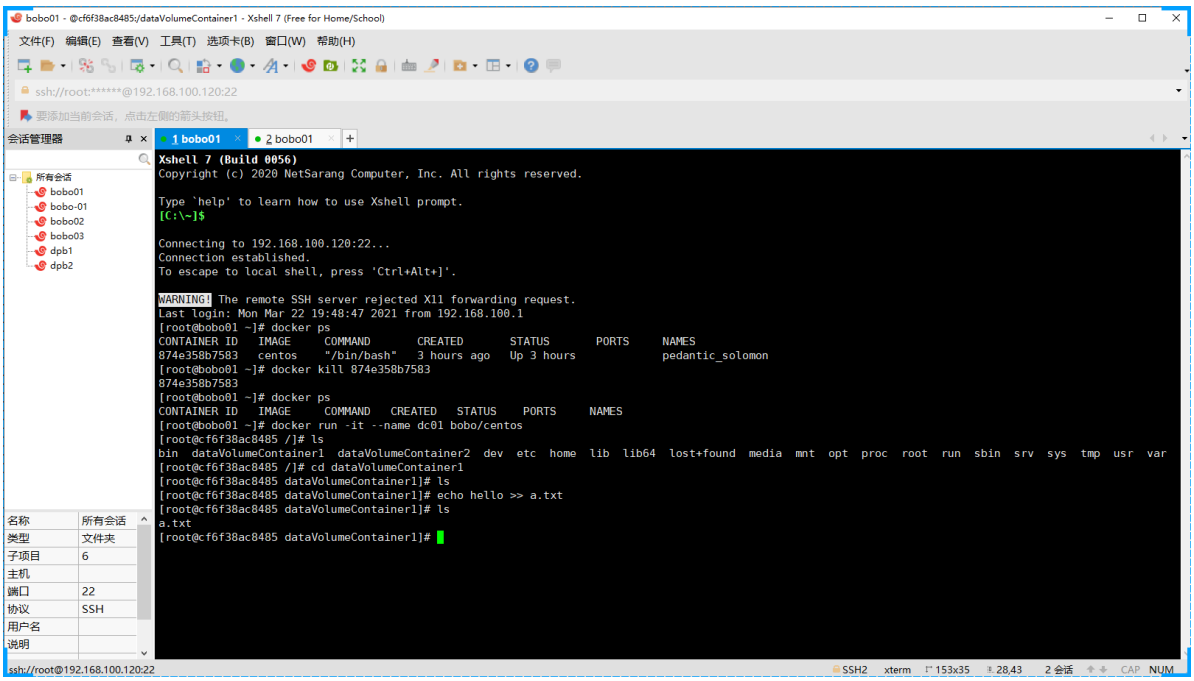


4.2 数据卷容器

命名的容器挂载数据卷，其他容器通过挂载这个容器实现数据共享，挂载数据的容器，称之为数据卷容器。

4.2.1 启动一个父容器

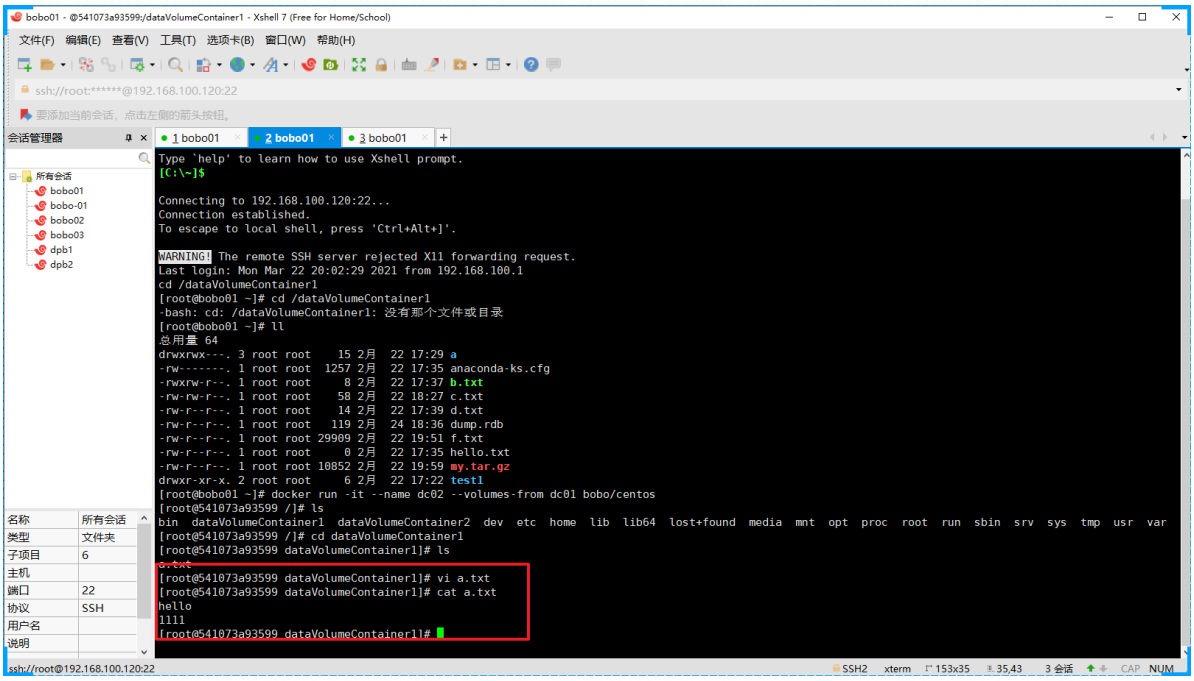
```
docker run -it --name dc01 bobo/centos
```

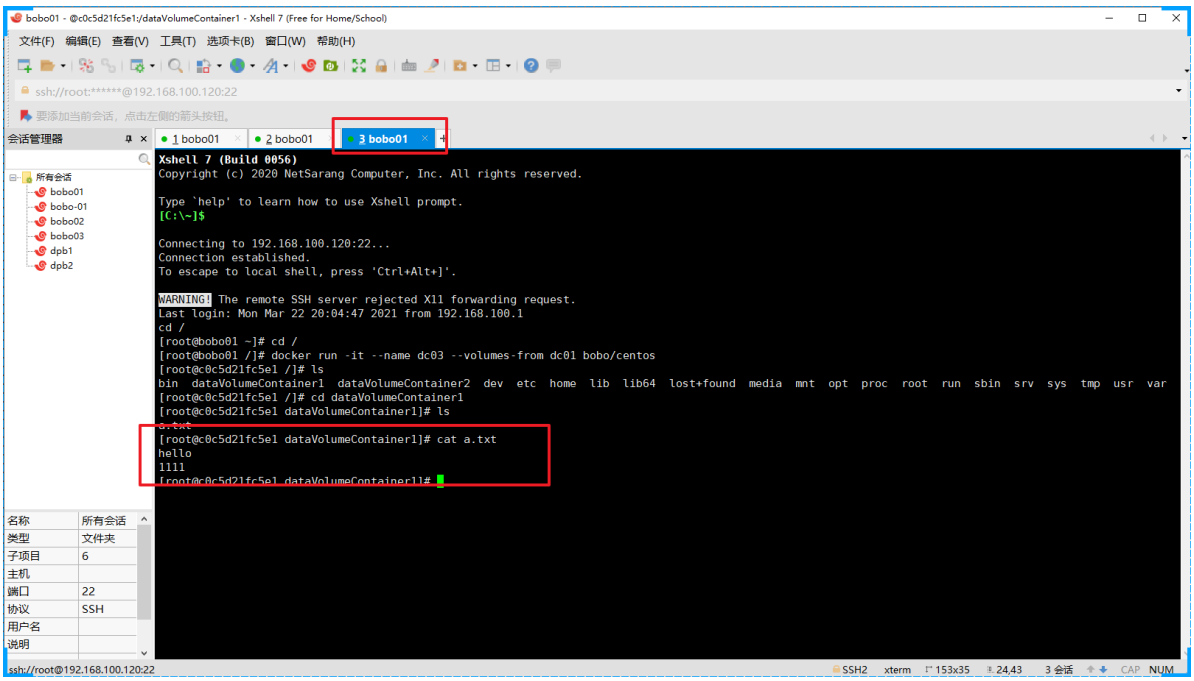


4.2.2 创建两个子容器

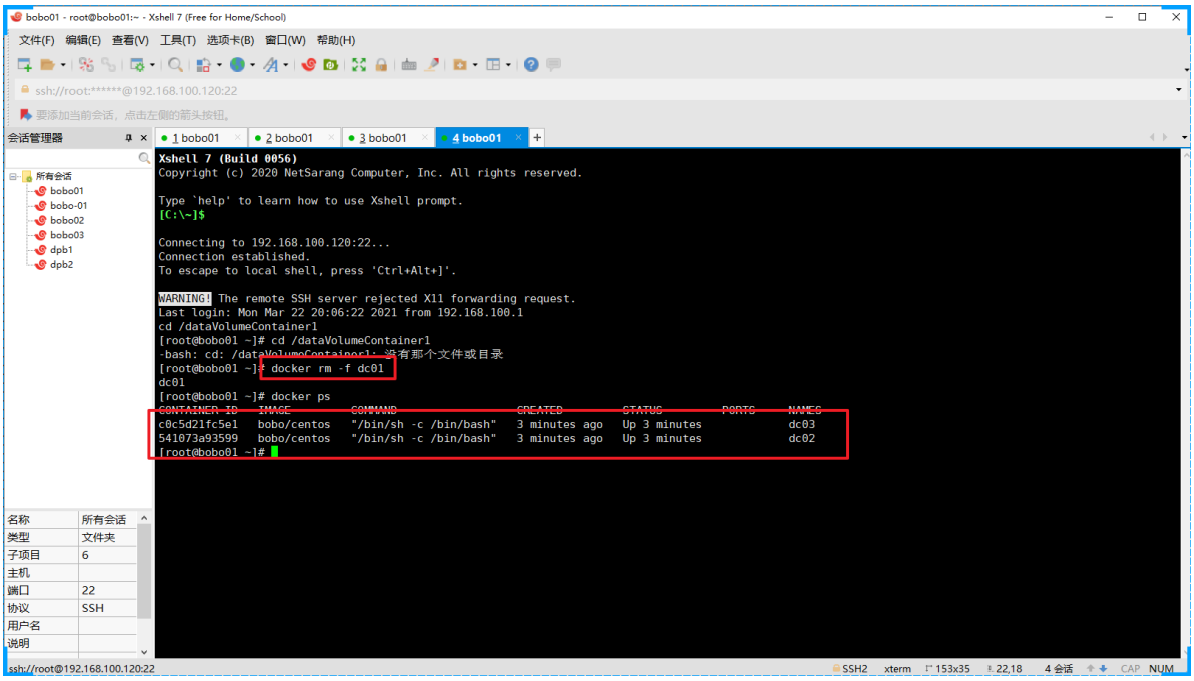
```
docker run -it --name dc02 --volumes-from dc01 bobo/centos
docker run -it --name dc03 --volumes-from dc01 bobo/centos
```

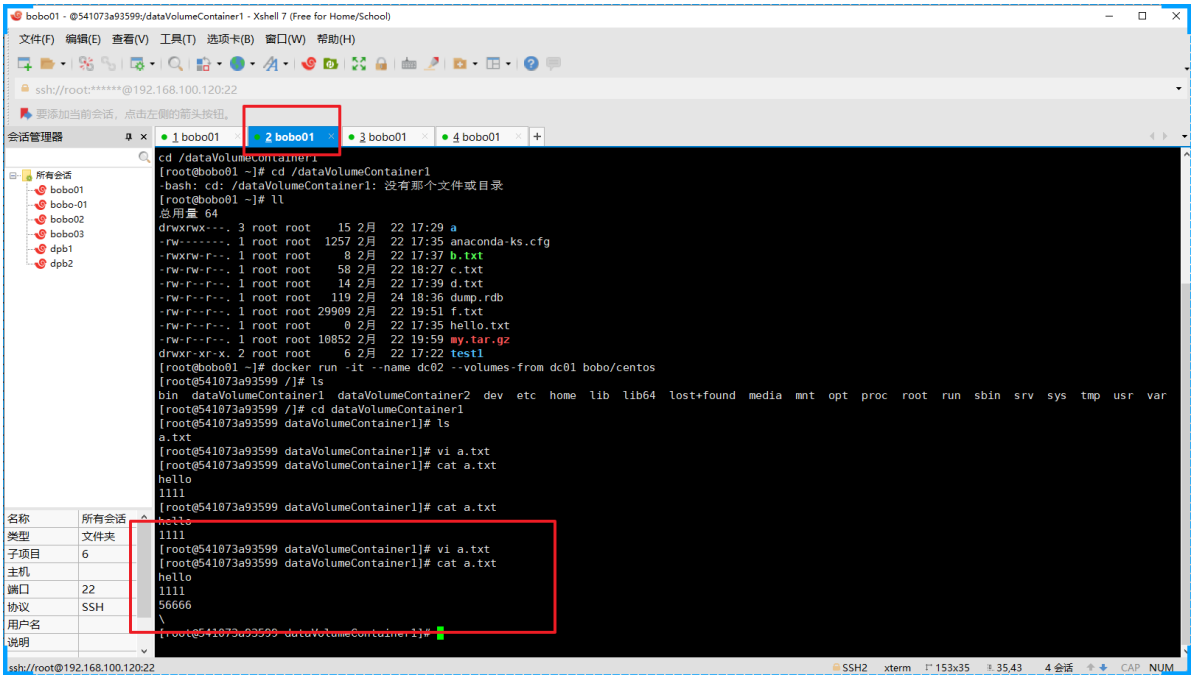
创建了两个子容器后，首先都可以看到dc01中的共享资源。第二个在dc01中修改了共享资源文件后，在两个容器中也是可见的。

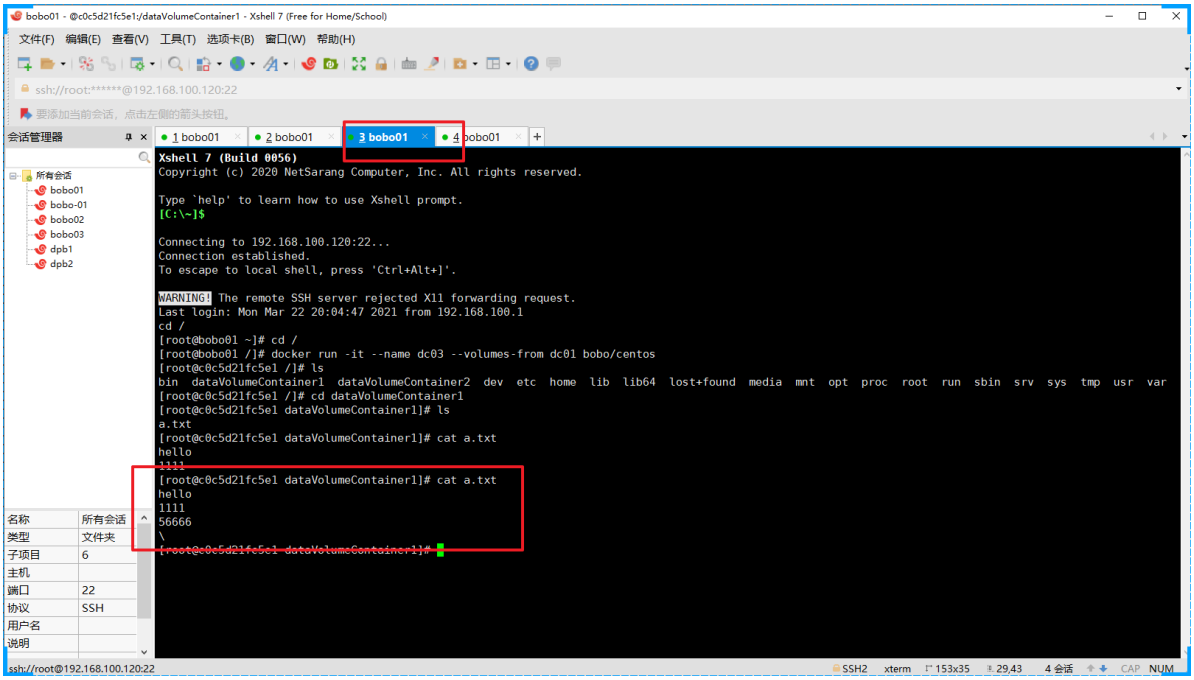




注意，删除dc01后，dc02和dc03之间数据还是共享的



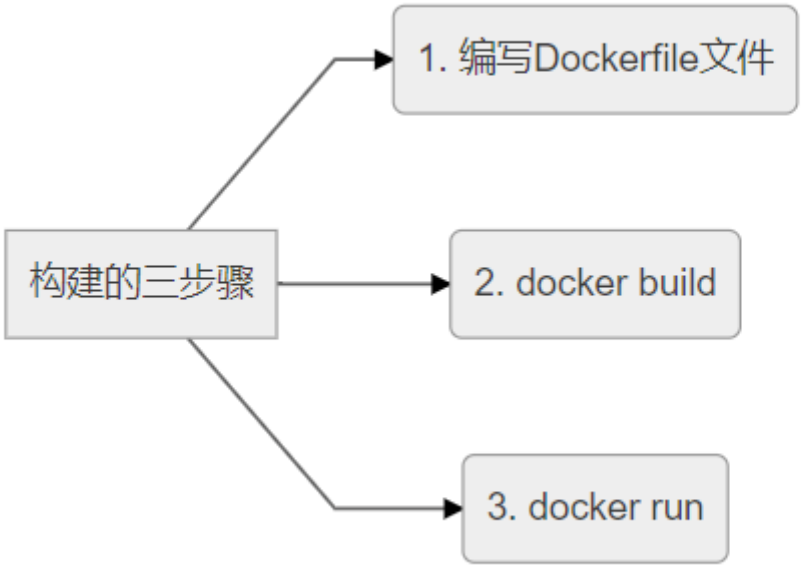




注意：容器之间配置信息的传递，数据卷的生命周期一直持续到没有容器使用它为止。

5 DockerFile

DockerFile是用来构建Docker镜像的`构建文件`，是由一系列`命令`和`参数`构成的`脚本`。

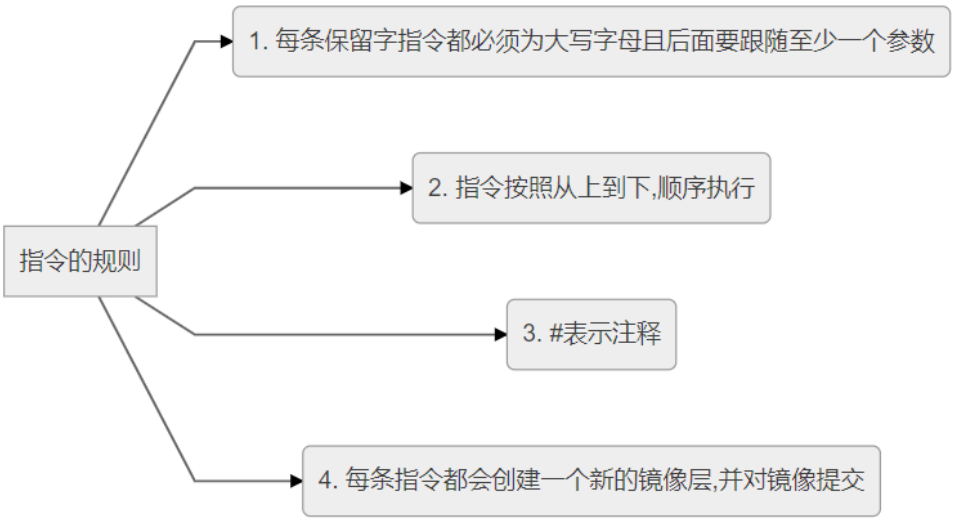



```
FROM scratch
ADD centos-7-x86_64-docker.tar.xz /

LABEL \
    org.label-schema.schema-version="1.0" \
    org.label-schema.name="CentOS Base Image" \
    org.label-schema.vendor="CentOS" \
    org.label-schema.license="GPLv2" \
    org.label-schema.build-date="20201113" \
    org.opencontainers.image.title="CentOS Base Image" \
    org.opencontainers.image.vendor="CentOS" \
    org.opencontainers.image.licenses="GPL-2.0-only" \
    org.opencontainers.image.created="2020-11-13 00:00:00+00:00"

CMD ["/bin/bash"]
```

5.1 DockerFile介绍

5.1.1. 构建过程

Dockerfile中的指令需要满足如下的规则



5.1.2 执行流程

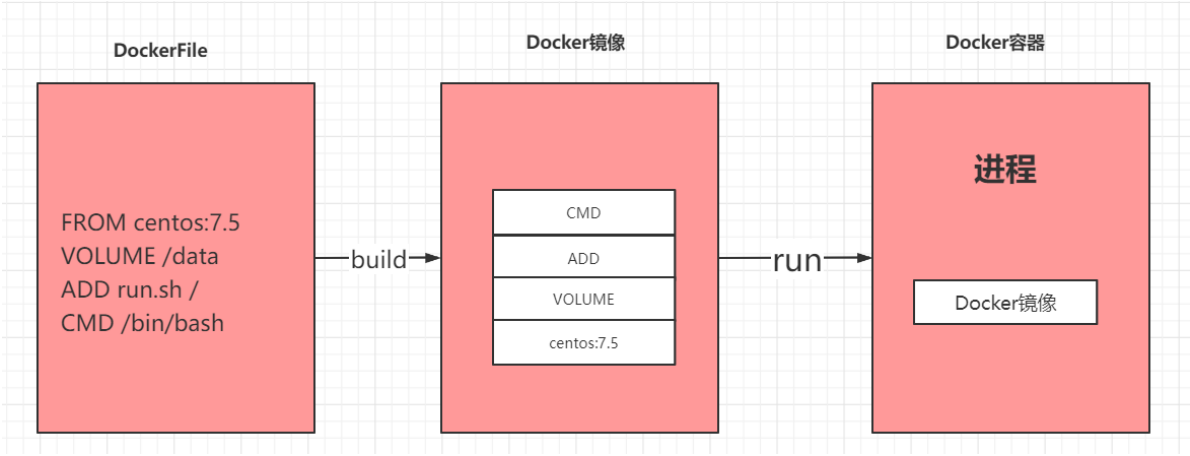
docker执行一个Dockerfile脚本的流程大致如下：

1. docker从基础镜像运行一个容器
2. 执行一条指令并对容器作出修改
3. 执行类似docker commit的操作提交一个新的镜像层
4. docker再基于刚提交的镜像运行一个新的容器
5. 执行dockerfile中的下一条指令直到所有指令都执行完成

从应用软件的角度来看，Dockerfile、Docker镜像与Docker容器分别代表软件的三个不同阶段，

- Dockerfile是软件的原材料
- Docker镜像是软件的交付品
- Docker容器则可以认为是软件的运行态。

Dockerfile面向开发，Docker镜像成为交付标准，Docker容器则涉及部署与运维，三者缺一不可，合力充当Docker体系的基石。



1. Dockerfile，需要定义一个Dockerfile，Dockerfile定义了进程需要的一切东西。Dockerfile涉及的内容包括执行代码或者是文件、环境变量、依赖包、运行时环境、动态链接库、操作系统的发行版、服务进程和内核进程(当应用进程需要和系统服务和内核进程打交道，这时需要考虑如何设计namespace的权限控制)等等;
2. Docker镜像，在用Dockerfile定义一个文件之后，docker build时会产生一个Docker镜像，当运行Docker镜像时，会真正开始提供服务;
3. Docker容器，容器是直接提供服务的。

指令	说明
FROM	基础镜像，当前新镜像是基于哪个镜像的,有继承的意味
MAINTAINER	镜像维护者的姓名和邮箱地址
RUN	容器构建时需要运行的命令
EXPOSE	当前容器对外暴露的端口
WORKDIR	指定在创建容器后，终端默认登录的进来工作目录，一个落脚点
ENV	用来在构建镜像过程中设置环境变量
ADD	将宿主机目录下的文件拷贝进镜像且ADD命令会自动处理URL和解压tar压缩包
COPY	类似ADD，拷贝文件和目录到镜像中。 将从构建上下文目录中<源路径>的文件/目录复制到新的一层的镜像内的<目标路径>位置 COPY src dest COPY ["src","dest"]
VOLUME	容器数据卷，用于数据保存和持久化工作
CMD	指定一个容器启动时要运行的命令 Dockerfile中可以有多个CMD指令，但只有最后一个生效，CMD会被docker run之后的参数替换
ENTRYPOINT	指定一个容器启动时要运行的命令 ENTRYPOINT的目的和CMD一样，都是在指定容器启动程序及参数
ONBUILD	当构建一个被继承的Dockerfile时运行命令,父镜像在被子继承后父镜像的onbuild被触发

```
RUN set -eux; \
    nativeLines="$(catalina.sh configtest 2>&1)"; \
    nativeLines="$(echo "$nativeLines" | grep 'Apache Tomcat Native')"; \
    nativeLines="$(echo "$nativeLines" | sort -u)"; \
    if ! echo "$nativeLines" | grep -E 'INFO: Loaded( APR based)? Apache Tomcat
Native library' >&2; then \
        echo >&2 "$nativeLines"; \
        exit 1; \
    fi

EXPOSE 8080
CMD ["catalina.sh", "run"]

docker run -it -p 7777:8080 tomcat  ls -l

RUN set -eux; \
    nativeLines="$(catalina.sh configtest 2>&1)"; \
    nativeLines="$(echo "$nativeLines" | grep 'Apache Tomcat Native')"; \
    nativeLines="$(echo "$nativeLines" | sort -u)"; \
```

```
    if ! echo "$nativeLines" | grep -E 'INFO: Loaded( APR based)? Apache Tomcat
Native library' >&2; then \
        echo >&2 "$nativeLines"; \
        exit 1; \
    fi

EXPOSE 8080
CMD ["catalina.sh", "run"]
CMD ls -l
```

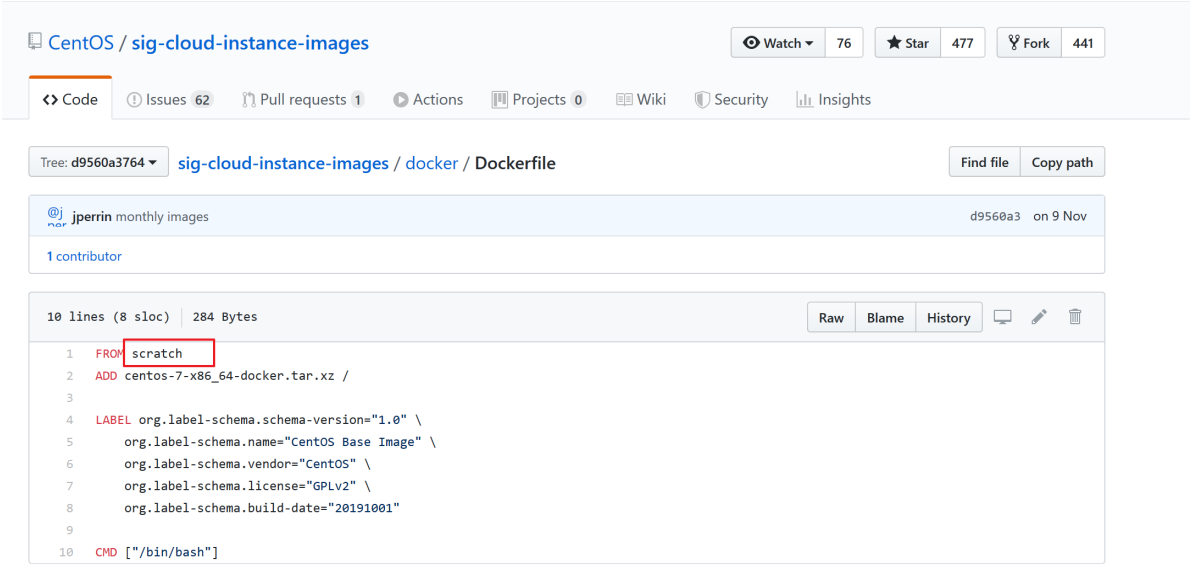
DockerFile命令

BUILD	BOTH	RUN
FROM	WORKDIR	CMD
MAINTAINER	USER	ENV
COPY		EXPOSE
ADD		VOLUME
RUN		ENTRYPOINT
ONBUILD		
.dockerignore		

5.2 DockerFile案例

5.2.1 Base镜像

Docker Hub中99%的镜像都是通过在base镜像中安装和配置需要的软件构建出来的，如下



`scratch`相对于java中的`Object`

5.2.2 自定义镜像mycentos

我们从官方pull下来的`centos`镜像是mini版的，所以不带有`vim`这些基础命令，那我们就来自定义一个镜像，功能比官方下载的强大点，同时运用下各个指令。

5.2.2.1 编写

首先我们来编写对应的Dockerfile文件。内容如下

```
FROM centos
MAINTAINER bobo<dengpbs@163.com>

ENV MYPATH /usr/local
WORKDIR $MYPATH

RUN yum -y install vim

EXPOSE 80

CMD echo $MYPATH
CMD echo "success--------------ok"
CMD /bin/bash
```

5.2.2.2 构建

然后将脚本构建成对应的镜像文件。

```
docker build -f dockerfile名称 -t 新建的镜像名:TAG .
```

```
[root@localhost mydocker]# docker build -f dockercentos -t bbcentos:1.6 .
Sending build context to Docker daemon  2.048kB
Step 1/9 : FROM centos
 ---> 0f3e07c0138f
Step 2/9 : MAINTAINER bobo<dengpbs@163.com>
 ---> Using cache
 ---> 0ce2e17f43b4
Step 3/9 : ENV MYPATH /usr/local
 ---> Using cache
 ---> 904984bd205a
Step 4/9 : WORKDIR $MYPATH
 ---> Using cache
 ---> d1406c95e3c9
Step 5/9 : RUN yum -y install vim    运行
 ---> Using cache
 ---> 524a2b747fb3
Step 6/9 : EXPOSE 80
 ---> Using cache
 ---> b0b2fb9a4135
Step 7/9 : CMD echo $MYPATH
 ---> Using cache
 ---> 94ec7c844101
Step 8/9 : CMD echo "success--------------ok"
 ---> Using cache
 ---> 0e7d765825f6
Step 9/9 : CMD /bin/bash
 ---> Using cache
 ---> 5f1217cdd1f7
Successfully built 5f1217cdd1f7     成功
Successfully tagged bbcentos:1.6
[root@localhost mydocker]#
```

查看镜像

```
[root@localhost mydocker]#
[root@localhost mydocker]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bbcentos            1.6                 5f1217cdd1f7        8 hours ago         286MB
mycentos            1.3                 5f1217cdd1f7        8 hours ago         286MB
bobokaoya/centos    latest              255e261ae0ba        13 hours ago        220MB
bobo/tomcat         1.666               36218cb45e59        27 hours ago        507MB
tomcat              latest              6fa48e047721        12 days ago         507MB
nginx               latest              231d40e811cd        4 weeks ago         126MB
centos              latest              0f3e07c0138f        2 months ago        220MB
hello-world         latest              fce289e99eb9        11 months ago       1.84kB
[root@localhost mydocker]#
```


5.2.2.3 运行

运行镜像文件。

```
docker run -it 新镜像名称:TAG
```

```
tomcat              latest              6fa48e047721        12 days ago         507MB
nginx               latest              231d40e811cd        4 weeks ago         126MB
centos              latest              0f3e07c0138f        2 months ago        220MB
hello-world         latest              fce289e99eb9        11 months ago       1.84kB
[root@localhost mydocker]# docker run -it bbcentos:1.6
[root@4697fbb4b7b6 local]# pwd
/usr/local
[root@4697fbb4b7b6 local]# vim a.txt
[root@4697fbb4b7b6 local]# cat a.txt
1231
[root@4697fbb4b7b6 local]#
```


运行容器后，落脚点是 `/usr/local` 因为我们配置了 `WORKDIR`

5.2.2.4 镜像历史

查看一个镜像文件的变更历史可以使用如下命令:

```
docker history 镜像名
```

```
exit
[root@localhost mydocker]# docker history bbcentos:1.6
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
5f1217cdd1f7        8 hours ago         /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "/bin…    0B
0e7d765825f6        8 hours ago         /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "echo…    0B
94ec7c844101        8 hours ago         /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "echo…    0B
b0b2fb9a4135        8 hours ago         /bin/sh -c #(nop)  EXPOSE 80                     0B
524a2b747fb3        8 hours ago         /bin/sh -c yum -y install vim                    66.8MB
d1406c95e3c9        9 hours ago         /bin/sh -c #(nop) WORKDIR /usr/local             0B
904984bd205a        9 hours ago         /bin/sh -c #(nop)  ENV MYPATH=/usr/local         0B
0ce2e17f43b4        9 hours ago         /bin/sh -c #(nop)  MAINTAINER bobo<dengpbs@1…    0B
0f3e07c0138f        2 months ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]             0B
<missing>           2 months ago        /bin/sh -c #(nop)  LABEL org.label-schema.sc…    0B
<missing>           2 months ago        /bin/sh -c #(nop) ADD file:d6fdacc1972df524a…    220MB
[root@localhost mydocker]#
```


在本例中我们用到了 `FROM` `MAINTAINER` `RUN` `EXPOSE` `ENV` `WORKDIR` 命令

5.2.3.CMD/ENTRYPOINT案例

接下来我们通过案例来看看 `CMD` 和 `ENTRYPOINT` 两个命令的区别，这两个命令的作用都是 `指定一个容器启动时要运行的命令`

5.2.3.1 CMD

Dockerfile中可以有多个CMD指令，但只有最后一个生效，CMD会被docker run之后的参数替换掉，我们通过tomcat的案例来介绍。
正常情况如下

```
docker run -it -p 8888:8080 tomcat
```

```
hello-world        latest              fce289e99eb9        11 months ago       1.84kB
[root@localhost mydocker]# docker run -it -p 8888:8080 tomcat
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat                    正常执行
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr/local/openjdk-8
Using CLASSPATH:       /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat-juli.jar
26-Dec-2019 17:06:27.650 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version name:   Apache Tomcat/8.5.50
26-Dec-2019 17:06:28.042 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:          Dec 7 2019 19:19:46 UTC
26-Dec-2019 17:06:28.043 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version number: 8.5.50.0
26-Dec-2019 17:06:28.044 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:               Linux
26-Dec-2019 17:06:28.045 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:            3.10.0-957.el7.x86_64
26-Dec-2019 17:06:28.045 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture:          amd64
26-Dec-2019 17:06:28.062 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Java Home:             /usr/local/openjdk-8/jre
26-Dec-2019 17:06:28.063 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Version:           1.8.0_232-b09
26-Dec-2019 17:06:28.063 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Vendor:            Oracle Corporation
26-Dec-2019 17:06:28.064 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_BASE:         /usr/local/tomcat
26-Dec-2019 17:06:28.064 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_HOME:         /usr/local/tomcat
26-Dec-2019 17:06:28.065 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.util.logging.config.file=
operties
26-Dec-2019 17:06:28.080 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.util.logging.manager=org.
26-Dec-2019 17:06:28.081 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djdk.tls.ephemeralDHKeySize=204
26-Dec-2019 17:06:28.081 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.protocol.handler.pkgs=org
26-Dec-2019 17:06:28.082 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dorg.apache.catalina.security.S
26-Dec-2019 17:06:28.082 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dignore.endorsed.dirs=
26-Dec-2019 17:06:28.083 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dcatalina.base=/usr/local/tomca
26-Dec-2019 17:06:28.093 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dcatalina.home=/usr/local/tomca
26-Dec-2019 17:06:28.094 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.io.tmpdir=/usr/local/tomc
26-Dec-2019 17:06:28.094 INFO [main] org.apache.catalina.core.AprLifecycleListener.lifecycleEvent Loaded APR based Apache Tomcat Native library [
26-Dec-2019 17:06:28.094 INFO [main] org.apache.catalina.core.AprLifecycleListener.lifecycleEvent APR capabilities: IPv6 [true], sendfile [true],
```

但是当我们在 执行命令后添加参数的话，如下

```
5-Dec-2019 17:07:42.565 INFO [Thread-5] org.apache.coyote.AbstractProtocol.destroy Destroying ProtocolHandl
5-Dec-2019 17:07:42.571 INFO [Thread-5] org.apache.coyote.AbstractProtocol.destroy Destroying ProtocolHandl
root@localhost mydocker]# docker run -it -p 8888:8080 tomcat ls -l
otal 124
rw-r--r--. 1 root root  19318 Dec  7 19:23 BUILDING.txt
rw-r--r--. 1 root root   5408 Dec  7 19:23 CONTRIBUTING.md
rw-r--r--. 1 root root  57011 Dec  7 19:23 LICENSE
rw-r--r--. 1 root root   1726 Dec  7 19:23 NOTICE
rw-r--r--. 1 root root   3255 Dec  7 19:23 README.md
rw-r--r--. 1 root root   7136 Dec  7 19:23 RELEASE-NOTES
rw-r--r--. 1 root root  16262 Dec  7 19:23 RUNNING.txt
rwxr-xr-x. 2 root root   4096 Dec 13 23:52 bin
rwxr-sr-x. 2 root root    238 Dec  7 19:23 conf
rwxr-sr-x. 2 root staff    78 Dec 13 23:52 include
rwxr-xr-x. 2 root root   4096 Dec 13 23:52 lib
rwxrwxrwx. 2 root root      6 Dec  7 19:19 logs
rwxr-sr-x. 3 root staff   151 Dec 13 23:52 native-jni-lib
rwxrwxrwx. 2 root root     30 Dec 13 23:52 temp
rwxr-xr-x. 7 root root     81 Dec  7 19:21 webapps
rwxrwxrwx. 2 root root      6 Dec  7 19:19 work
root@localhost mydocker]#
```

添加了参数，tomcat就不能正常启动了

原因是我们先看Tomact对应的 Dockerfile文件

```
112             chmod 777 logs temp work
113
114     # verify Tomcat Native is working properly
115     RUN set -e \
116             && nativeLines="$(catalina.sh configtest 2>&1)" \
117             && nativeLines="$(echo "$nativeLines" | grep 'Apache Tomcat Native')" \
118             && nativeLines="$(echo "$nativeLines" | sort -u)" \
119             && if ! echo "$nativeLines" | grep 'INFO: Loaded APR based Apache Tomcat Native library'
120                     echo >&2 "$nativeLines"; \
121                     exit 1; \
122             fi
123
124     EXPOSE 8080
125     CMD ["catalina.sh", "run"]
```

最后是要执行 tomcat启动脚本的

然而我们的run命令 把Dockerfile中的最后的CMD命令覆盖了~~

5.2.3.2 ENTRYPOINT

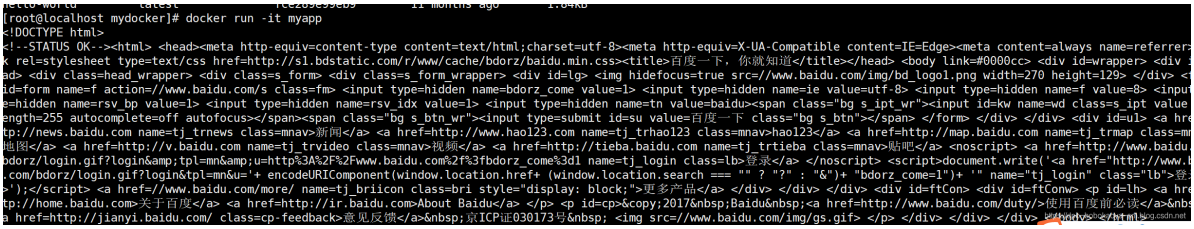
有别于`CMD`命令，`ENTRYPOINT`命令是在`docker run`之后的参数会被当做参数传递给ENTRYPOINT，之后形成新的组合命令。我们通过`curl`指令来介绍这个案例。
Dockerfile文件如下:

```
FROM centos

RUN yum install -y curl

ENTRYPOINT [ "curl", "-s", "http://www.baidu.com" ]
```

构建

```
-rw-r--r--. 1 root root  90 12月 27 01:19 dockerfile1
[root@localhost mydocker]# docker build -f dockerfile1 -t myapp .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM centos
 ---> 0f3e07c0138f
Step 2/3 : RUN yum install -y curl
 ---> Running in 051e8fb6c8af
k^HCentOS-8 - AppStream                          762 kB/s | 6.3 MB     00:08
CentOS-8 - Base                                  700 kB/s | 7.9 MB     00:11
CentOS-8 - Extras                                432  B/s | 2.1 kB     00:04
Package curl-7.61.1-8.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
Removing intermediate container 051e8fb6c8af
 ---> 2574bc0ffe41
Step 3/3 : ENTRYPOINT [ "curl", "-s", "http://www.baidu.com" ]
 ---> Running in 3e5db12279b2
Removing intermediate container 3e5db12279b2
 ---> e09385ee1c02
Successfully built e09385ee1c02
Successfully tagged myapp:latest
[root@localhost mydocker]#
```


正常run



加`-i`参数 查看响应报文头

通过这个例子 可以看到`ENTRYPOINT`不会覆盖，而是组合成了一个新的命令。

5.2.4.自定义Tomcat

最后我们通过自定义一个`tomcat`镜像来介绍下`ADD`和`COPY`这两个命令的区别。

5.2.4.1 创建个tomcat目录

```
[root@localhost mydocker]# mkdir tomcat
[root@localhost mydocker]# ll
总用量 8
-rw-r--r--. 1 root root 186 12月 26 16:23 dockercentos
-rw-r--r--. 1 root root  90 12月 27 01:19 dockerfile1
drwxr-xr-x. 2 root root   6 12月 27 01:28 tomcat
[root@localhost mydocker]# cd tomcat/
[root@localhost tomcat]# ll
总用量 0
[root@localhost tomcat]# pwd
/mydocker/tomcat
[root@localhost tomcat]#
```


5.2.4.2 添加一个文件

在当前目录下创建一个 `hello.txt` 文件，作用是 `COPY` 到容器中

```
/mydocker/tomcat
[root@localhost tomcat]# vim hello.txt
[root@localhost tomcat]# cat hello.txt
hello ...
[root@localhost tomcat]#
```

5.2.4.3 拷贝相关软件

准备对应的 `jdk` 和 `tomcat` 的压缩文件。

```
总用量 196480
-rw-r--r--. 1 root root   9368661 11月 13 2017 apache-tomcat-8.0.47.tar.gz
-rw-r--r--. 1 root root        11 12月 27 01:29 hello.txt
-rw-r--r--. 1 root root 191817140 3月  15 2019 jdk-8u201-linux-x64.tar.gz
[root@localhost tomcat]# 1
```

5.2.4.4 创建Dockerfile文件

创建对应的Dockerfile文件，如下：

```
FROM         centos
MAINTAINER    bobo<dengpbs@163.com>
#把宿主机当前上下文的hello.txt拷贝到容器/usr/local/路径下
COPY readme.txt /usr/local/helloincontainer.txt
#把java与tomcat添加到容器中
ADD jdk-8u73-linux-x64.tar.gz /usr/local/
ADD apache-tomcat-8.5.71.tar.gz /usr/local/
#安装vim编辑器
RUN yum -y install vim
#设置工作访问时候的WORKDIR路径，登录落脚点
ENV MYPATH /usr/local
WORKDIR $MYPATH
#配置java与tomcat环境变量
ENV JAVA_HOME /usr/local/jdk1.8.0_73
ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
ENV CATALINA_HOME /usr/local/apache-tomcat-8.5.71
ENV CATALINA_BASE /usr/local/apache-tomcat-8.5.71
```

```
ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/lib:$CATALINA_HOME/bin
#容器运行时监听的端口
EXPOSE  8080
#启动时运行tomcat
# ENTRYPOINT ["/usr/local/apache-tomcat-8.0.47/bin/startup.sh" ]
# CMD ["/usr/local/apache-tomcat-8.0.47/bin/catalina.sh","run"]
CMD /usr/local/apache-tomcat-8.5.71/bin/startup.sh && tail -F /usr/local/apache-tomcat-8.0.47/bin/logs/catalina.out
```

5.2.4.5 构建

```
docker build -f dockerfile -t bobotomcat .
```

```
-rw-r--r--. 1 root root 191817140 3月  15 2019 jdk-8u201-linux-x64.tar.gz
[root@localhost tomcat]# docker build -f dockerfile -t bobotomcat .
Sending build context to Docker daemon  201.2MB
Step 1/15 : FROM         centos
 ---> 0f3e07c0138f
Step 2/15 : MAINTAINER    bobo<dengpbs@163.com>
 ---> Using cache
 ---> 0ce2e17f43b4
Step 3/15 : COPY hello.txt /usr/local/helloincontainer.txt
 ---> ce10ef1d8a3b
Step 4/15 : ADD jdk-8u201-linux-x64.tar.gz /usr/local/
 ---> c6d59ad8e4fd
Step 5/15 : ADD apache-tomcat-8.0.47.tar.gz /usr/local/
 ---> 7db913996020
Step 6/15 : RUN yum -y install vim
 ---> Running in 6e1bd33626bc
CentOS-8 - AppStream                            746 kB/s | 6.3 MB     00:08
CentOS-8 - Base                                 789 kB/s | 7.9 MB     00:10
CentOS-8 - Extras                               266  B/s | 2.1 kB     00:08
Dependencies resolved.
================================================================================
 Package             Arch        Version                  Repository      Size
================================================================================
Installing:
 vim-enhanced        x86_64      2:8.0.1763-10.el8        AppStream      1.4 M
Installing dependencies:
 gpm-libs            x86_64      1.20.7-15.el8            AppStream       39 k
 vim-common          x86_64      2:8.0.1763-10.el8        AppStream      6.3 M
 vim-filesystem      noarch      2:8.0.1763-10.el8        AppStream       48 k
 which               x86_64      2.21-10.el8              BaseOS          49 k

Transaction Summary
================================================================================
Install  5 Packages

Total download size: 7.8 M
Installed size: 30 M
Downloading Packages:
(1/5): gpm-libs-1.20.7-15.el8.x86_64.rpm         87 kB/s |  39 kB     00:00
```

```
 ---> fc038cdb82d7
Step 7/15 : ENV MYPATH /usr/local
 ---> Running in 5d5a5f80ef48
Removing intermediate container 5d5a5f80ef48
 ---> 006667a92ec0
Step 8/15 : WORKDIR $MYPATH
 ---> Running in 91bbd26ea64c
Removing intermediate container 91bbd26ea64c
 ---> b6459d66d8ad
Step 9/15 : ENV JAVA_HOME /usr/local/jdk1.8.0_201
 ---> Running in 38335163a71a
Removing intermediate container 38335163a71a
 ---> 1826f8e02215
Step 10/15 : ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
 ---> Running in 28cb6dcb4080
Removing intermediate container 28cb6dcb4080
 ---> b88dd27ddda0
Step 11/15 : ENV CATALINA_HOME /usr/local/apache-tomcat-8.0.47
 ---> Running in df46439cdb15
Removing intermediate container df46439cdb15
 ---> 13abd0e71aae
Step 12/15 : ENV CATALINA_BASE /usr/local/apache-tomcat-8.0.47
 ---> Running in c2f9bf55250d
Removing intermediate container c2f9bf55250d
 ---> 2f469af5d58b
Step 13/15 : ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/lib:$CATALINA_HOME/bin
 ---> Running in 082e3abe20ac
Removing intermediate container 082e3abe20ac
 ---> 3fb349e3a3ae
Step 14/15 : EXPOSE  8080
 ---> Running in 9718998e3810
Removing intermediate container 9718998e3810
 ---> 8fec4737afbf
Step 15/15 : CMD /usr/local/apache-tomcat-8.0.47/bin/startup.sh && tail -F /usr/local/apache-tomcat-8.0.47/bin/logs/catalina.out
 ---> Running in c253a91d0de6
Removing intermediate container c253a91d0de6
 ---> b855070eb393
Successfully built b855070eb393
Successfully tagged bobotomcat:latest
[root@localhost tomcat]#
```



```
[root@localhost tomcat]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bobotomcat          latest              b855070eb393        3 minutes ago       697MB
myapp               latest              e09385ee1c02        25 minutes ago      249MB
bbcentos            1.6                 5f1217cdd1f7        9 hours ago         286MB
mycentos            1.3                 5f1217cdd1f7        9 hours ago         286MB
bobokaoya/centos    latest              255e261ae0ba        14 hours ago        220MB
bobo/tomcat         1.666               36218cb45e59        28 hours ago        507MB
tomcat              latest              6fa48e047721        12 days ago         507MB
nginx               latest              231d40e811cd        4 weeks ago         126MB
centos              latest              0f3e07c0138f        2 months ago        220MB
hello-world         latest              fce289e99eb9        11 months ago       1.84kB
```


构建成功。

5.2.4.6 run

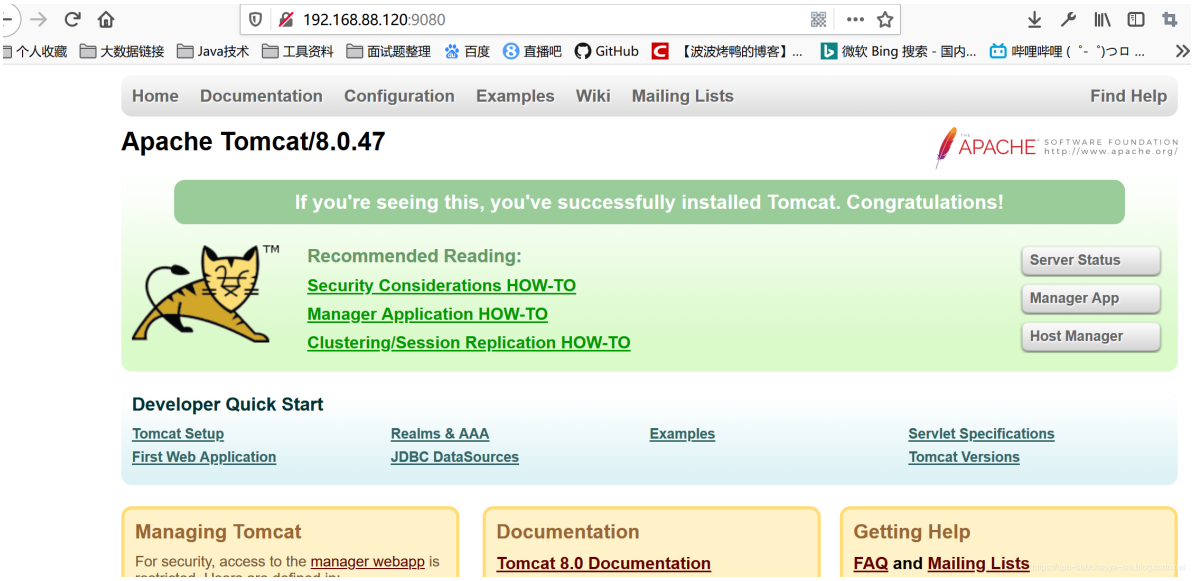
构建成功后，我们就可以运行了，命令如下:

```
docker run -it -p 9080:8080 --name mytomcat8.5 -v
/root/dockerfile/tomcat/test:/usr/local/apache-tomcat-8.5.71/webapps/test -v
/root/dockerfile/tomcat/tomcatlogs/:/usr/local/apache-tomcat-8.5.71/logs --
privileged=true mytomcat8.5
```

注意：我们在启动的时候指定了相关的 数据卷

```
hello-world         latest              fce289e99eb9        11 months ago       1.84kB
[root@localhost tomcat]# docker run -d -p 9080:8080 --name mytomcat -v /mydocker/tomcat/test:/usr/local/apache-tomcat-8.0.47/webapps/test -v /mydocker/tomcat/tomcatlogs/:/usr/local/apache-tomcat-8.0.47/logs --privileged=true bobotomcat
e70889e77ad8a663abcde9d5c44bac2599ccf41842b1f09dae58c802f6ba710a
[root@localhost tomcat]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
e70889e77ad8        bobotomcat          "/bin/sh -c '/usr/lo…"   About a minute ago   Up 58 seconds       0.0.0.0:9080->8080/tcp   mytomcat
[root@localhost tomcat]#
```

5.2.4.7 验证

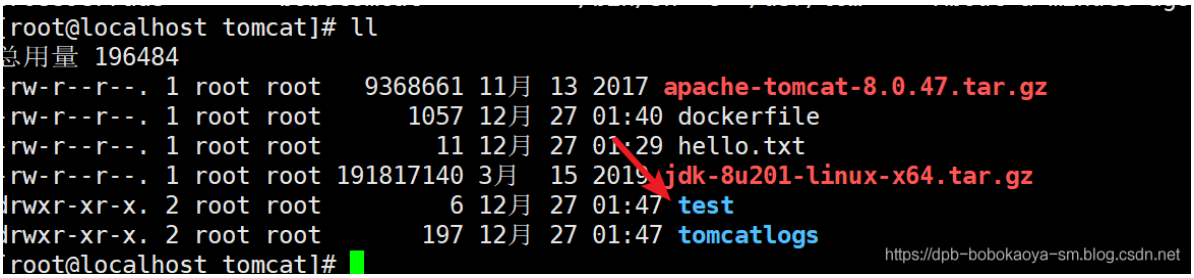


5.2.4.8 部署web项目

既然我们已经部署好了我们自己的tomcat容器，而且也设置了对应的数据卷，那么我们来实际部署一个web案例来看看

5.2.4.8.1 web.xml文件

我们在test目录下创建`WEB-INF`目录，然后创建`web.xml`文件，



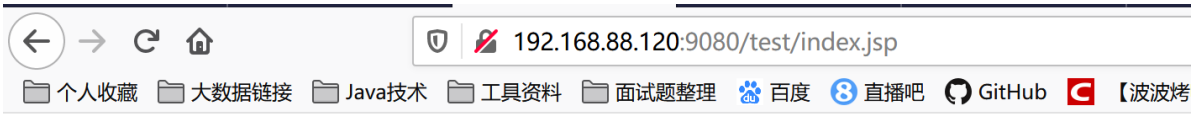

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xmlns="http://java.sun.com/xml/ns/javaee"

  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"

  id="WebApp_ID" version="2.5">



  <display-name>test</display-name>



</web-app>
```

5.2.4.8.2 index.jsp文件

然后创建一个简单的jsp文件即可

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <title>Insert title here</title>

  </head>

  <body>

    -----------welcome------------

    <%="i am in docker tomcat self "%>

    <br>

    <br>

    <% System.out.println("==============docker tomcat self");%>

  </body>

</html>
```

5.2.4.8.3 重启容器访问即可



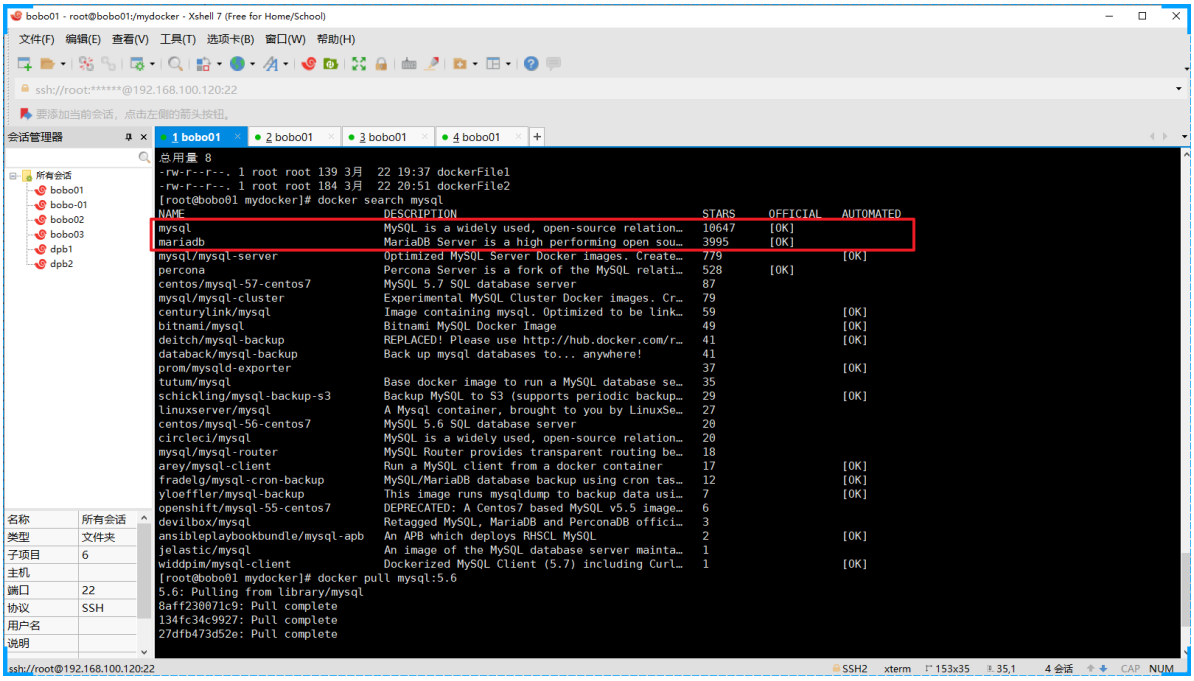
----------welcome------------ i am in docker tomcat self

6.Docker常用软件安装

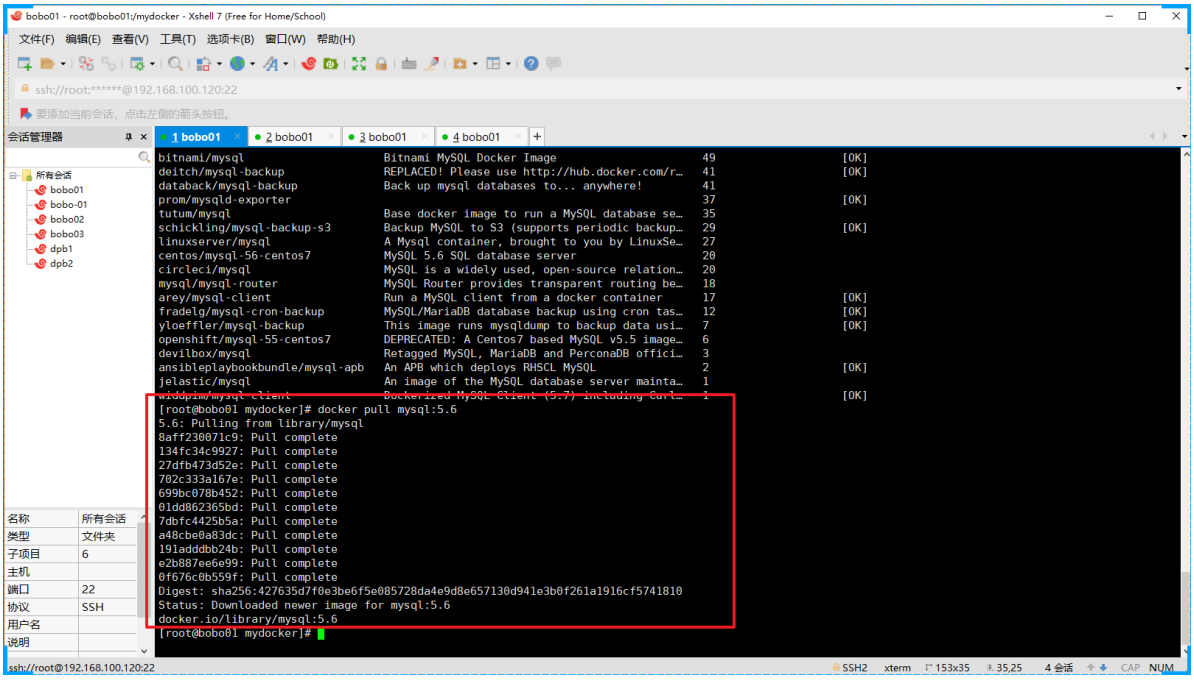
6.1. MySQL的安装

search命令查询

```
docker search mysql
```



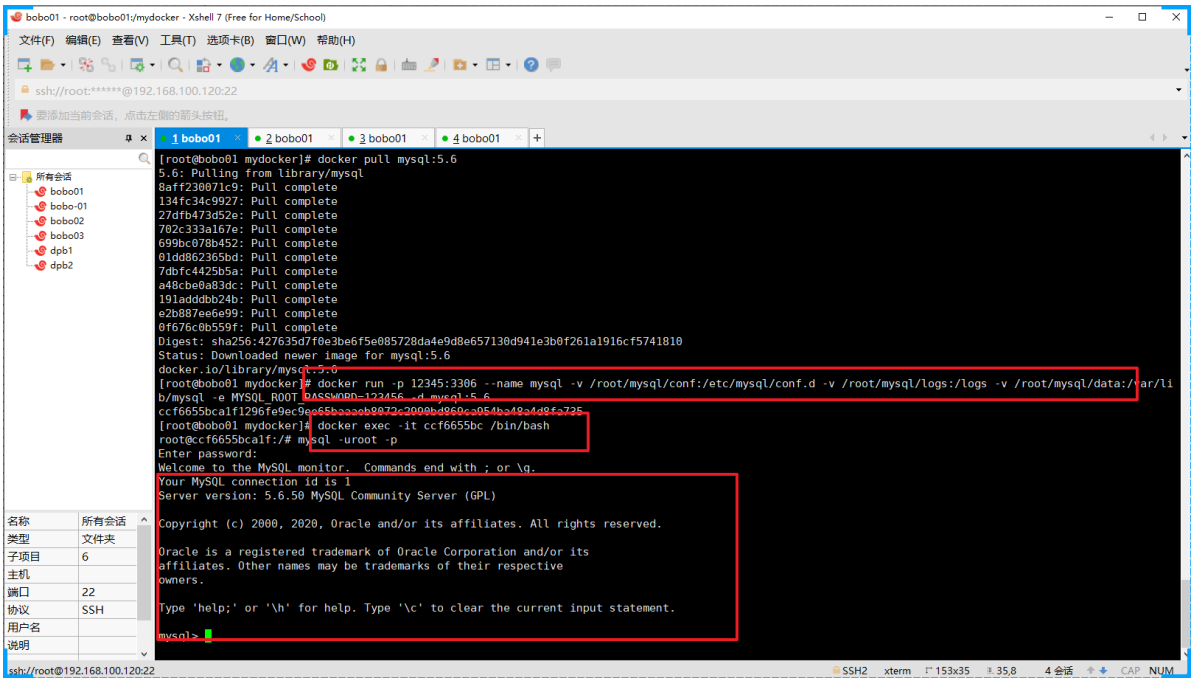
然后下载对应的mysql镜像



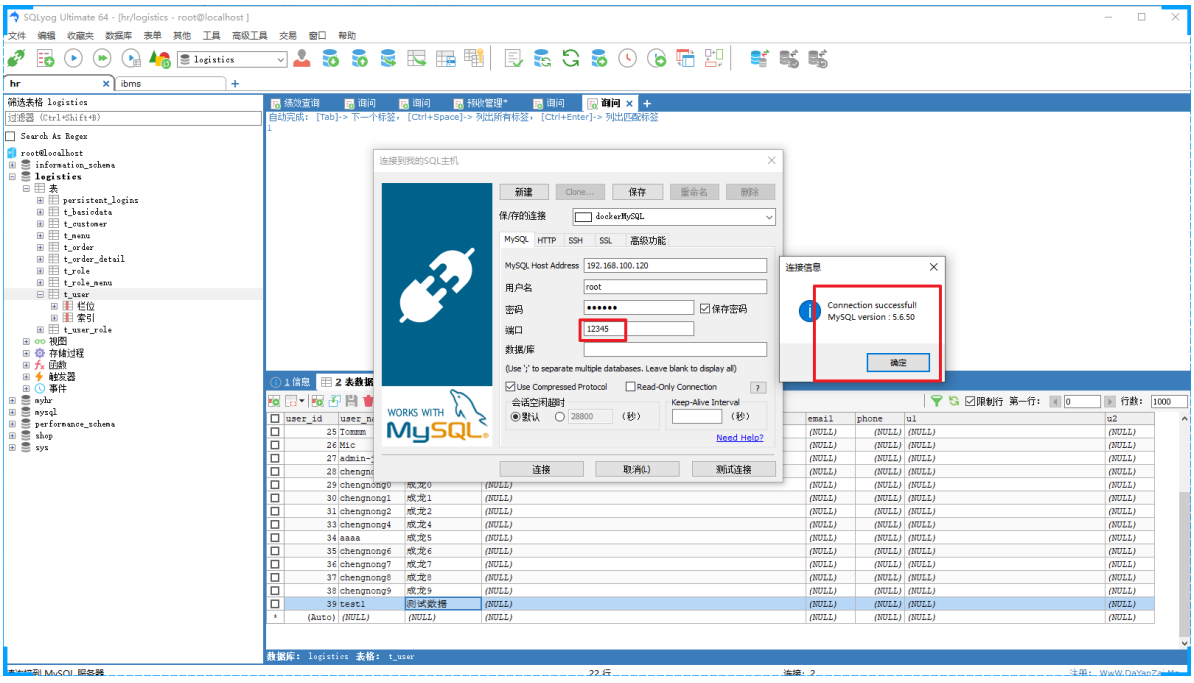
构建容器

```
docker run -p 12345:3306 --name mysql -v /root/mysql/conf:/etc/mysql/conf.d -v
/root/mysql/logs:/logs -v /root/mysql/data:/var/lib/mysql -e
MYSQL_ROOT_PASSWORD=123456 -d mysql:5.6
```

进入容器中查看

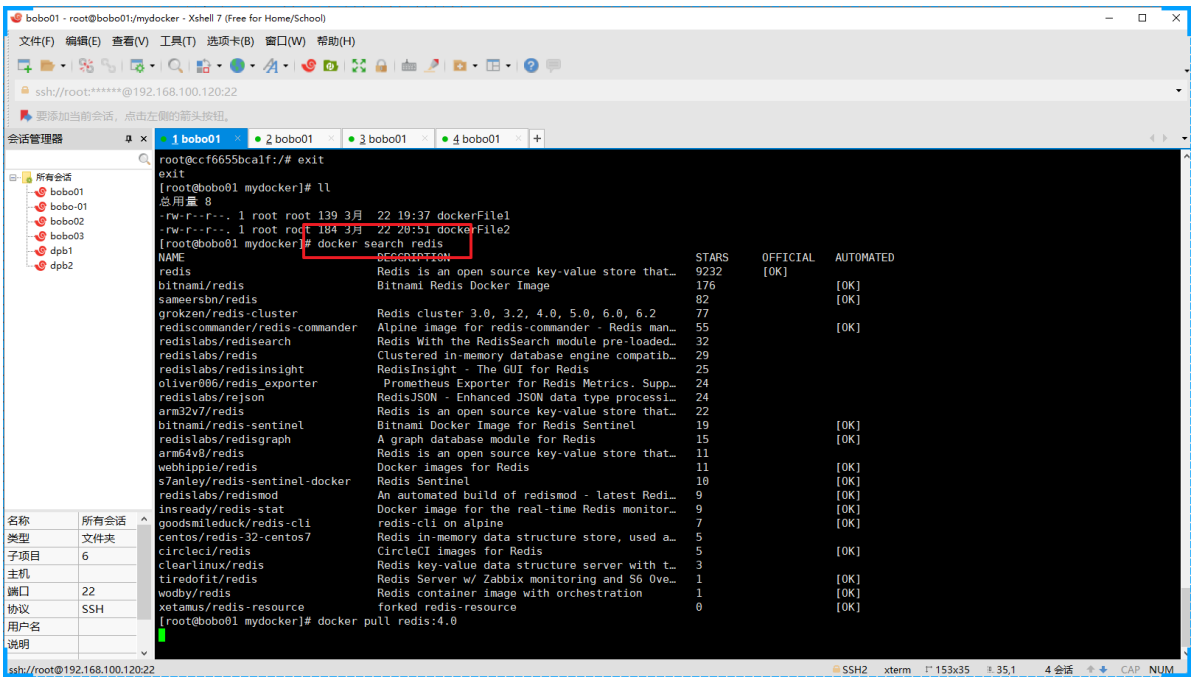


我们也可以在Windows平台中通过MySQL数据库客户端来连接容器中的数据库

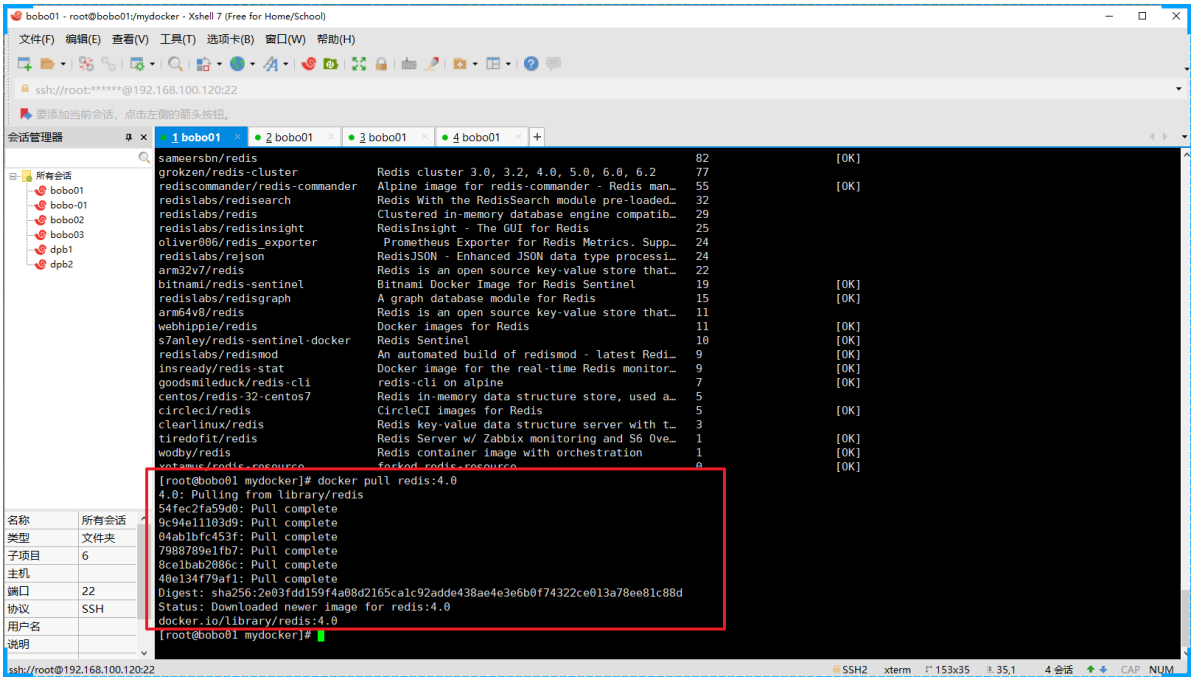


6.2. Redis的安装

搜索Redis

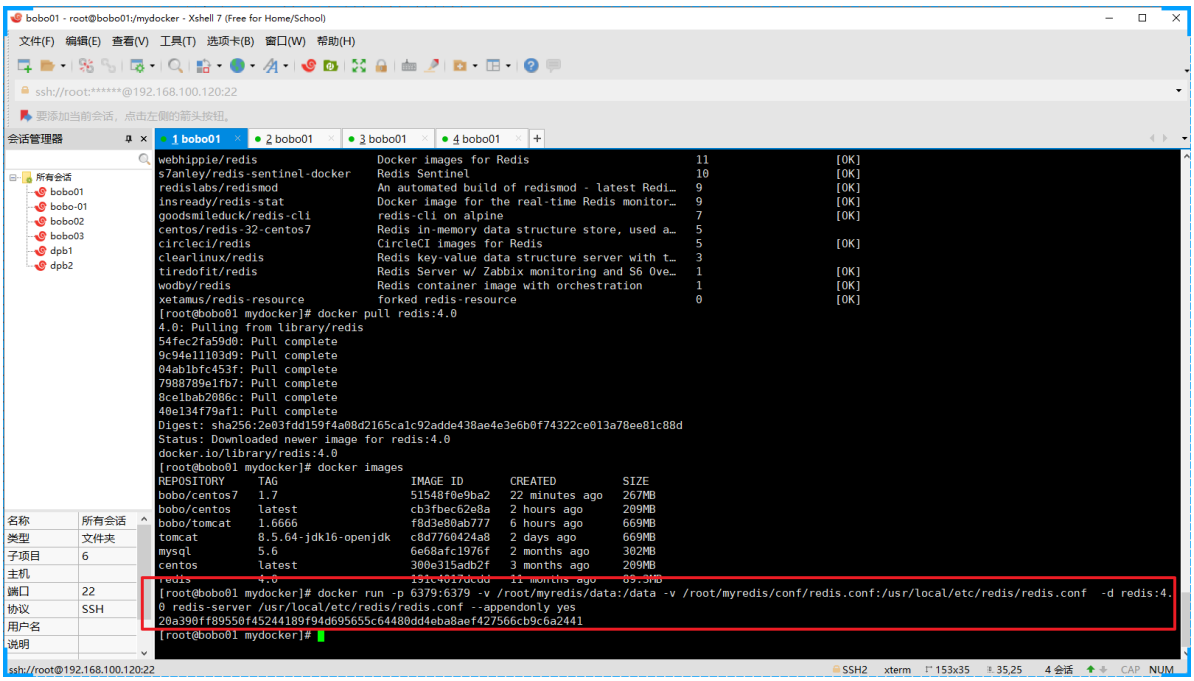


下载对应镜像文件



创建并启动容器

```
docker run -p 6379:6379 -v /root/myredis/data:/data -v
/root/myredis/conf/redis.conf:/usr/local/etc/redis/redis.conf  -d redis:4.0
redis-server /usr/local/etc/redis/redis.conf --appendonly yes
```



连接命令

```
[root@localhost tomcat]# docker exec -it 4f0 redis-cli
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379> set name bobo
OK
127.0.0.1:6379> set age 18
OK
127.0.0.1:6379> get name
"bobo"
127.0.0.1:6379> get age
```

redis的配置文件

```
port 6379
tcp-backlog 511
timeout 0
tcp-keepalive 300
supervised no
pidfile /var/run/redis_6379.pid


loglevel notice
logfile ""

databases 16

save 120 1
save 300 10
save 60 10000
```

```
stop-writes-on-bgsave-error yes
rdbcompression yes

rdbchecksum yes
dbfilename dump.rdb

dir ./
slave-serve-stale-data yes

slave-read-only yes
repl-diskless-sync no
repl-diskless-sync-delay 5
repl-disable-tcp-nodelay no

slave-priority 100


appendonly no
appendfilename "appendonly.aof"

appendfsync everysec
no-appendfsync-on-rewrite no

auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb


lua-time-limit 5000
slowlog-log-slower-than 10000
slowlog-max-len 128

notify-keyspace-events ""

hash-max-ziplist-entries 512
hash-max-ziplist-value 64

list-max-ziplist-size -2
list-compress-depth 0

set-max-intset-entries 512

zset-max-ziplist-entries 128
zset-max-ziplist-value 64

hll-sparse-max-bytes 3000
activerehashing yes

client-output-buffer-limit normal 0 0 0
client-output-buffer-limit slave 256mb 64mb 60
client-output-buffer-limit pubsub 32mb 8mb 60

hz 10
aof-rewrite-incremental-fsync yes
```

测试连接

 image-20210323151232702

二、Docker高级篇

1.Docker网络介绍

Docker是基于Linux Kernel的namespace，CGroups,UnionFileSystem等技术封装成的一种自定义容器格式，从而提供了一套虚拟运行环境。

> namespace: 用来做隔离的，比如 pid[进程]、net【网络】、mnt【挂载点】
>
> CGroups：Controller Groups 用来做资源限制，比如内存和CPU等
>
> Union File Systems：用来做Image和Container分层

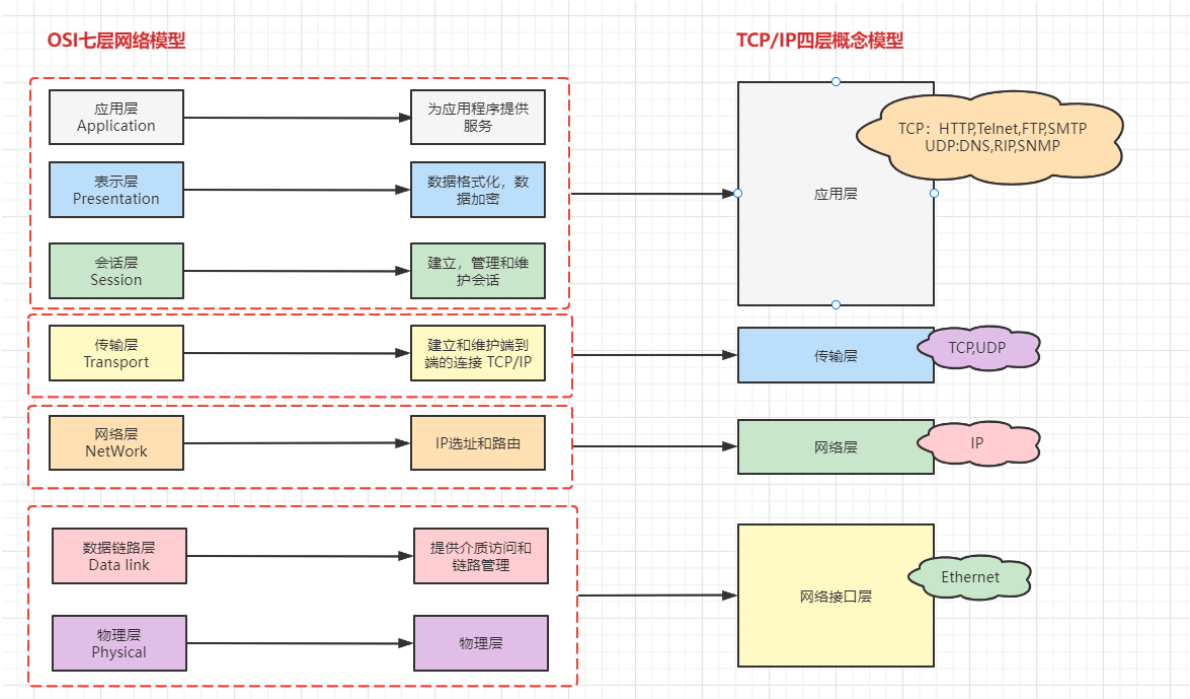
1.1 计算机网络模型

Docker网络官网：https://docs.docker.com/network/。

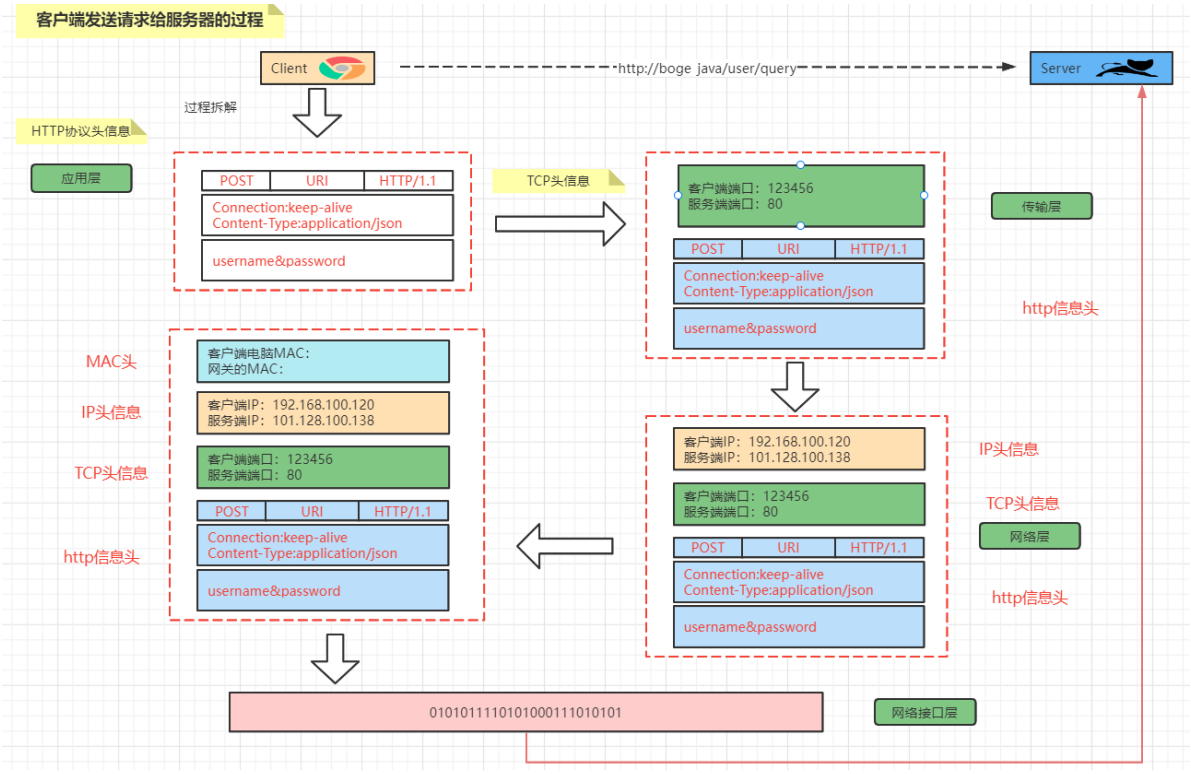
OSI：开放系统互联参考模型(Open System Interconnect)

TCP/IP:传输控制协议/网际协议(Transmission Control/Internet Protocol),是指能够在多个不同网络间实现信息传输的协议簇。TCP/IP协议不仅仅指的是TCP 和IP两个协议，而是指一个由FTP、SMTP、TCP、UDP、IP等协议构成的协议簇， 只是因为在TCP/IP协议中TCP协议和IP协议最具代表性，所以被称为TCP/IP协议。

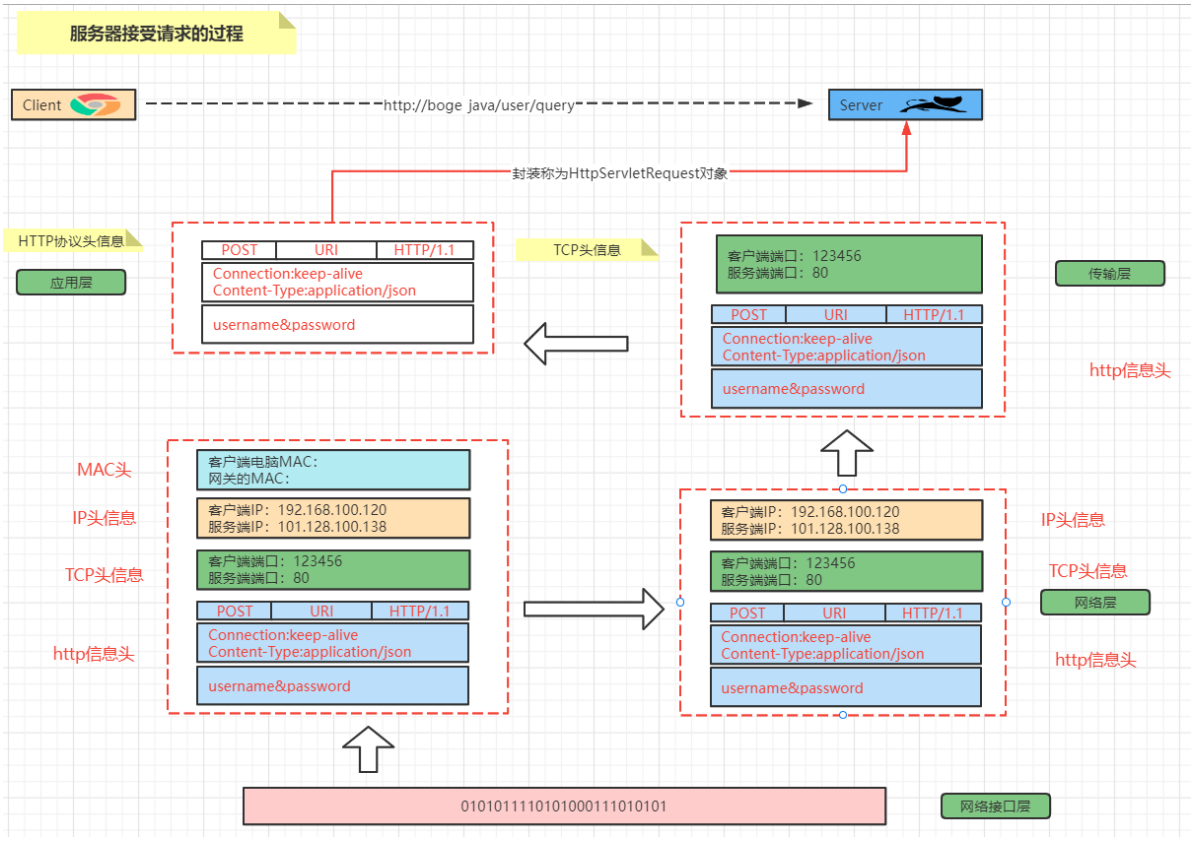
分层思想：分层的基本想法是每一层都在它的下层提供的服务基础上提供更高级的增值服务，而最高层提供能运行分布式应用程序的服务



客户端发送请求：



服务端接受请求：



1.2 Liunx中网卡

1.2.1 查看网卡信息

查看网卡的命令:ip a

```
[vagrant@localhost ~]$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
       valid_lft 85987sec preferred_lft 85987sec
    inet6 fe80::5054:ff:fe4d:77d3/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:6e:31:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.10/24 brd 192.168.56.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe6e:3145/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:bf:79:9f:de brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

通过ip a 可以看到当前的centos中有的4个网卡信息作用分别是

名称	作用
lo	本地网卡【lo 是 loopback 的缩写，也就是环回的意思，linux系统默认会有一块名为 lo 的环回网络接口】
eth0	连接网络的网卡
eth1	和宿主机通信的网卡
docker0	docker的网卡

ip link show:

```
[vagrant@localhost ~]$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 08:00:27:6e:31:45 brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN mode DEFAULT group default
    link/ether 02:42:bf:79:9f:de brd ff:ff:ff:ff:ff:ff
```

以文件的形式查看网卡：ls /sys/class/net

```
[vagrant@localhost ~]$ ls /sys/class/net
docker0  eth0  eth1  lo
```

1.2.2 配置文件

在Linux中网卡对应的其实就是文件，所以找到对应的网卡文件即可，存放的路径

```
[vagrant@localhost network-scripts]$ cd /etc/sysconfig/network-scripts/
[vagrant@localhost network-scripts]$ ls
ifcfg-eth0   ifdown-eth   ifdown-ppp       ifdown-tunnel  ifup-ippp   ifup-post
ifup-TeamPort      network-functions-ipv6
ifcfg-eth1   ifdown-ippp  ifdown-routes    ifup           ifup-ipv6   ifup-ppp
  ifup-tunnel
ifcfg-lo     ifdown-ipv6  ifdown-sit       ifup-aliases   ifup-isdn   ifup-routes
ifup-wireless
ifdown       ifdown-isdn  ifdown-Team      ifup-bnep      ifup-plip   ifup-sit
  init.ipv6-global
ifdown-bnep  ifdown-post  ifdown-TeamPort  ifup-eth       ifup-plusb  ifup-Team
network-functions
```

1.2.3 网卡操作

网卡中增加ip地址

```
[root@localhost ~]# ip addr add 192.168.100.120/24 dev eth0
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
       valid_lft 84918sec preferred_lft 84918sec
    inet 192.168.100.120/24 scope global eth0  #### 增加了一个IP地址
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe4d:77d3/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:6e:31:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.10/24 brd 192.168.56.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe6e:3145/64 scope link
       valid_lft forever preferred_lft forever
```

```
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:bf:79:9f:de brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

删除IP地址: ip addr delete 192.168.100.120/24 dev eth0

```
[root@localhost ~]# ip addr delete 192.168.100.120/24 dev eth0
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
       valid_lft 84847sec preferred_lft 84847sec
    inet6 fe80::5054:ff:fe4d:77d3/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:6e:31:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.10/24 brd 192.168.56.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe6e:3145/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    lik/ether 02:42:bf:79:9f:de brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

1.2.4 网卡信息解析

状态：UP/DOWN/UNKOWN等

link/ether：MAC地址

inet：绑定的IP地址

1.3 Network Namespace

Network Namespace 是实现网络虚拟化的重要功能，它能创建多个隔离的网络空间，它们有独自的网络栈信息。不管是虚拟机还是容器，运行的时候仿佛自己就在独立的网络中。

1.3.1 Network Namespce 实战

添加一个namespace

```
ip netns add ns1
```

查看当前具有的namespace

```
ip netns list
```

```
[root@localhost ~]# ip netns add ns1
[root@localhost ~]# ip netns list
ns1
```

删除namespace

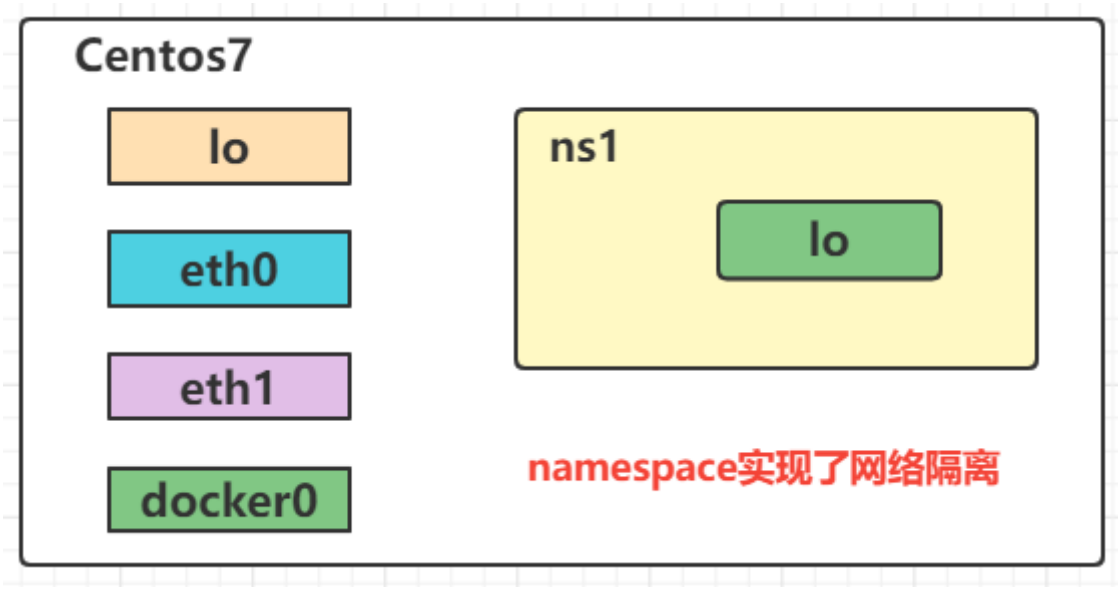
```
ip netns delete ns1
```

```
[root@localhost ~]# ip netns add ns1
[root@localhost ~]# ip netns list
ns1
[root@localhost ~]# ip netns delete ns1
[root@localhost ~]# ip netns list
[root@localhost ~]#
```

查看namespace【ns1】的网卡情况

```
ip netns exec ns1 ip a
```

```
[root@localhost ~]# ip netns exec ns1 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```



启动网络状态

```
ip netns exec ns1 ifup lo
```

```
[root@localhost ~]# ip netns exec ns1 ip link show
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
[root@localhost ~]# ip netns exec ns1 ifup lo
[root@localhost ~]# ip netns exec ns1 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
[root@localhost ~]#
```

关掉网络状态

```
[root@localhost ~]# ip netns exec ns1 ifdown lo
[root@localhost ~]# ip netns exec ns1 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noqueue state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

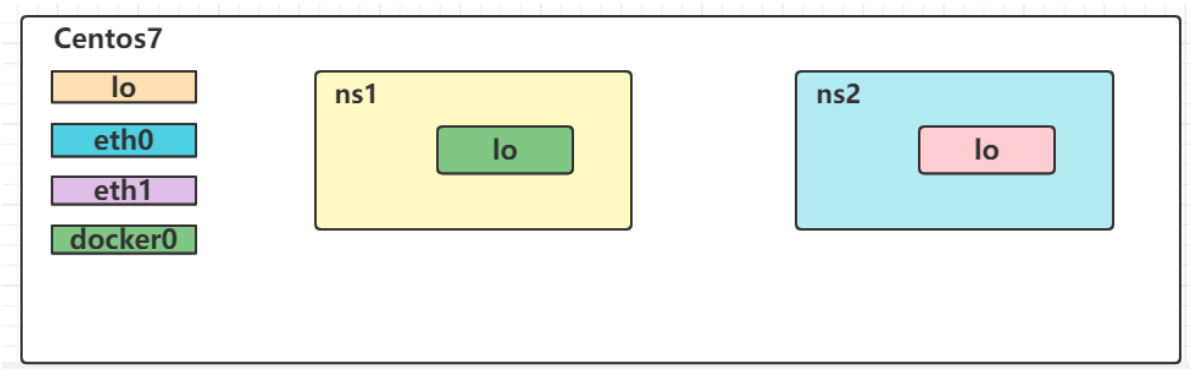
还可以通过 link 来设置状态

```
[root@localhost ~]# ip netns exec ns1 ip link set lo up
[root@localhost ~]# ip netns exec ns1 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
[root@localhost ~]# ip netns exec ns1 ip link set lo down
[root@localhost ~]# ip netns exec ns1 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noqueue state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
[root@localhost ~]#
```

再次添加一个namespace【ns2】

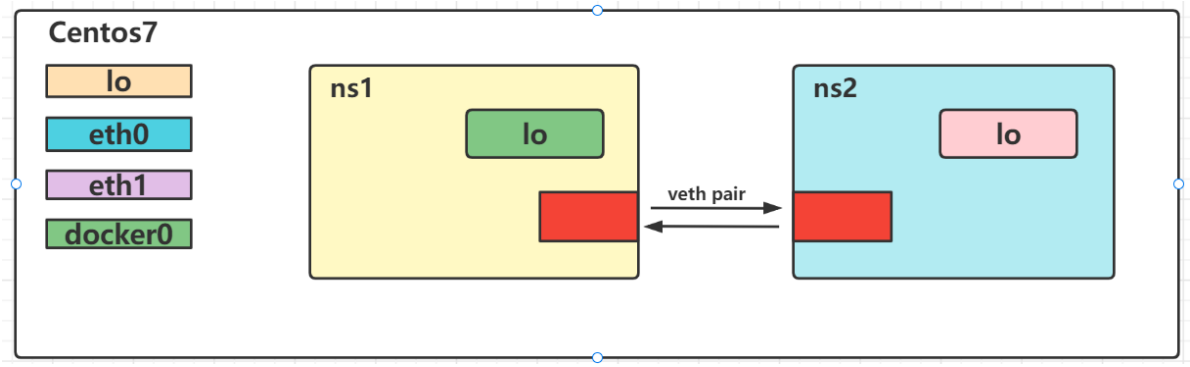
```
[root@localhost ~]# ip netns add ns2
[root@localhost ~]# ip netns list
ns2
ns1
```

现在要实现两个namespace的通信



要实现两个network namespace的通信，我们需要实现到的技术是：

veth pair：Virtual Ethernet Pair，是一个成对的端口，可以实现上述功能



创建一对link，也就是接下来要通过veth pair连接的link

```
ip link add veth-ns1 type veth peer name veth-ns2
```

然后在宿主机中就会多出一对网卡信息

```
[root@localhost ~]# ip link add veth-ns1 type veth peer name veth-ns2
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
       valid_lft 81116sec preferred_lft 81116sec
    inet6 fe80::5054:ff:fe4d:77d3/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:6e:31:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.10/24 brd 192.168.56.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe6e:3145/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:bf:79:9f:de brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
5: veth-ns2@veth-ns1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 7e:f8:18:5a:ef:1f brd ff:ff:ff:ff:ff:ff
6: veth-ns1@veth-ns2: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 7e:bb:ee:13:a2:9a brd ff:ff:ff:ff:ff:ff
[root@localhost ~]#
```

然后将创建好的 veth-ns1交给namespace1，把veth-ns2交给namespace2

```
ip link set veth-ns1 netns ns1
ip link set veth-ns2 netns ns2
```

```
    link/ether 7e:bb:ee:13:a2:9a brd ff:ff:ff:ff:ff:ff
[root@localhost ~]# ip link set veth-ns1 netns ns1
[root@localhost ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:6e:31:45 brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:bf:79:9f:de brd ff:ff:ff:ff:ff:ff
5: veth-ns2@if6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 7e:f8:18:5a:ef:1f brd ff:ff:ff:ff:ff:ff link-netnsid 0
[root@localhost ~]# ip link set veth-ns2 netns ns2
[root@localhost ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:6e:31:45 brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:bf:79:9f:de brd ff:ff:ff:ff:ff:ff
```

将两个网卡交给了network namespace后宿主机中的就没有了

再查看ns1和ns2中的link情况

```
[root@localhost ~]# ip netns exec ns1 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
6: veth-ns1@if5: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode
DEFAULT group default qlen 1000
    link/ether 7e:bb:ee:13:a2:9a brd ff:ff:ff:ff:ff:ff link-netnsid 1
[root@localhost ~]# ip netns exec ns2 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: veth-ns2@if6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode
DEFAULT group default qlen 1000
    link/ether 7e:f8:18:5a:ef:1f brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

此时veth-ns1和veth-ns2还没有ip地址，显然通信还缺少点条件

```
ip netns exec ns1 ip addr add 192.168.0.11/24 dev veth-ns1
ip netns exec ns2 ip addr add 192.168.0.12/24 dev veth-ns2
```

```
[root@localhost ~]# ip netns exec ns1 ip addr add  192.168.0.11/24 dev veth-ns1
[root@localhost ~]# ip netns exec ns1 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
5: veth-ns1@if5: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 7e:bb:ee:13:a2:9a brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet 192.168.0.11/24 scope global veth-ns1
       valid_lft forever preferred_lft forever
[root@localhost ~]# ip netns exec ns2 ip addr add  192.168.0.12/24 dev veth-ns2
[root@localhost ~]# ip netns exec ns2 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
6: veth-ns2@if6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 7e:f8:18:5a:ef:1f brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.0.12/24 scope global veth-ns2
       valid_lft forever preferred_lft forever
```

再次查看，发现state是DOWN.所以我们需要启用对应的网卡

```
[root@localhost ~]# ip netns exec ns1 ip link set veth-ns1 up
[root@localhost ~]# ip netns exec ns2 ip link set veth-ns2 up
```

然后查看状态

```
[root@localhost ~]# ip netns exec ns1 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
6: veth-ns1@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 100
    link/ether 7e:bb:ee:13:a2:9a brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet 192.168.0.11/24 scope global veth-ns1
       valid_lft forever preferred_lft forever
    inet6 fe80::7cbb:eeff:fe13:a29a/64 scope link
       valid_lft forever preferred_lft forever
[root@localhost ~]# ip netns exec ns2 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
5: veth-ns2@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 100
    link/ether 7e:f8:18:5a:ef:1f brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.0.12/24 scope global veth-ns2
       valid_lft forever preferred_lft forever
    inet6 fe80::7cf8:18ff:fe5a:ef1f/64 scope link
       valid_lft forever preferred_lft forever
```

然后就可以相互之间ping通了

```
ip netns exec ns1 ping 192.168.0.12 ip netns exec ns2 ping 192.168.0.11
```

```
[root@localhost ~]# ip netns exec ns1 ping 192.168.0.12
PING 192.168.0.12 (192.168.0.12) 56(84) bytes of data.
64 bytes from 192.168.0.12: icmp_seq=1 ttl=64 time=0.058 ms
64 bytes from 192.168.0.12: icmp_seq=2 ttl=64 time=0.040 ms
64 bytes from 192.168.0.12: icmp_seq=3 ttl=64 time=0.035 ms
^C
--- 192.168.0.12 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.035/0.044/0.058/0.011 ms
[root@localhost ~]# ip netns exec ns2 ping 192.168.0.11
PING 192.168.0.11 (192.168.0.11) 56(84) bytes of data.
64 bytes from 192.168.0.11: icmp_seq=1 ttl=64 time=0.030 ms
64 bytes from 192.168.0.11: icmp_seq=2 ttl=64 time=0.037 ms
^C
--- 192.168.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.030/0.033/0.037/0.006 ms
[root@localhost ~]#
```

1.3.2 Container的NameSpace

按照上面的描述，实际上每个container，都会有自己的network namespace，并且是独立的，我们可以进入到容器中进行验证

创建两个Tomcat容器

> docker run -d --name tomcat01 -p 8081:8080 tomcat
>
> docker run -d --name tomcat02 -p 8082:8080 tomcat

进入到两个容器中，查看ip

> docker exec -it tomcat01 ip a
>
> docker exec -it tomcat02 ip a

相互ping是可以ping通的

```
[root@localhost tomcat]# docker exec -it tomcat01 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
[root@localhost tomcat]# docker exec -it tomcat02 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
25: eth0@if26: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
[root@localhost tomcat]# docker exec -it tomcat01 ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.072 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.052 ms
^C
--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.052/0.062/0.072/0.010 ms
```

问题：此时tomcat01和tomcat02属于两个network namespace，是如何能够ping通的？ 有些小伙伴可能会想，不就跟上面的namespace实战一样吗？注意这里并没有veth-pair技术

1.4 深入分析container网络-Bridge

1.4.1 Docker默认Bridge

首先我们通过`ip a`可以查看当前宿主机的网络情况

```
[root@localhost tomcat]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
       valid_lft 66199sec preferred_lft 66199sec
    inet6 fe80::5054:ff:fe4d:77d3/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:6e:31:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.10/24 brd 192.168.56.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe6e:3145/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:52:d4:0a:9f brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:52ff:fed4:a9f/64 scope link
       valid_lft forever preferred_lft forever
24: veth78a90d0@if23: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether 7e:6b:8c:bf:7e:30 brd ff:ff:ff:ff:ff:ff link-netnsid 2
    inet6 fe80::7c6b:8cff:febf:7e30/64 scope link
       valid_lft forever preferred_lft forever
26: vetha2bfbf4@if25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether ce:2f:ed:e5:61:32 brd ff:ff:ff:ff:ff:ff link-netnsid 3
    inet6 fe80::cc2f:edff:fee5:6132/64 scope link
       valid_lft forever preferred_lft forever
```

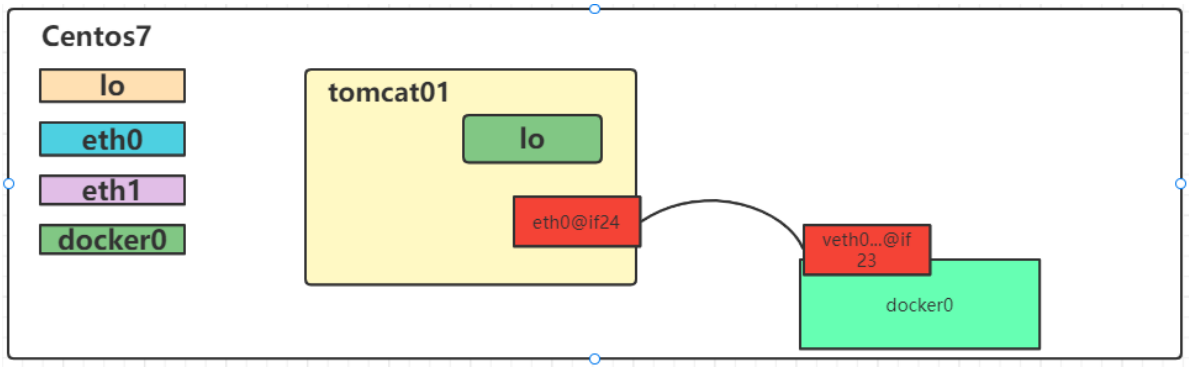
然后查看 tomcat01中的网络： docker exec -it tomcat01 ip a可以发现

```
[root@localhost tomcat]# docker exec -it tomcat01 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

我们发现在宿主机中是可以ping通Tomcat01的网络的。

```
[root@localhost tomcat]# ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.038 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.038 ms
^C
--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.038/0.038/0.038/0.000 ms
```

既然可以ping通，而且centos和tomcat01又属于两个不同的NetWork NameSpace，他们是怎么连接的？看图



其实在tomcat01中有一个eth0和centos的docker0中有一个veth是成对的，类似于之前实战中的veth-ns1和veth-ns2,要确认也很简单

```
yum install bridge-utils
brctl show
```

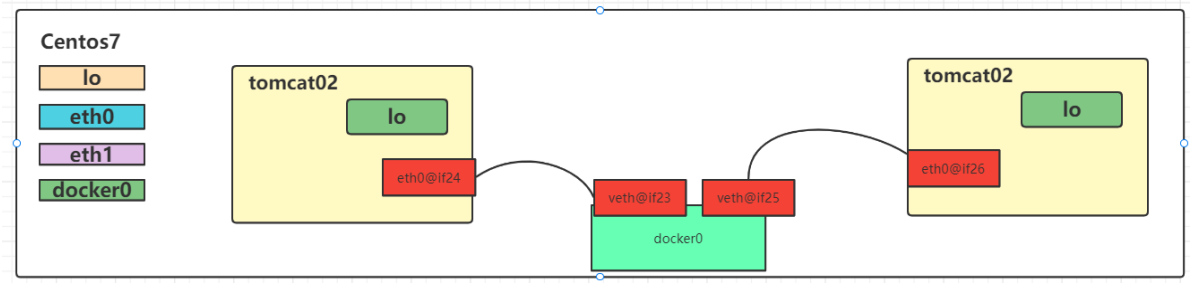
执行

```
[root@localhost tomcat]# brctl show
bridge name bridge id       STP enabled interfaces
docker0     8000.024252d40a9f   no      veth78a90d0
                                        vetha2bfbf4
```

对比 ip a 情况

```
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 52:54:00:4d:77:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
       valid_lft 65692sec preferred_lft 65692sec
    inet6 fe80::5054:ff:fe4d:77d3/64 scope link
       valid_lft forever preferred_lft forever
3:                                                                  roup default qlen 1000
    [root@localhost tomcat]# brctl show
    bridge name     bridge id               STP enabled     interfaces
    docker0         8000.024252d40a9f       no              veth78a90d0
                                                            vetha2bfbf4
    [root@localhost tomcat]# ip a
       valid_lft forever preferred_lft forever
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:52:d4:0a:9f brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:52ff:fed4:a9f/64 scope link
       valid_lft forever preferred_lft forever
24: veth78a90d0@if23: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 7e:6b:8c:bf:7e:30 brd ff:ff:ff:ff:ff:ff link-netnsid 2
    inet6 fe80::7c6b:8cff:febf:7e30/64 scope link
       valid_lft forever preferred_lft forever
26: vetha2bfbf4@if25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether ce:2f:ed:e5:61:32 brd ff:ff:ff:ff:ff:ff link-netnsid 3
    inet6 fe80::cc2f:edff:fee5:6132/64 scope link
       valid_lft forever preferred_lft forever
[root@localhost tomcat]#
```

那么画图说明：



这种网络连接方法我们称之为Bridge，其实也可以通过命令查看docker中的网络模式：docker network ls , bridge也是docker中默认的网络模式

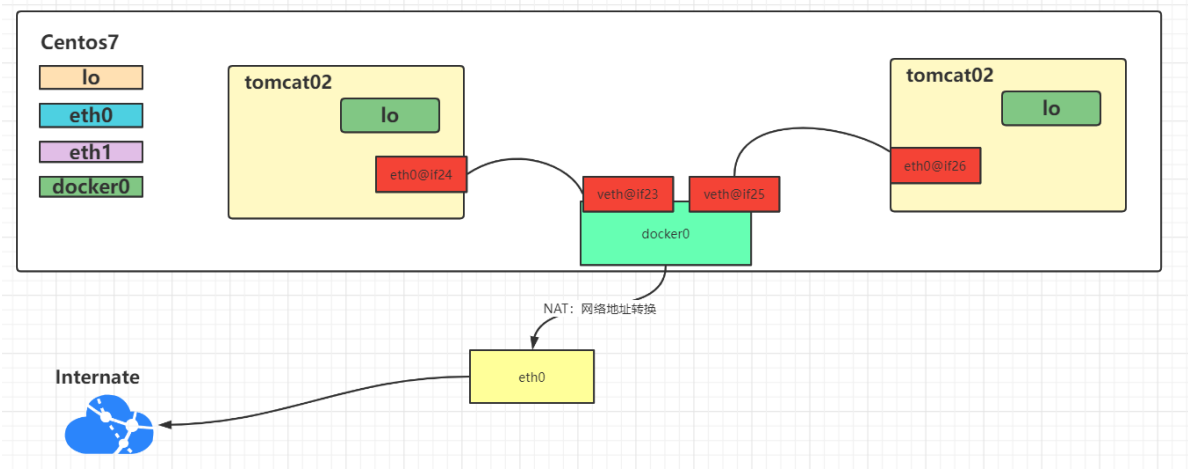
```
[root@localhost tomcat]# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
92242fc0f805   bridge    bridge    local
96b999d7fcc2   host      host      local
17b86f9caa33   none      null      local
```

不妨检查一下bridge：docker network inspect bridge

```
"Containers": {
            "4b3500fed6b99c00b3ed1ae46bd6bc33040c77efdab343175363f32fbcf42e63":
{
                "Name": "tomcat01",
                "EndpointID":
"40fc0925fcb59c9bb002779580107ab9601640188bf157fa57b1c2de9478053a",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "92d2ff3e9be523099ac4b45058c5bf4652a77a27b7053a9115ea565ab43f9ab0":
{
                "Name": "tomcat02",
                "EndpointID":
"1d6c3bd73e3727dd368edf3cc74d2f01b5c458223f844d6188486cb26ea255bc",
```

```
            "MacAddress": "02:42:ac:11:00:03",
            "IPv4Address": "172.17.0.3/16",
            "IPv6Address": ""
        }
    }
```

在tomcat01容器中是可以访问互联网的，顺便把这张图画一下咯，NAT是通过iptables实现的



1.4.2 自定义NetWork

创建一个network，类型为 Bridge

```
docker network create tomcat-net
或者
docker network create tomcat-net --subnet=172.18.0.0/24 tomcat-net
```

查看已有的NetWork: docker network ls

```
[root@localhost ~]# docker network create tomcat-net
43915cba1f9204751b48896d7d28b83b4b6cf35f06fac6ff158ced5fb9ddb5b3
[root@localhost ~]# docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
b5c9cfbc0410   bridge        bridge    local
96b999d7fcc2   host          host      local
17b86f9caa33   none          null      local
43915cba1f92   tomcat-net    bridge    local
```

查看tomcat-net详情信息：docker network inspect tomcat-net

```
[root@localhost ~]# docker network inspect tomcat-net
[
    {
        "Name": "tomcat-net",
        "Id":
"43915cba1f9204751b48896d7d28b83b4b6cf35f06fac6ff158ced5fb9ddb5b3",
        "Created": "2021-10-11T12:10:19.543766962Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
```

```
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

删除network：docker network rm tomcat-net

创建tomcat容器，并指定使用tomcat-net

```
[root@localhost ~]# docker run -d --name custom-net-tomcat --network tomcat-net
tomcat-ip:1.0
264b3901f8f12fd7f4cc69810be6a24de48f82402b1e5b0df364bd1ee72d8f0e
```

查看custom-net-tomcat的网络信息:截取了关键信息

```
12: br-43915cba1f92: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether 02:42:71:a6:67:c7 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-43915cba1f92
       valid_lft forever preferred_lft forever
    inet6 fe80::42:71ff:fea6:67c7/64 scope link
       valid_lft forever preferred_lft forever
14: veth282a555@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-43915cba1f92 state UP group default
    link/ether 3a:3d:83:15:3f:ed brd ff:ff:ff:ff:ff:ff link-netnsid 3
    inet6 fe80::383d:83ff:fe15:3fed/64 scope link
       valid_lft forever preferred_lft forever
```

查看网卡接口信息

```
[root@localhost ~]# brctl show
bridge name               bridge id     STP   enabled     interfaces
br-43915cba1f92     8000.024271a667c7   no                veth282a555
docker0             8000.02423964f095   no                veth4526c0c
                                                          vethaa2f6f4
                                                          vethc6ad4c2
```

此时在custom-net-tomcat容器中ping一些tomcat01发现是ping不通的

```
[root@localhost ~]# docker exec -it custom-net-tomcat ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
^C
--- 172.17.0.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2000ms
```

此时如果tomcat01容器能够连接上tomcat-net上应该就可以了

```
docker network connect tomcat-net tomcat01
```

```
[root@localhost ~]# docker exec -it tomcat01 ping custom-net-tomcat
PING custom-net-tomcat (172.18.0.2) 56(84) bytes of data.
64 bytes from custom-net-tomcat.tomcat-net (172.18.0.2): icmp_seq=1 ttl=64
time=0.138 ms
^C
--- custom-net-tomcat ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.138/0.138/0.138/0.000 ms
[root@localhost ~]# docker exec -it custom-net-tomcat ping tomcat01
PING tomcat01 (172.18.0.3) 56(84) bytes of data.
64 bytes from tomcat01.tomcat-net (172.18.0.3): icmp_seq=1 ttl=64 time=0.031 ms
```

1.5 深入分析 Container网络-Host&None

1.5.1 Host

Host模式下,容器将共享主机的网络堆栈,并且主机的所有接口都可供容器使用.容器的主机名将与主机系统上的主机名匹配

创建一个容器，并指定网络为host

```
docker run -d --name my-tomcat-host --network host tomcat-ip:1.0
```

查看ip地址

```
docker exec -it my-tomcat-host ip a
```

检查host网络

```
docker network inspect host
```

```
"Containers": {
            "f495a6892d422e61daab01e3fcfa4abb515753e5f9390af44c93cae376ca7464":
{
                "Name": "my-tomcat-host",
                "EndpointID":
"77012b1ac5d15bde3105d2eb2fe0e58a5ef78fb44a88dc8b655d373d36cde5da",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            }
        }
```

1.5.2 None

None模式不会为容器配置任何IP,也不能访问外部网络以及其他容器.它具有环回地址,可用于运行批处理作业.

创建一个tomcat容器，并指定网络为none

```
docker run -d --name my-tomcat-none --network none tomcat-ip:1.0
```

查看ip地址

```
docker exec -it my-tomcat-none
```

检查none网络

```
docker network inspect none
```

```
        "Containers": {
            "c957b61dae93fbb9275acf73c370e5df1aaf44a986579ee43ab751f790220807":
{
                "Name": "my-tomcat-none",
                "EndpointID":
"16bf30fb7328ceb433b55574dc071bf346efa58e2eb92b6f40d7a902ddc94293",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            }
        }
```

1.6 端口映射

创建一个tomcat容器，名称为port-tomcat

```
docker run -d --name port-tomcat tomcat-ip:1.0
```

思考如何访问tomcat的服务

```
docker exec -it port-tomcat bash
curl localhost:8080
```

如果要载centos7上访问呢

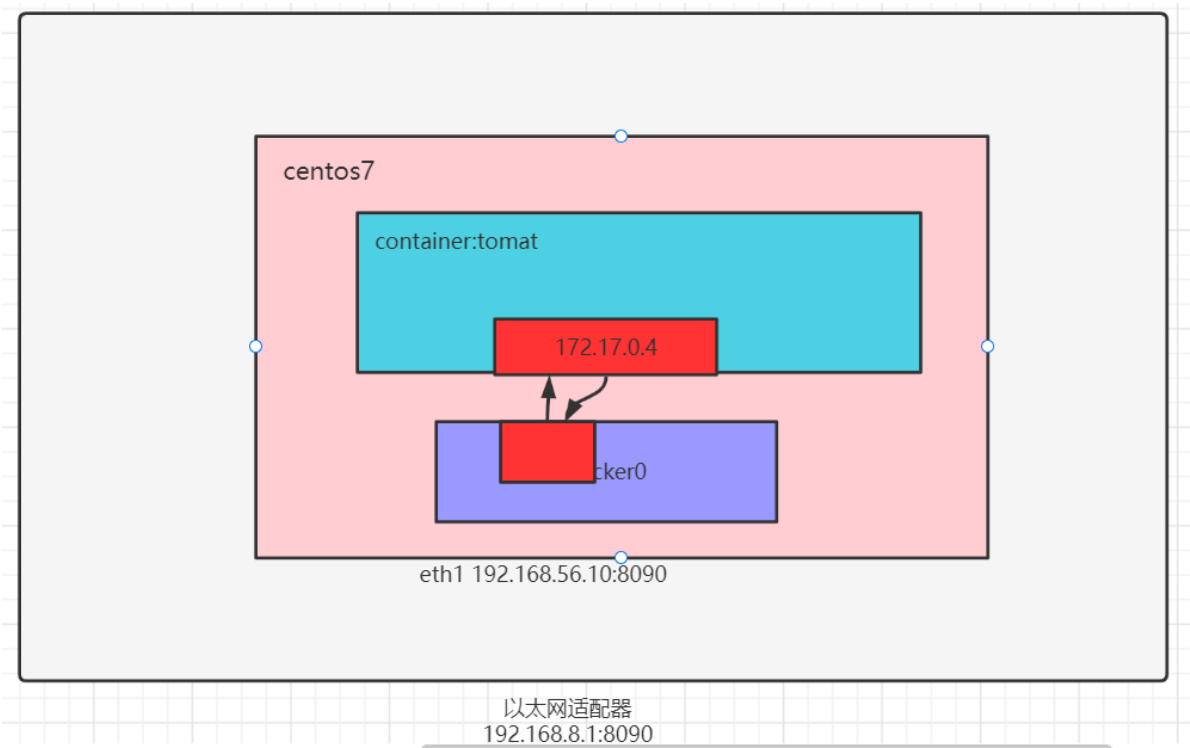
```
docker exec -it port-tomcat ip a
curl 172.17.0.4:8080
```

如果我们需要在centos中通过localhost来访问呢?这时我们就需要将port-tomcat中的8080端口映射到centos上了

```
docker rm -f port-tomcat
docker run -d --name port-tomcat -p 8090:8080 tomcat-ip:1.0
curl localhost:8090
```

centos7是运行在win10上的虚拟机，如果想要在win10上通过ip:port方式访问呢?

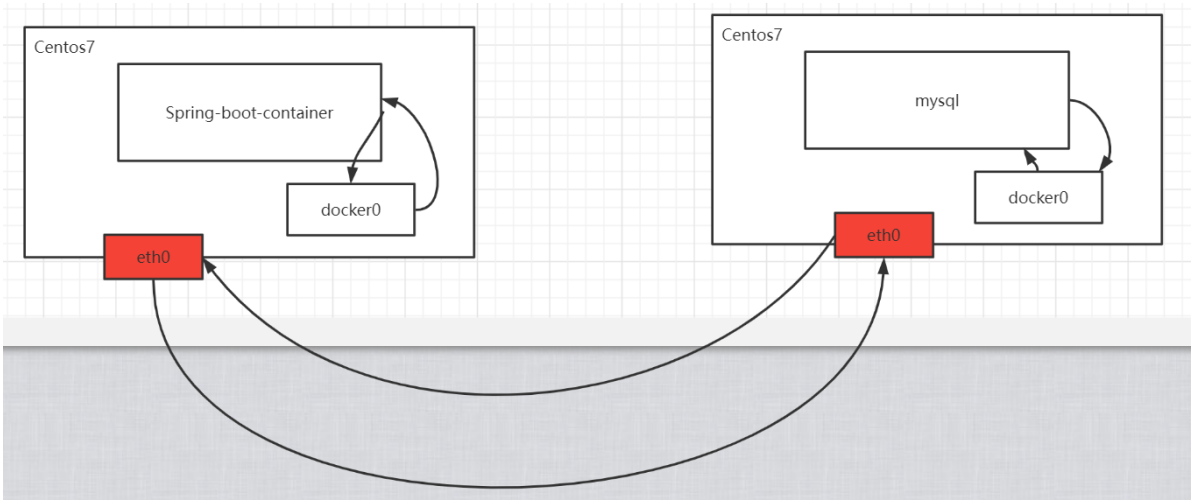
```
#此时需要centos和win网络在同一个网段，所以在Vagrantfile文件中

#这种方式等同于桥接网络。也可以给该网络指定使用物理机哪一块网卡，比如
#config.vm.network"public_network",:bridge=>'en1: Wi-Fi (AirPort)'
config.vm.network"public_network"

centos7: ip a   --->192.168.8.118
win10:浏览器访问     192.168.8.118:9080
```



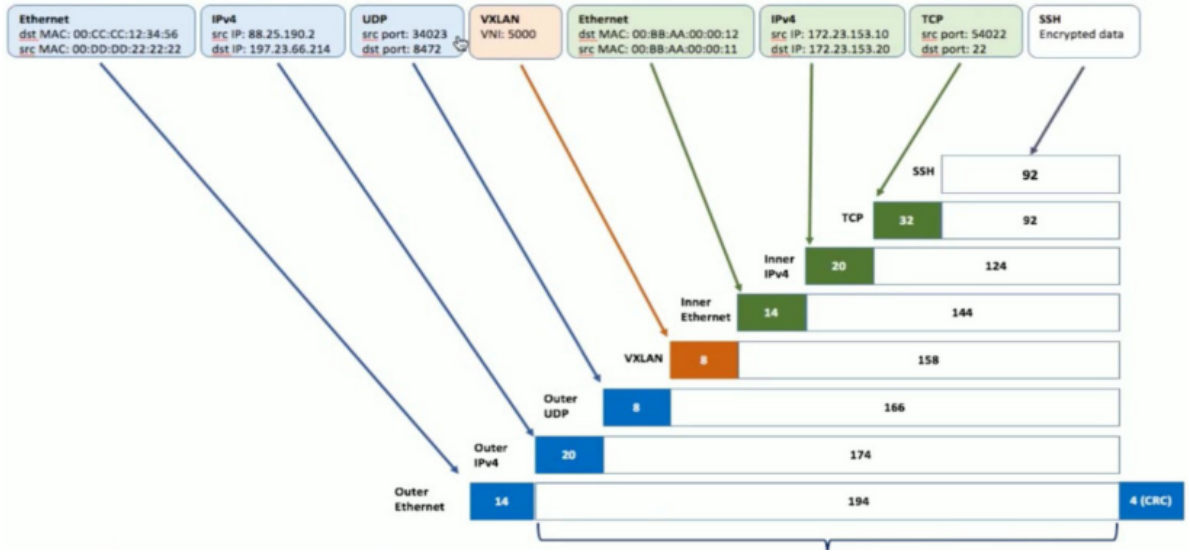
1.7 多机之间通信

具体深入介绍会在 Docker Swarm 中详聊，本节简单介绍。

在同一台centos7机器上，发现无论怎么折腾，我们一定有办法让两个container通信。 那如果是在两台centos7机器上呢？画个图



VXLAN技术实现：Virtual Extensible LAN(虚拟可扩展局域网)。



2.Docker实战

2.1 MySQL高可用集群搭建

MySQL集群搭建在实际项目中还是非常必须的，我们通过PXC【Percona XtraDB Cluster】来实现强一致性数据库集群搭建。

2.1.1 MySQL集群搭建

1> 拉去镜像

```
docker pull percona/percona-xtradb-cluster:5.7.21
```

2> 复制pxc镜像【重命名】

```
docker tag percona/percona-xtradb-cluster:5.7.21 pxc
```

3>删除原来的镜像

```
docker rmi percona/percona-xtradb-cluster:5.7.21
```

4>创建单独的网段，给MySQL数据库集群使用

```
docker network create --subnet=172.20.0.0/24 pxc-net
docker network inpsect pxc-net # 查看详情
docker network rm pxc-net # 删除网段
```

5> 创建和删除volume

```
docker volume create --name v1 # 创建 volume
docker volume rm v1 # 删除volume
docker volume inspect v1 # 查看详情
```

6> 搭建pxc集群

准备三个数据卷

```
docker volume create --name v1
docker volume create --name v2
docker volume create --name v3
```

运行3个PXC容器

[CLUSTER_NAME PXC集群名字]

[XTRABACKUP_PASSWORD数据库同步需要用到的密码]

创建第一个节点

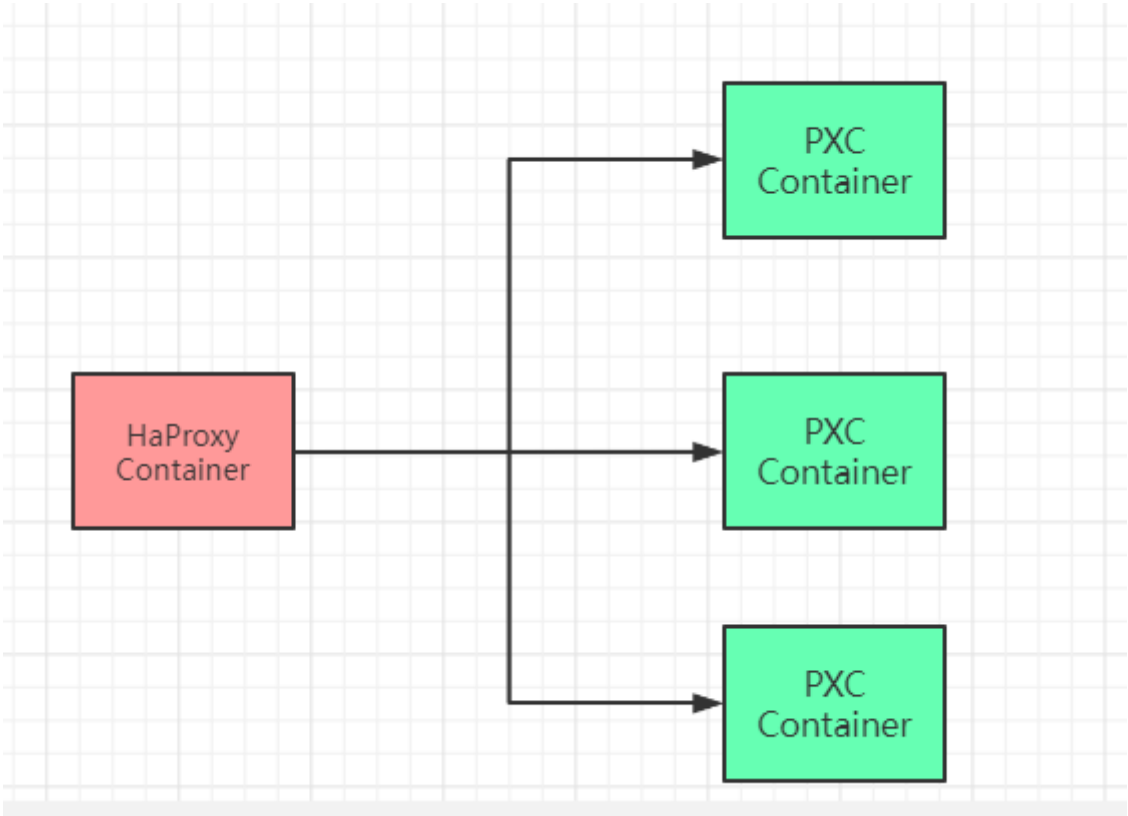
```
docker run -d -p 3301:3306 -v v1:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 -e
CLUSTER_NAME=PXC -e EXTRABACKUP_PASSWROD=123456 --privileged --name=node1 --
net=pxc-net --ip 172.22.0.2 pxc
```

创建第二个和第三个节点: 注意 `-e CLUSTER_JOIN=node1`

```
docker run -d -p 3302:3306 -v v2:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 -e
CLUSTER_NAME=PXC -e EXTRABACKUP_PASSWROD=123456 -e CLUSTER_JOIN=node1 --
privileged --name=node2 --net=pxc-net --ip 172.22.0.3 pxc
```

```
docker run -d -p 3303:3306 -v v3:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 -e
CLUSTER_NAME=PXC -e EXTRABACKUP_PASSWROD=123456 -e CLUSTER_JOIN=node1 --
privileged --name=node3 --net=pxc-net --ip 172.22.0.4 pxc
```

2.1.2 负载均衡服务搭建



1>拉去镜像

```
docker pull haproxy
```

2>创建haproxy的配置文件。

```
touch /tmp/haproxy/haproxy.cfg
```

配置文件中的内容

```
global
    #工作目录，这边要和创建容器指定的目录对应
    # chroot /usr/local/etc/haproxy
    #日志文件
    log 127.0.0.1 local5 info
    #守护进程运行
    daemon

defaults
    log global
    mode    http
    #日志格式
    option  httplog
    #日志中不记录负载均衡的心跳检测记录
    option  dontlognull
    #连接超时（毫秒）
    timeout connect 5000
    #客户端超时（毫秒）
    timeout client  50000
    #服务器超时（毫秒）
    timeout server  50000

    #监控界面
    listen  admin_stats
    #监控界面的访问的IP和端口
    bind  0.0.0.0:8888
    #访问协议
    mode        http
    #URI相对地址
    stats uri   /dbs_monitor
    #统计报告格式
    stats realm     Global\ statistics
    #登陆帐户信息
    stats auth  admin:admin
    #数据库负载均衡
    listen  proxy-mysql
    #访问的IP和端口，haproxy开发的端口为3306
    #假如有人访问haproxy的3306端口，则将请求转发给下面的数据库实例
    bind  0.0.0.0:3306
    #网络协议
    mode  tcp
    #负载均衡算法（轮询算法）
    #轮询算法：roundrobin
    #权重算法：static-rr
    #最少连接算法：leastconn
    #请求源IP算法：source
```

```
    balance  roundrobin
    #日志格式
    option  tcplog
    #在MySQL中创建一个没有权限的haproxy用户，密码为空。
    #Haproxy使用这个账户对MySQL数据库心跳检测
    option  mysql-check user haproxy
    server  MySQL_1 172.22.0.2:3306 check weight 1 maxconn 2000
    server  MySQL_2 172.22.0.3:3306 check weight 1 maxconn 2000
    server  MySQL_3 172.22.0.4:3306 check weight 1 maxconn 2000
    #使用keepalive检测死链
    option  tcpka
```

3>创建haproxy容器

```
docker run  -d -p 8888:8888 -p 3306:3306 -v /tmp/haproxy:/usr/local/etc/haproxy
--name haproxy01 --privileged --net=pxc-net haproxy
```

4>在MySQL数据库上创建用户，用于心跳检测

```
CREATE USER 'haproxy'@'%' IDENTIFIED BY '';
```

5>win浏览器访问

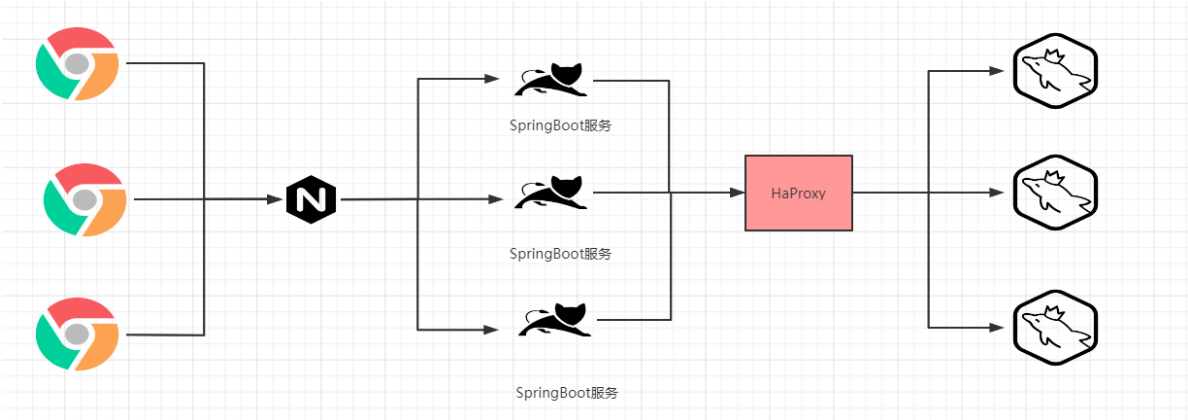
```
http://centos_ip:8888/dbs_monitor
用户名密码都是:admin
```

6>客户端连接工具连接

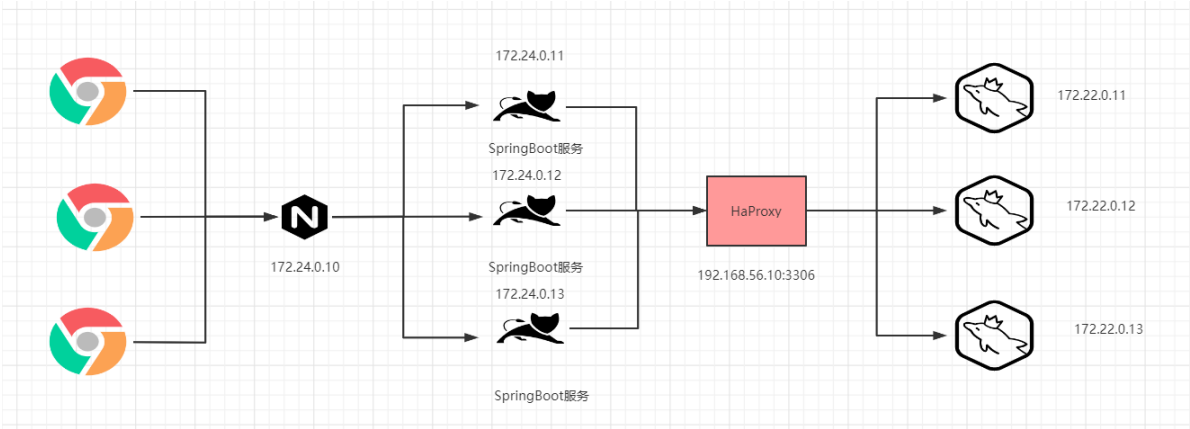
```
ip:centos_ip
port:3306
user:root
password:123456
```

2.2 SpringBoot项目部署

咱们一个实际的项目部署情况应该是这样的。



接下来我们就在MySQL集群环境的基础上来完成一个SpringBoot项目的集群部署操作。网络地址分配为：



1> 创建对应的网络

```
docker network create --subnet=172.24.0.0/24 sbm-net
```

2>创建SpringBoot项目

通过SpringBoot项目整合MyBatis实现CRUD操作，

属性文件中配置的jdbc信息为

```
# jdbc的相关配置信息
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://192.168.56.10:3306/haproxy-test?
serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8&useSSL=true
spring.datasource.username=root
spring.datasource.password=123456
# 连接池
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
# mybatis给package设置别名
mybatis.type-aliases-package=com.bobo.pojo
#指定映射文件的位置
mybatis.mapper-locations=classpath:mapper/*.xml
```

3>对应的项目打成jar包，并上传到centos7中目录放在 /tmp/springboot/ 下 然后创建Dockerfile文件

```
yum install -y lrzsz
```

```
FROM openjdk:8
MAINTAINER bobo
LABEL name="springboot-mybatis" version="1.0" author="bobo"
COPY springboot-mybatis-demo-0.0.1-SNAPSHOT.jar springboot-mybatis.jar
CMD ["java","-jar","springboot-mybatis.jar"]
```

4>基于Dockerfile构建镜像

```
    docker build -t sbm-image .
```

5>基于image创建container

```
docker run -d --name sb01 -p 8081:8080 --net=sbm-net --ip 172.24.0.11 sbm-image
```

6>查看启动日志docker logs sb01

7>浏览器访问测试

http://192.168.56.10:8081/user/query

8>创建多个容器

```
docker run -d --name sb01 -p 8081:8080 --net=pro-net --ip 172.24.0.11 sbm-image
docker run -d --name sb02 -p 8082:8080 --net=pro-net --ip 172.24.0.12 sbm-image
docker run -d --name sb03 -p 8083:8080 --net=pro-net --ip 172.24.0.13 sbm-image
```

9>Nginx安装

我们通过Nginx来实现负载均衡服务

在centos的/tmp/nginx下新建nginx.conf文件，并进行相应的配置

```
user nginx;
worker_processes  1;
events {
    worker_connections  1024;
}
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;


    server {
        listen 80;
        location / {
         proxy_pass http://balance;
        }
    }

    upstream balance{
        server 172.24.0.11:8080;
        server 172.24.0.12:8080;
        server 172.24.0.13:8080;
    }
    include /etc/nginx/conf.d/*.conf;
}
```

创建容器

```
docker run -d --name my-nginx -p 80:80 -v
/tmp/nginx/nginx.conf:/etc/nginx/nginx.conf --network=pxc-net --ip 172.24.0.10
nginx
```

搞定~

3.DockerCompose

3.1 DockerCompose介绍

Compose 是用于定义和运行多容器 Docker 应用程序的工具。通过 Compose，您可以使用 YML 文件来配置应用程序需要的所有服务。然后，使用一个命令，就可以从 YML 文件配置中创建并启动所有服务。

一键启动所有的服务

DockerCompose的使用步骤

- 创建对应的DockerFile文件
- 创建yml文件，在yml文件中编排我们的服务
- 通过`docker-compose up`命令 一键运行我们的容器

3.2 Compose安装

官网地址：https://docs.docker.com/compose

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

速度比较慢的话使用下面的地址：

```
curl -L https://get.daocloud.io/docker/compose/releases/download/1.25.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

修改文件夹权限

```
chmod +x /usr/local/bin/docker-compose
```

建立软连接

```
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

校验是否安装成功

```
docker-compose --version
```

3.3 Compose初体验

通过官方案例来演示：https://docs.docker.com/compose/gettingstarted/

创建对应的目录

```
mkdir composetest
cd composetest
```

创建Python文件 app.py

```
import time

import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello World! I have been seen {} times.\n'.format(count)
```

在同级目录下创建`requirements.txt`文件

```
flask
redis
```

然后创建对应的Dockerfile文件

```
# syntax=docker/dockerfile:1
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

然后创建核心的 yml文件`docker-compose.yml`

```yaml
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

最终通过`docker-compose up`命令来启动容器

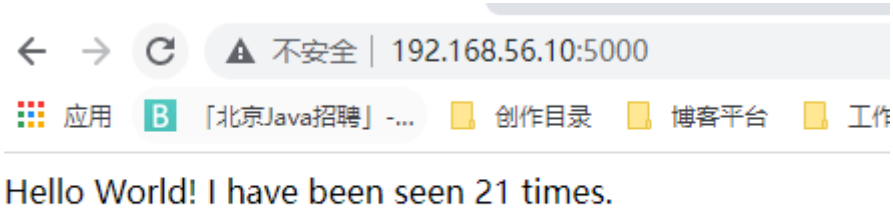
```
docker-compose up
```

```
Step 1/10 : FROM python:3.7-alpine
 ---> 206aee52d71d
Step 2/10 : WORKDIR /code
 ---> Running in 72980bfcd88d
Removing intermediate container 72980bfcd88d
 ---> 454d04b60253
Step 3/10 : ENV FLASK_APP=app.py
 ---> Running in c4e953e7af35
Removing intermediate container c4e953e7af35
 ---> 3d34c8852c51
Step 4/10 : ENV FLASK_RUN_HOST=0.0.0.0
 ---> Running in 6c1bec6a2038
Removing intermediate container 6c1bec6a2038
 ---> 26aacba3b2dc
Step 5/10 : RUN apk add --no-cache gcc musl-dev linux-headers
 ---> Running in c7fa80e249cc
fetch https://dl-cdn.alpinelinux.org/alpine/v3.14/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.14/community/x86_64/APKINDEX.tar.gz
(1/13) Installing libgcc (10.3.1_git20210424-r2)
(2/13) Installing libstdc++ (10.3.1_git20210424-r2)
(3/13) Installing binutils (2.35.2-r2)
(4/13) Installing libgomp (10.3.1_git20210424-r2)
(5/13) Installing libatomic (10.3.1_git20210424-r2)
(6/13) Installing libgphobos (10.3.1_git20210424-r2)
(7/13) Installing gmp (6.2.1-r0)
(8/13) Installing isl22 (0.22-r0)
(9/13) Installing mpfr4 (4.1.0-r0)
(10/13) Installing mpc1 (1.2.1-r0)
(11/13) Installing gcc (10.3.1_git20210424-r2)
(12/13) Installing linux-headers (5.10.41-r0)
(13/13) Installing musl-dev (1.2.2-r3)
```

```
Removing intermediate container c7fa80e249cc
 ---> 6c796002a5e0
Step 6/10 : COPY requirements.txt requirements.txt
 ---> 07dc2f72a334
Step 7/10 : RUN pip install -r requirements.txt
 ---> Running in f87be966b9b6
Collecting flask
  Downloading Flask-2.0.2-py3-none-any.whl (95 kB)
Collecting redis
  Downloading redis-3.5.3-py2.py3-none-any.whl (72 kB)
Collecting click>=7.1.2
  Downloading click-8.0.3-py3-none-any.whl (97 kB)
Collecting Jinja2>=3.0
  Downloading Jinja2-3.0.2-py3-none-any.whl (133 kB)
Collecting Werkzeug>=2.0
  Downloading Werkzeug-2.0.2-py3-none-any.whl (288 kB)
Collecting itsdangerous>=2.0
  Downloading itsdangerous-2.0.1-py3-none-any.whl (18 kB)
Collecting importlib-metadata
  Downloading importlib_metadata-4.8.1-py3-none-any.whl (17 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.0.1.tar.gz (18 kB)
Collecting typing-extensions>=3.6.4
  Downloading typing_extensions-3.10.0.2-py3-none-any.whl (26 kB)
Collecting zipp>=0.5
  Downloading zipp-3.6.0-py3-none-any.whl (5.3 kB)
Building wheels for collected packages: MarkupSafe
  Building wheel for MarkupSafe (setup.py): started
  Building wheel for MarkupSafe (setup.py): finished with status 'done'
  Created wheel for MarkupSafe: filename=MarkupSafe-2.0.1-cp37-cp37m-linux_x86_64.whl size=14616 sha256=54c1cf7688a924094838a78d4
c4422dd2497fa440712449c4aba282b0e5cf58e
```

启动时间比较久，耐心等待即可

测试访问：



如果要退出服务 `Ctrl+c` 或者 `docker-compose down`

3.4 Compose配置规则

docker-compse.yml核心

官网地址：https://docs.docker.com/compose/compose-file/compose-file-v3/

```
version: '' # 版本
servers:  # 服务
  服务1: web
     # 服务的配置
     build
     network
     images
  服务2: redis
  服务3:
  服务4:
  ...
# 其他配置 网络，全局的规则 数据卷
volumes:
configs:
networks:
```

3.5 Compose一键部署实战

3.5.1 一键部署WP博客

1> 创建my_wordpress目录

```
mkdir my_wordpress
```

2>创建yml文件docker-compose.yml

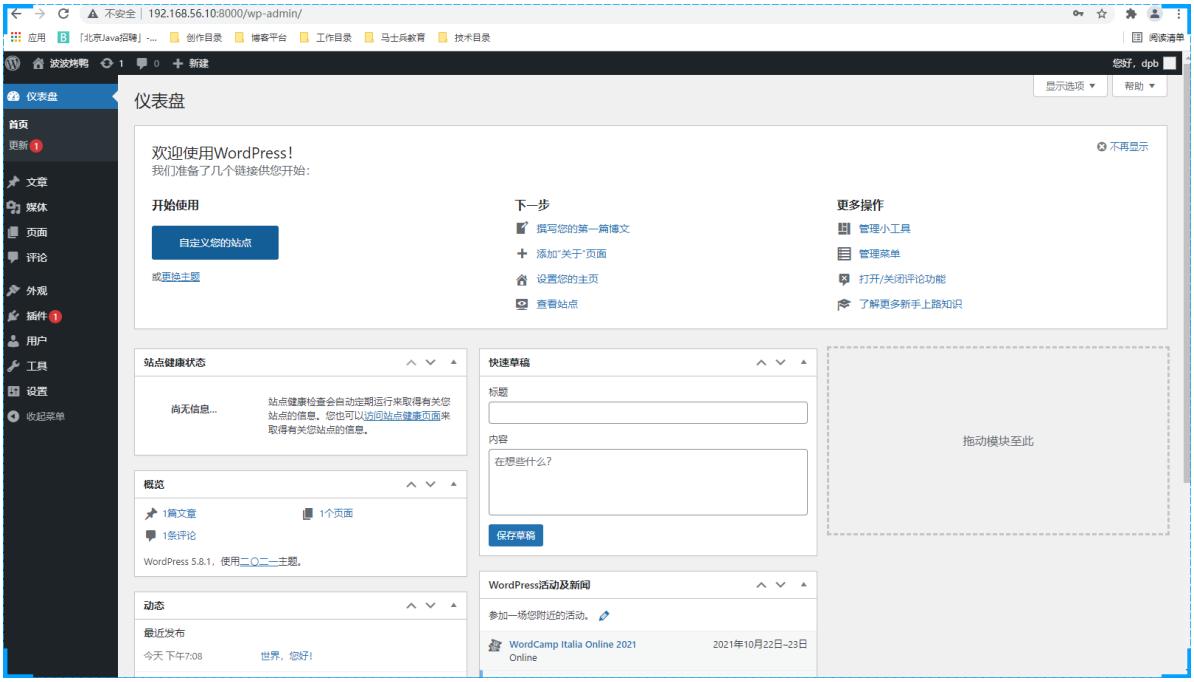
```
version: "3.9"

services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
```

```
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    volumes:
      - wordpress_data:/var/www/html
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
  wordpress_data: {}
```

3>通过up命令启动

```
docker-compose up -d
```

```
[root@localhost my-wordpress]# docker-compose up -d
Creating network "my-wordpress_default" with the default driver
Creating volume "my-wordpress_db_data" with default driver
Creating volume "my-wordpress_wordpress_data" with default driver
Creating my-wordpress_db_1 ... done
Creating my-wordpress_wordpress_1 ... done
```



3.5.2 部署一个SpringBoot项目

我们自己通过Java项目实现访问计数的功能

```
FROM java:8
COPY my-counter-views-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8080
CMD ["java","-jar","app.jar"]
```

yml

```
version: '3.9'
services:
  myapp:
    build: .
    image: myapp
    depends_on:
      - redis
    ports:
      - "8080:8080"
  redis:
    image: "library/redis:alpine"
```

3.6 Compose常见操作

(1)查看版本

docker-compose version

(2)根据yml创建service

docker-compose up

指定yaml：docker-compose up -f xxx.yaml

后台运行：docker-compose up -d

(3)查看启动成功的service

docker-compose ps

也可以使用docker ps

(4)查看images

docker-compose images

(5)停止/启动service

docker-compose stop/start

(6)删除service[同时会删除掉network和volume]

docker-compose down

(7)进入到某个service

docker-compose exec redis sh

3.7 scale扩缩容

```
docker-compose up --scale web=5 -d
```

```
    --no-log-prefix           Don't print prefix in logs.
[root@localhost myapp]# docker-compose up --scale redis=5 -d
Creating network "myapp_default" with the default driver
Creating myapp_redis_1 ... done
Creating myapp_redis_2 ... done
Creating myapp_redis_3 ... done
Creating myapp_redis_4 ... done
Creating myapp_redis_5 ... done
Creating myapp_myapp_1 ... done
[root@localhost myapp]# docker-compose ps
    Name                  Command              State                    Ports
-------------------------------------------------------------------------------------------
myapp_myapp_1   java -jar app.jar               Up      0.0.0.0:8080->8080/tcp,:::8080->8080/tcp
myapp_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
myapp_redis_2   docker-entrypoint.sh redis ...   Up      6379/tcp
myapp_redis_3   docker-entrypoint.sh redis ...   Up      6379/tcp
myapp_redis_4   docker-entrypoint.sh redis ...   Up      6379/tcp
myapp_redis_5   docker-entrypoint.sh redis ...   Up      6379/tcp
[root@localhost myapp]#
```

小结：docker-compose

工程-->服务-->容器

```
version:
services:
  服务1:
  服务2:
  服务3:
```

4.Harbor

镜像私服仓库

4.1 Docker hub

官网地址：hub.docker.com

(1)在docker机器上登录

docker login

(2)输入用户名和密码

```
[root@localhost ~]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
.docker.com to create one.
Username: q279583842q
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

(3)docker push q279583842q/tomcat-ip

[注意镜像名称要和docker id一致，不然push不成功]

(4)给image重命名，并删除掉原来的

docker tag tomcat-ip q279583842q/tomcat-ip

docker rmi -f tomcat-ip

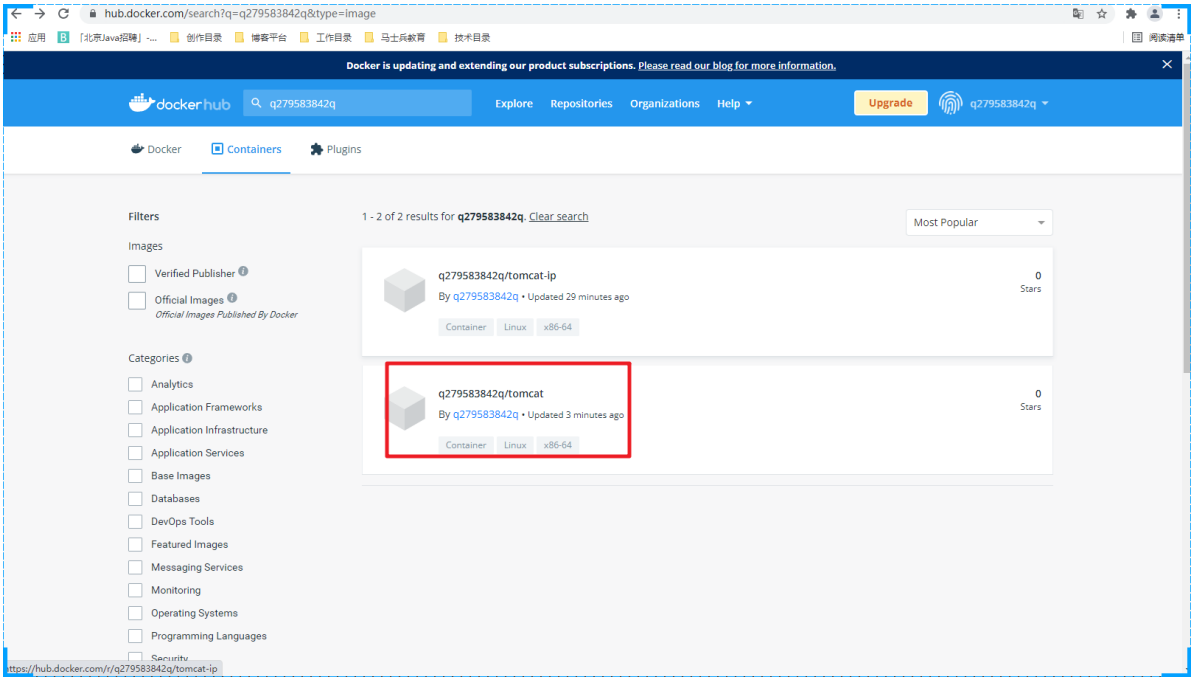
(5)再次推送，刷新hub.docker.com后台，发现成功

```
[root@localhost ~]# docker push q279583842q/tomcat-ip
Using default tag: latest
The push refers to repository [docker.io/q279583842q/tomcat-ip]
58dbe55cc107: Pushed
c1f1c0e7b50d: Pushed
39bfade81d13: Pushed
584e5d5c5e44: Pushed
09ec2ec0b67f: Pushed
74ddd0ec08fa: Pushed
latest: digest: sha256:e6f3b11959dbf151f5525dff3522fe7b394480fb0eb290c6f01fb9dd78396f1f size: 15
[root@localhost ~]#
```

(6)别人下载，并且运行

docker pull q279583842q/tomcat-ip

docker run -d --name user01 -p 6661:8080 q279583842q/tomcat-ip

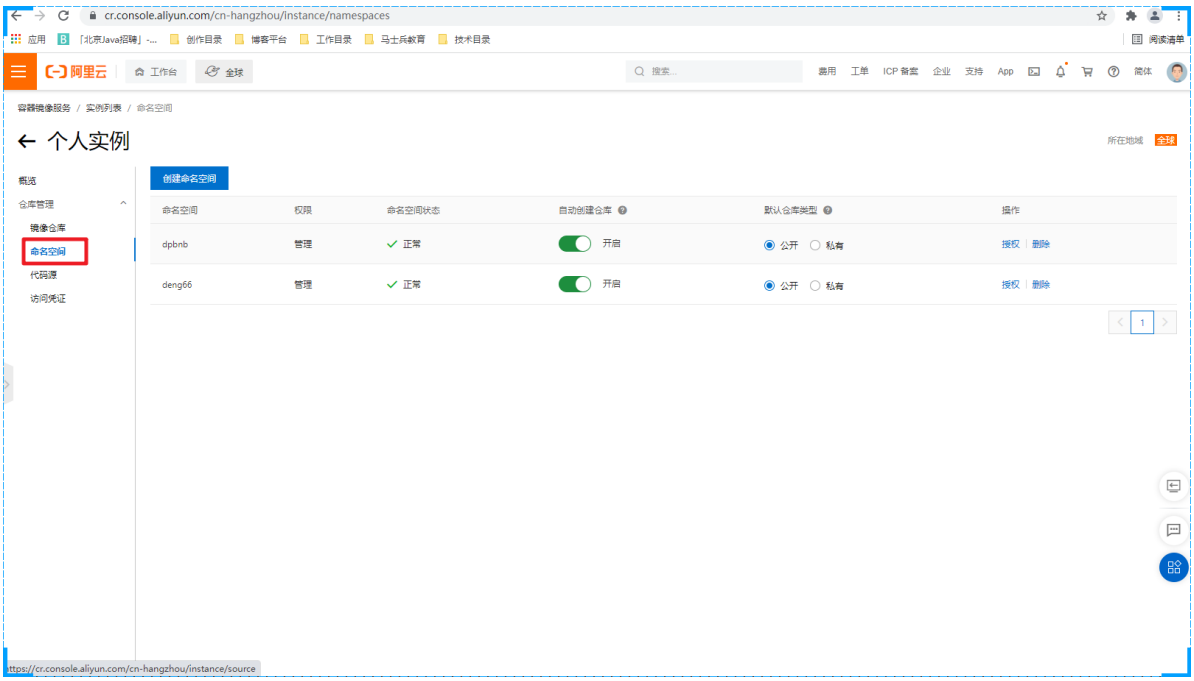


服务上传和下载比较耗时

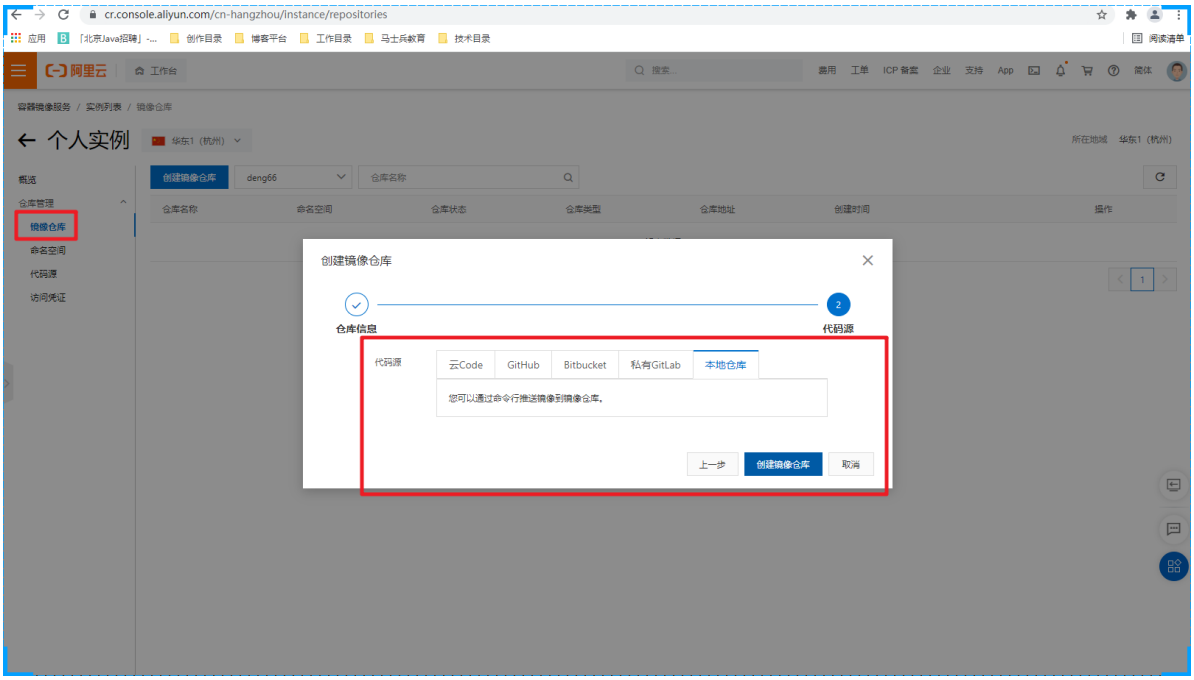
4.2 阿里云Docker Registry

仓库地址：https://cr.console.aliyun.com/cn-hangzhou/instances/repositories

登录阿里云，进入到镜像服务，创建命名空间



创建镜像仓库



将镜像推送到阿里云镜像仓库

```
$ docker login --username=dpb2****83842 registry.cn-hangzhou.aliyuncs.com
$ docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/deng66/dpb-tomcat:[镜像版本号]
$ docker push registry.cn-hangzhou.aliyuncs.com/deng66/dpb-tomcat:[镜像版本号]
```

```
[root@localhost ~]# docker push registry.cn-hangzhou.aliyuncs.com/deng66/dpb-tomcat:1.0
The push refers to repository [registry.cn-hangzhou.aliyuncs.com/deng66/dpb-tomcat]
58dbe55cc107: Pushed
c1f1c0e7b50d: Pushed
39bfade81d13: Pushed
584e5d5c5e44: Pushed
09ec2ec0b67f: Pushed
74ddd0ec08fa: Pushed
1.0: digest: sha256:e6f3b11959dbf151f5525dff3522fe7b394480fb0eb290c6f01fb9dd78396f1f size: 1585
```

我们pull镜像的步骤

```
$ docker pull registry.cn-hangzhou.aliyuncs.com/deng66/dpb-tomcat:[镜像版本号]
```

4.3 私服Harbor

4.3.1 Harbor简介

Docker容器应用的开发和运行离不开可靠的镜像管理，虽然Docker官方也提供了公共的镜像仓库，但是从安全和效率等方面考虑，部署我们私有环境内的Registry也是非常必要的。Harbor是由VMware公司开源的企业级的Docker Registry管理项目，它包括权限管理(RBAC)、LDAP、日志审核、管理界面、自我注册、镜像复制和中文支持等功能。

4.3.2 功能介绍

组件	功能
harbor-adminserver	配置管理中心
harbor-db	Mysql数据库
harbor-jobservice	负责镜像复制
harbor-log	记录操作日志
harbor-ui	Web管理页面和API
nginx	前端代理，负责前端页面和镜像上传/下载转发
redis	会话
registry	镜像存储

4.3.3 Harbor安装

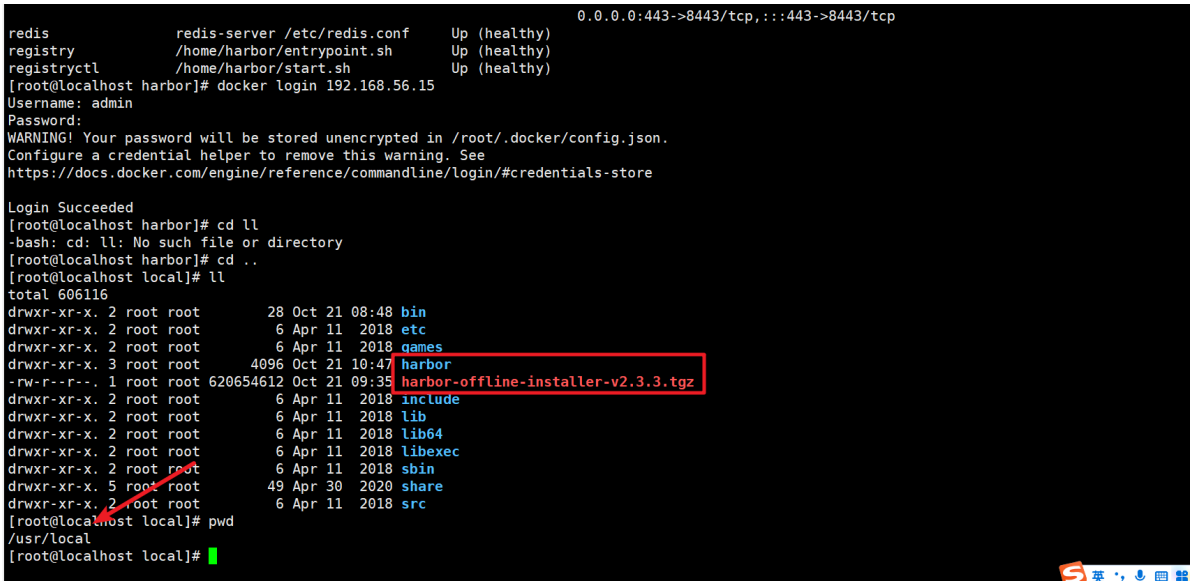
官网安装教程：https://goharbor.io/docs/2.3.0/install-config/

首先需要下载对于的安装文件：https://github.com/goharbor/harbor/releases

因为比较大，从官网下载很慢，所以会在附近中直接给大家提供，使用最新的2.3.3版本

1> 上传解压安装文件

```
cd /usr/local/
```



2>修改配置文件

把 harbor.yml.tmpl 修改为 harbor.yml 文件

```yaml
hostname: 192.168.56.15
http:
 port: 80
https:
  port: 443
  certificate: /data/cert/192.168.56.15.crt
  private_key: /data/cert/192.168.56.15.key
external_url: https://192.168.56.15
harbor_admin_password: Harbor12345
database:
  password: root123
  max_idle_conns: 50
  max_open_conns: 100
data_volume: /data/harbor
clair:
  updaters_interval: 12
jobservice:
  max_job_workers: 10
notification:
  webhook_job_max_retry: 10
chart:
  absolute_url: disabled
log:
  level: info
  local:
    rotate_count: 50
    rotate_size: 200M
    location: /data/harbor/logs
_version: 1.10.0
proxy:
  http_proxy:
  https_proxy:
  no_proxy:
  components:
    - core
```

```
    - jobservice
    - clair
```

3>.harbor配置 https 访问

参考文档:

https://goharbor.io/docs/1.10/install-config/configure-https/

https://goharbor.io/docs/1.10/install-config/troubleshoot-installation/#https

默认情况下，Harbor不附带证书。可以在没有安全性的情况下部署Harbor，以便您可以通过HTTP连接到它。但是，只有在没有外部网络连接的空白测试或开发环境中，才可以使用HTTP。在没有空隙的环境中使用HTTP会使您遭受中间人攻击。在生产环境中，请始终使用HTTPS。如果启用Content Trust with Notary来正确签名所有图像，则必须使用HTTPS。

要配置HTTPS，必须创建SSL证书。您可以使用由受信任的第三方CA签名的证书，也可以使用自签名证书

生成证书颁发机构证书

在生产环境中，您应该从CA获得证书。在测试或开发环境中，您可以生成自己的CA。要生成CA证书，请运行以下命令。

生成CA证书私钥。

```
openssl genrsa -out ca.key 4096
```

生成CA证书

调整`-subj`选项中的值以反映您的组织。如果使用FQDN连接Harbor主机，则必须将其指定为通用名称（`CN`）属性。

```
openssl req -x509 -new -nodes -sha512 -days 3650 \
 -subj "/C=CN/ST=Beijing/L=Beijing/O=example/OU=Personal/CN=harbor.od.com" \
 -key ca.key \
 -out ca.crt
```

如果是ip访问， 将 `harbor.od.com` 改成 ip地址

生成服务器证书

证书通常包含一个`.crt`文件和一个`.key`文件

生成私钥

```
openssl genrsa -out harbor.od.com.key 4096
```

生成证书签名请求（CSR）

```
openssl req -sha512 -new \
    -subj "/C=CN/ST=Beijing/L=Beijing/O=example/OU=Personal/CN=harbor.od.com" \
    -key harbor.od.com.key \
    -out harbor.od.com.csr
```

如果是ip访问， 将 `harbor.od.com` 改成 ip地址

生成一个x509 v3扩展文件

无论您使用FQDN还是IP地址连接到Harbor主机，都必须创建此文件，以便可以为您的Harbor主机生成符合主题备用名称（SAN）和x509 v3的证书扩展要求。替换 `DNS` 条目以反映您的域

```
cat > v3.ext <<-EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
DNS.1=harbor.od.com
DNS.2=harbor.od.com
DNS.3=harbor.od.com
EOF
```

- 如果是ip访问

```
cat > v3.ext <<-EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = IP:192.168.56.15
EOF
```

使用该 `v3.ext` 文件为您的Harbor主机生成证书

```
openssl x509 -req -sha512 -days 3650 \
    -extfile v3.ext \
    -CA ca.crt -CAkey ca.key -CAcreateserial \
    -in harbor.od.com.csr \
    -out harbor.od.com.crt
```

如果是ip访问， 将 `harbor.od.com` 改成 ip地址

提供证书给Harbor和Docker

生成后 `ca.crt` ， `harbor.od.com.crt` 和 `harbor.od.com.key` 文件，必须将它们提供给 `Harbor` 和 `docker` ，重新配置它们

将服务器证书和密钥复制到Harbor主机上的 `/data/cert/` 文件夹中

```
mkdir -p /data/cert/
cp harbor.od.com.crt /data/cert/
cp harbor.od.com.key /data/cert/
```

转换`harbor.od.com.crt`为`harbor.od.com.cert`，供Docker使用

```
openssl x509 -inform PEM -in harbor.od.com.crt -out harbor.od.com.cert
```

将服务器证书，密钥和CA文件复制到Harbor主机上的Docker证书文件夹中。您必须首先创建适当的文件夹

```
mkdir -p /etc/docker/certs.d/harbor.od.com/
cp harbor.od.com.cert /etc/docker/certs.d/harbor.od.com/
cp harbor.od.com.key /etc/docker/certs.d/harbor.od.com/
cp ca.crt /etc/docker/certs.d/harbor.od.com/
```

如果将默认`nginx`端口443 映射到其他端口，请创建文件夹`/etc/docker/certs.d/yourdomain.com:port`或`/etc/docker/certs.d/harbor_IP:port`

重新启动Docker Engine

```
systemctl restart docker
```

证书目录结构

```
/etc/docker/certs.d/
└── harbor.od.com
    ├── ca.crt
    ├── harbor.od.com.cert
    └── harbor.od.com.key
```

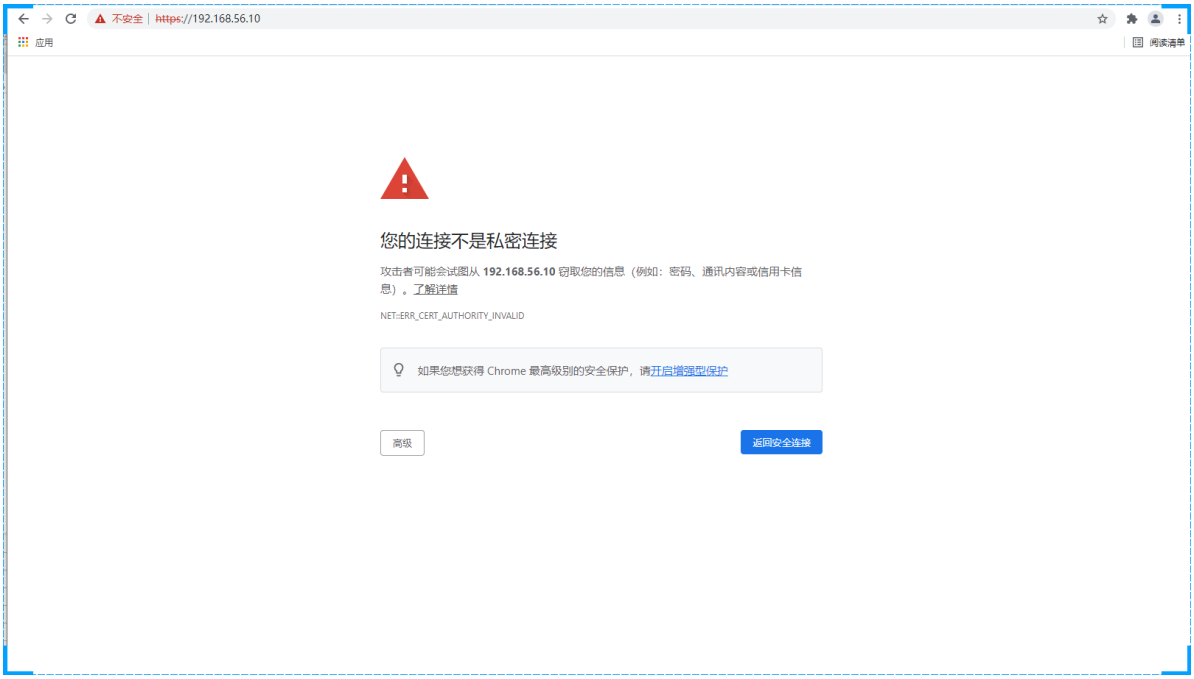
Harbor将`nginx`实例用作所有服务的反向代理。您可以使用`prepare`脚本来配置`nginx`为使用HTTPS

```
./prepare
```

初始化服务

```
sh install.sh
```

```
[Step 4]: preparing harbor configs ...
prepare base dir is set to /usr/local/harbor
Generated configuration file: /config/portal/nginx.conf
Generated configuration file: /config/log/logrotate.conf
Generated configuration file: /config/log/rsyslog_docker.conf
Generated configuration file: /config/nginx/nginx.conf
Generated configuration file: /config/core/env
Generated configuration file: /config/core/app.conf
Generated configuration file: /config/registry/config.yml
Generated configuration file: /config/registryctl/env
Generated configuration file: /config/registryctl/config.yml
Generated configuration file: /config/db/env
Generated configuration file: /config/jobservice/env
Generated configuration file: /config/jobservice/config.yml
Generated and saved secret to file: /data/secret/keys/secretkey
Successfully called func: create_root_cert
Generated configuration file: /compose_location/docker-compose.yml
Clean up the input dir



[Step 5]: starting Harbor ...
Creating network "harbor_harbor" with the default driver
Creating harbor-log ... done
Creating harbor-db      ... done
Creating redis          ... done
Creating registry       ... done
Creating registryctl    ... done
Creating harbor-portal  ... done
Creating harbor-core    ... done
Creating nginx              ... done
Creating harbor-jobservice ... done
✔ ----Harbor has been installed and started successfully.----
[root@localhost harbor]# ll
```



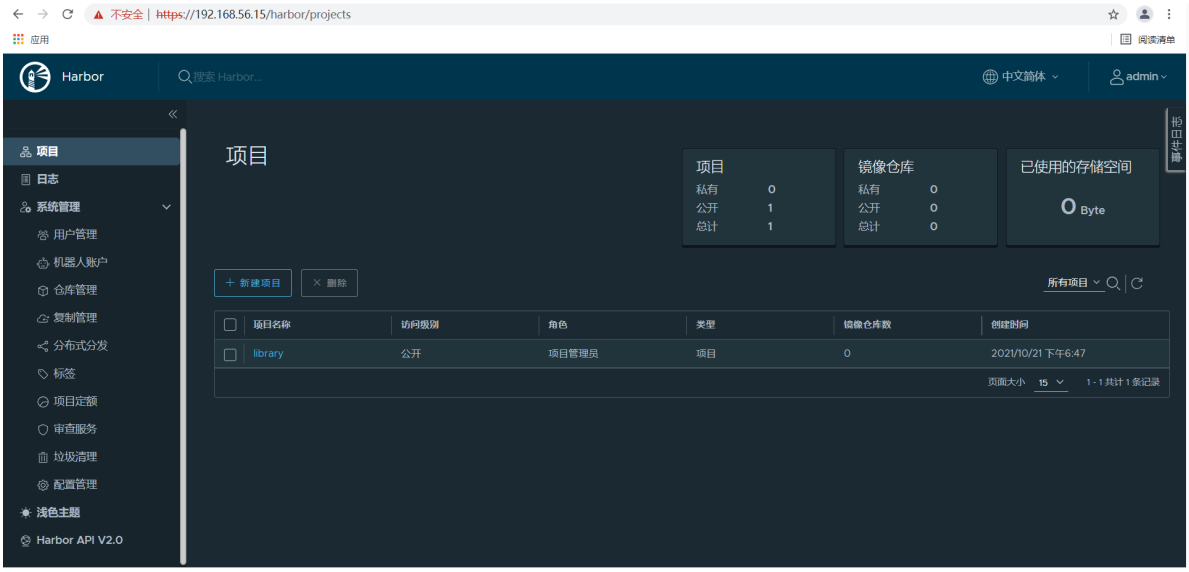
停止并删除服务

```
docker-compose down -v
```

重启服务

```
docker-compose up -d
```

连接验证



docker登录

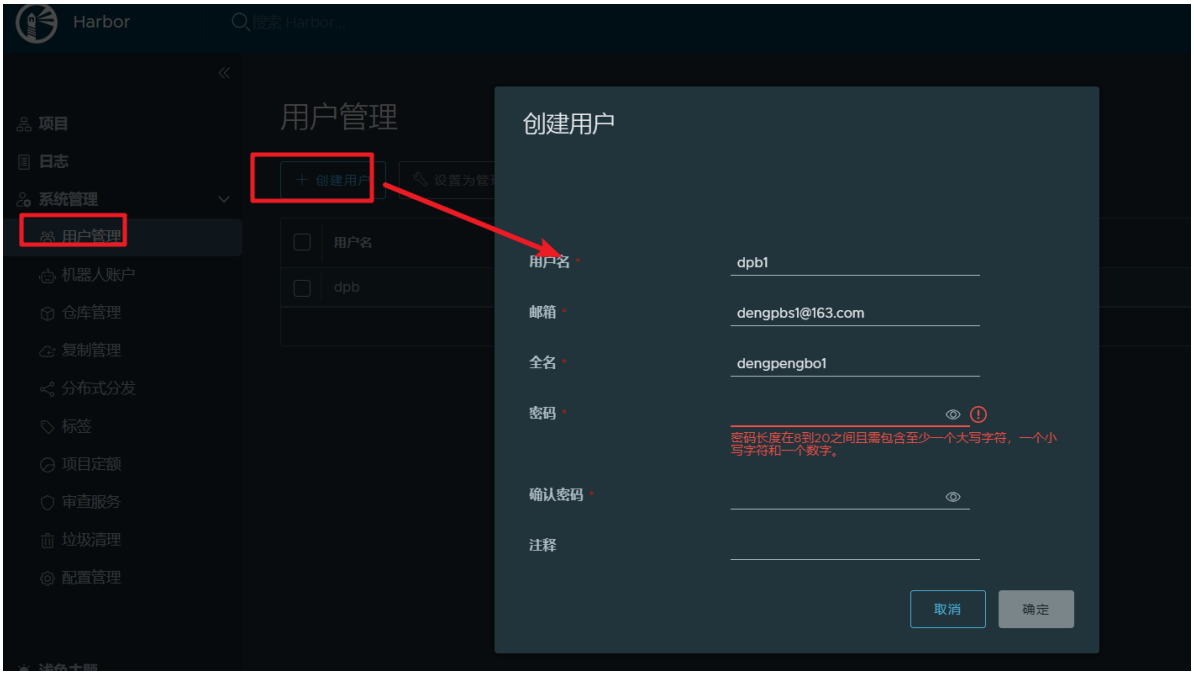
```
[root@localhost harbor]# docker login 192.168.56.15
Username: admin
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

4.3.4 Harbor部署应用

上传镜像到Harbor服务中

在Web服务中创建项目和用户

创建用户



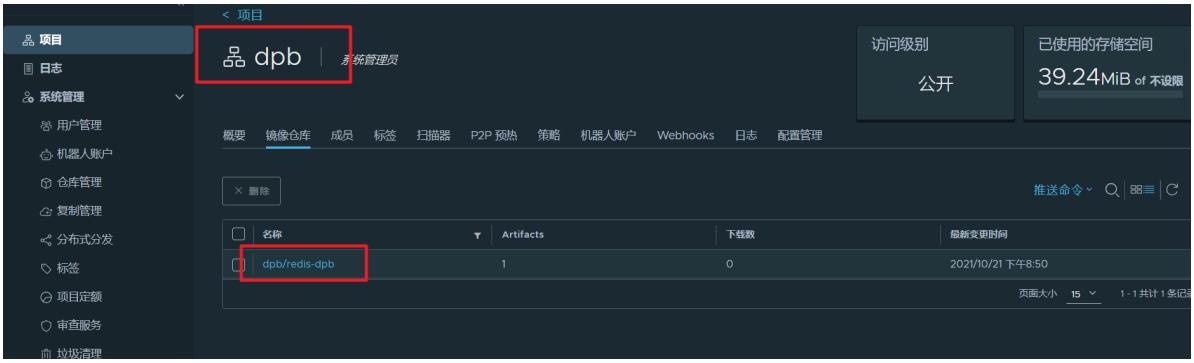
然后项目分配用户

推送镜像到Harbor仓库中

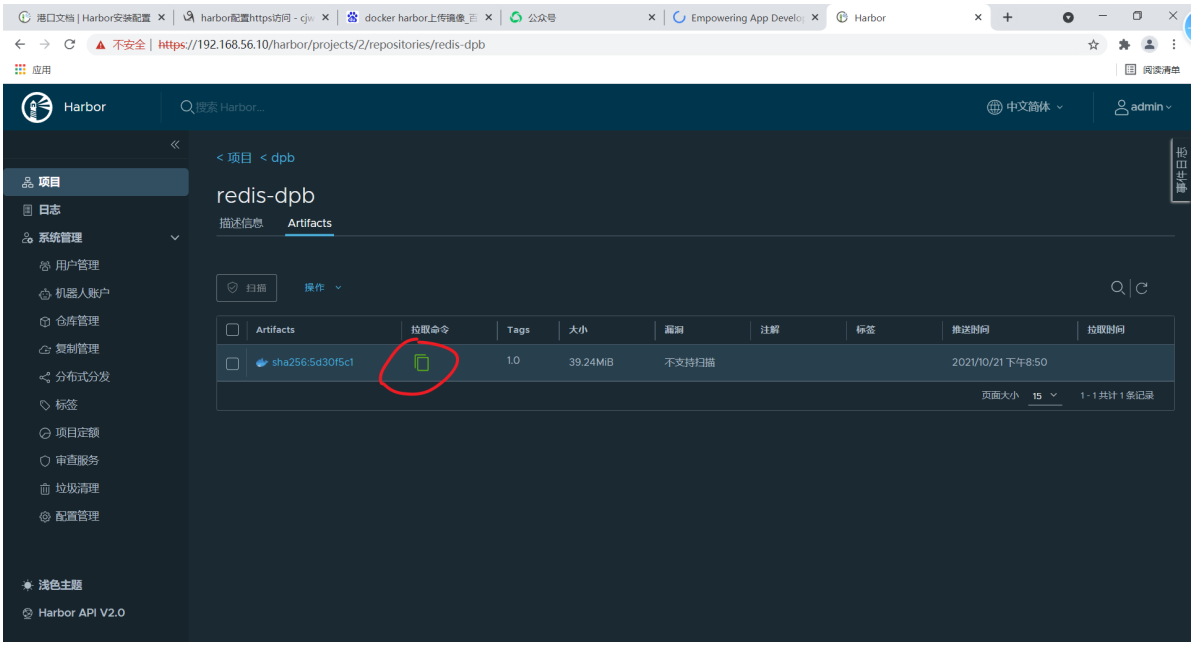
```
docker tag redis:latest 192.68.56.10/dpb/redis-dpb:1.0
docker push 192.168.56.10/dpb/redis-dpb:1.0
```

```
[root@localhost harbor]# docker push 192.168.56.10/dpb/redis-dpb:1.0
The push refers to repository [192.168.56.10/dpb/redis-dpb]
146262eb3841: Pushed
0bd13b42de4d: Pushed
6b01cc47a390: Pushed
8b9770153666: Pushed
b43651130521: Pushed
e8b689711f21: Pushed
1.0: digest: sha256:5d30f5c16e473549ad7c950b0ac3083039719b1c9749519c50e18017dd4bfc54 size: 1573
[root@localhost harbor]# ll
```

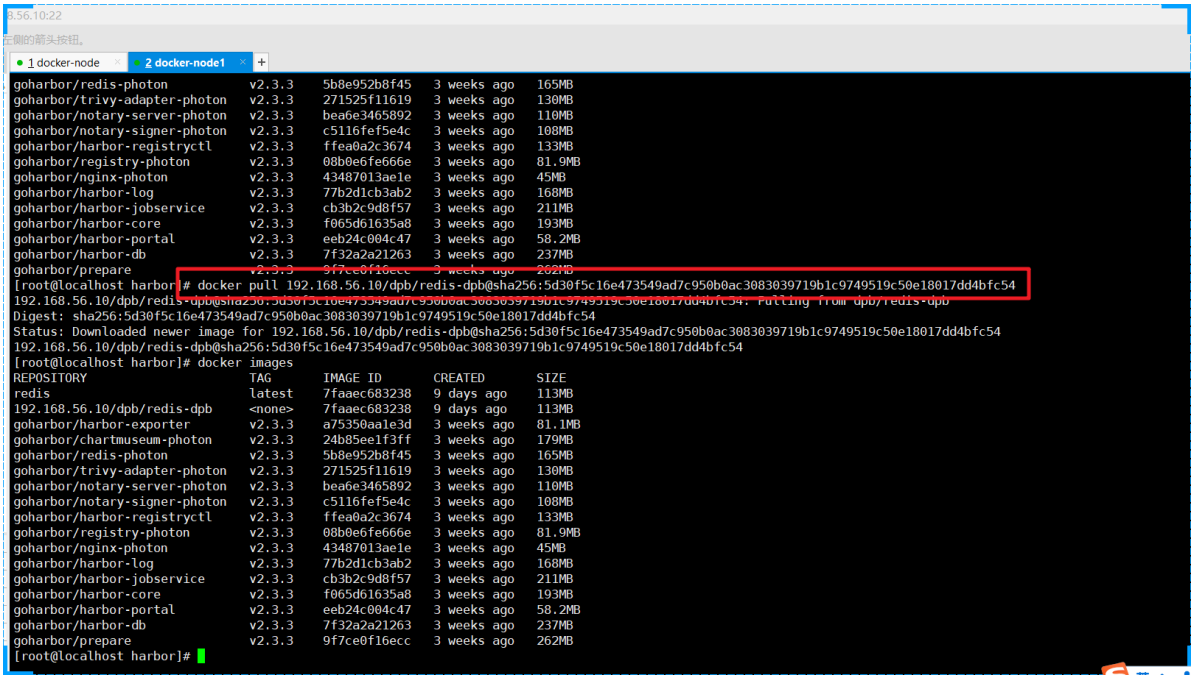
在Web项目中可以看到上传的镜像



从Harbor镜像仓库拉取镜像



执行拉取命令



搞定~

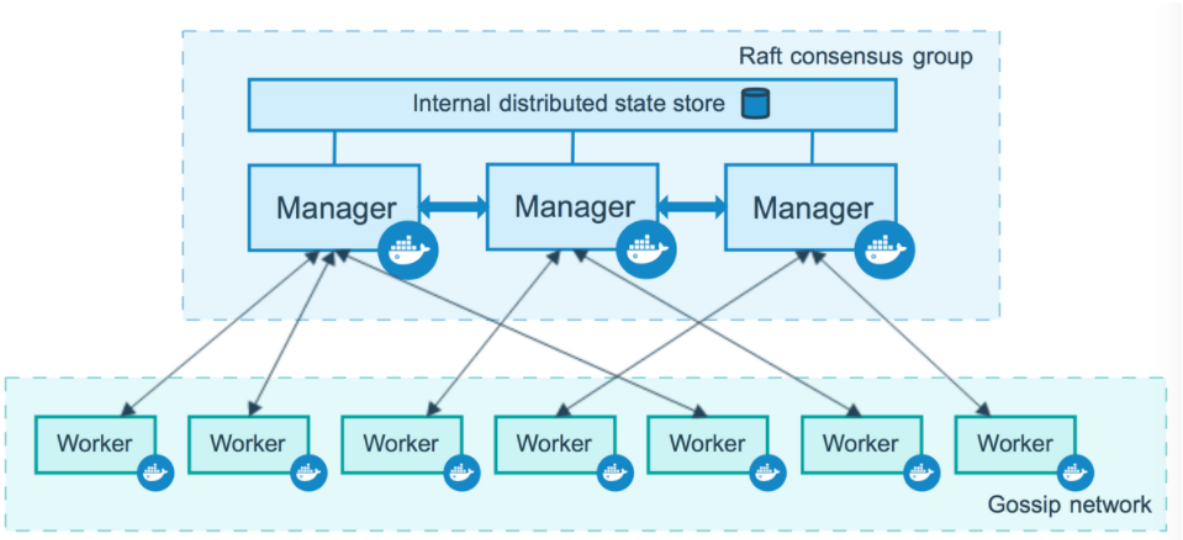
5. Swarm

解决集群问题 k8s

官网地址：https://docs.docker.com/engine/swarm/

5.1 Swarm介绍

Swarm是Docker官方提供的一款集群管理工具，其主要作用是把若干台Docker主机抽象为一个整体，并且通过一个入口统一管理这些Docker主机上的各种Docker资源。Swarm和Kubernetes比较类似，但是更加轻，具有的功能也较kubernetes更少一些



管理节点

管理节点处理集群管理任务：

- 维护集群状态
- 调度服务
- 服务群模式HTTP API 端点

使用`Raft`实现，管理器维护整个 swarm 及其上运行的所有服务的一致内部状态。出于测试目的，可以使用单个管理器运行 swarm。如果单管理器群中的管理器出现故障，您的服务会继续运行，但您需要创建一个新集群来恢复。

为了利用 swarm 模式的容错特性，Docker 建议您根据组织的高可用性要求实现奇数个节点。当您有多个管理器时，您可以在不停机的情况下从管理器节点的故障中恢复。

- 三个管理器的群体最多可以容忍一个管理器的损失。
- 一个五管理器群可以容忍最大同时丢失两个管理器节点。
- 一个`N`管理器集群最多可以容忍管理器的丢失`(N-1)/2`。
- Docker 建议一个群最多有七个管理器节点。

工作节点

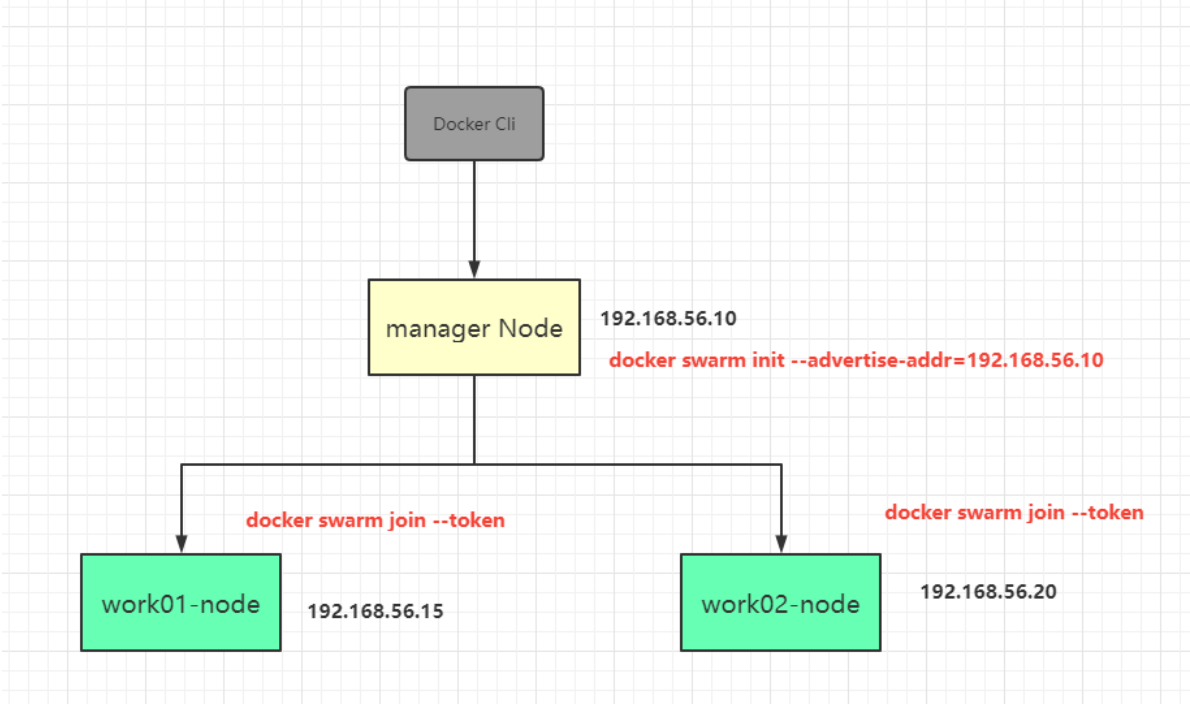
工作节点也是 Docker 引擎的实例，其唯一目的是执行容器。Worker 节点不参与 Raft 分布式状态，不做出调度决策，也不为 swarm 模式 HTTP API 提供服务。

您可以创建一个由一个管理器节点组成的群，但是如果没有至少一个管理器节点，您就不能拥有一个工作节点。默认情况下，所有经理也是工人。在单个管理器节点集群中，您可以运行类似命令`docker service create`，调度程序将所有任务放在本地引擎上。

为防止调度程序将任务放置在多节点群中的管理器节点上，请将管理器节点的可用性设置为`Drain`。调度器在`Drain`mode 中优雅地停止节点上的任务并调度`Active`节点上的任务 。调度程序不会将新任务分配给具有`Drain` 可用性的节点。

5.2 Swarm集群搭建

环境准备



准备3个节点，通过vagrant新增加两个节点

需要单独指定hostname

```
config.vm.hostname="work01-node"
```

还有就是每个节点需要具备Docker环境

集群环境搭建

1> 创建manager节点

进入manager节点，manager node也可以作为worker node提供服务

```
docker swarm init -advertise 192.168.56.10
```

`注意观察日志，拿到worker node加入manager node的信息`

```
docker swarm join --token SWMTKN-1-
0a5ph4nehwdm9wzcmlbj2ckqqso38pkd238rprzwcoawabxtdq-arcpra6yzltedpafk3qyvv0y3
192.168.56.10:2377
```

```
Run 'docker swarm COMMAND --help' for more information on a command.
[root@manager-node ~]# docker swarm init --advertise-addr 192.168.56.10
Swarm initialized: current node (z3a1dddvxn9w83zoqrbdbjgry) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-5zu2bhm7ftgptu87bkvqxdn2wxd92bil7gjow07ixjae48aai8-13exz1izbr2lsy32epl63nwuk 192.168.56.10:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[root@manager-node ~]#
```

2>进入两个Worker

```
docker swarm join --token SWMTKN-1-
0a5ph4nehwdm9wzcmlbj2ckqqso38pkd238rprzwcoawabxtdq-arcpra6yzltedpafk3qyvv0y3
192.168.56.10:2377
```

3>进入manager node 查看集群情况

```
docker node ls
```

4>node类型转换

可以将worker提升成manager，从而保证manager的高可用

```
docker node promote worker01-node
docker node promote worker02-node

#降级可以用demote
docker node demote worker01-node
```

在线Swarm演示：http://labs.play-with-docker.com 通过Dock Hub 的账号密码登录即可，有效会话4个小时

5.3 Raft一致性协议

Raft一致性协议：保证manager节点半数存活集群环境可用

一主两从

还是参考上面的案例直接操作

```
[root@manager-node ~]# docker node ls
ID                            HOSTNAME       STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
wmopfh8pme4m6qkx5kpibk2qa *   manager-node   Ready     Active         Leader           20.10.9
nz7j6fylopqodld9hsmo896yy     work01-node    Ready     Active                          20.10.9
irzuvtgq6wz6eblymx62eu491     work02-node    Ready     Active                          20.10.9
[root@manager-node ~]#
```

只有一个manager节点，而且没有备用的manager节点

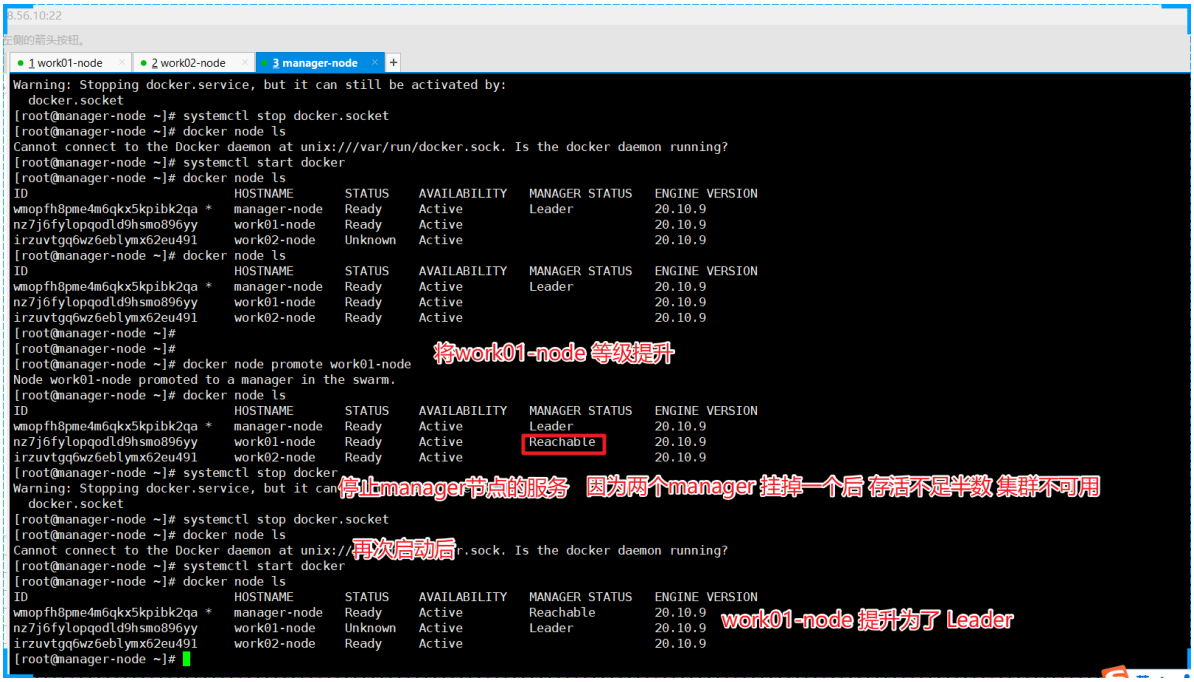
我们停掉manager节点，那么整个集群环境是不可用的

```
[root@manager-node ~]# docker node ls
ID                            HOSTNAME       STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
wmopfh8pme4m6qkx5kpibk2qa *   manager-node   Ready     Active         Leader           20.10.9
irzuvtgq6wz6eblymx62eu491     work02-node    Ready     Active                          20.10.9
[root@manager-node ~]# docker node ls
ID                            HOSTNAME       STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
wmopfh8pme4m6qkx5kpibk2qa *   manager-node   Ready     Active         Leader           20.10.9
nz7j6fylopqodld9hsmo896yy     work01-node    Ready     Active                          20.10.9
irzuvtgq6wz6eblymx62eu491     work02-node    Ready     Active                          20.10.9
[root@manager-node ~]# systemctl stop docker
Warning: Stopping docker.service, but it can still be activated by:
  docker.socket
[root@manager-node ~]# systemctl stop docker.socket
[root@manager-node ~]# docker node ls
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
[root@manager-node ~]# systemctl start docker
[root@manager-node ~]# docker node ls
ID                            HOSTNAME       STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
wmopfh8pme4m6qkx5kpibk2qa *   manager-node   Ready     Active         Leader           20.10.9
nz7j6fylopqodld9hsmo896yy     work01-node    Ready     Active                          20.10.9
irzuvtgq6wz6eblymx62eu491     work02-node    Unknown   Active                          20.10.9
[root@manager-node ~]# docker node ls
ID                            HOSTNAME       STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
wmopfh8pme4m6qkx5kpibk2qa *   manager-node   Ready     Active         Leader           20.10.9
nz7j6fylopqodld9hsmo896yy     work01-node    Ready     Active                          20.10.9
irzuvtgq6wz6eblymx62eu491     work02-node    Ready     Active                          20.10.9
[root@manager-node ~]#
```

docker 停止后不可用

docker启动后又可以使用

我们将一个work节点提升等级



二主一从

除了上面的 `promote` 提升到 主的案例意外，我们还可用从新来搭集群处理

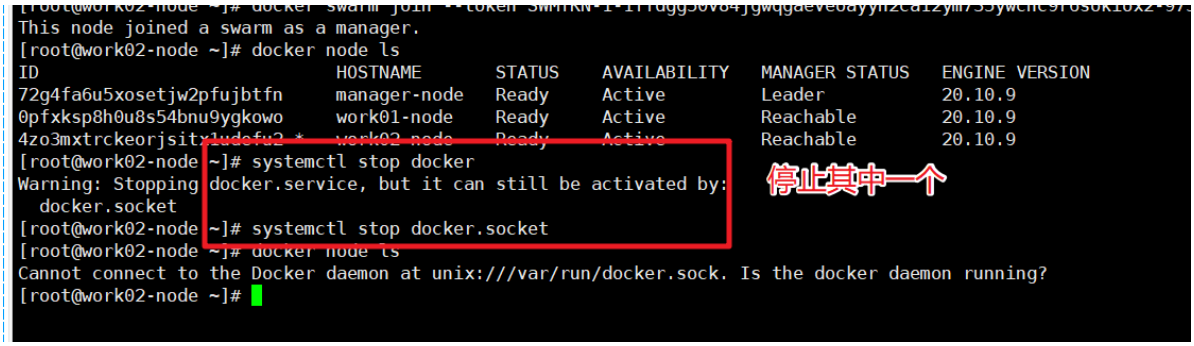
我们可以在init后直接在 manager节点执行如下命令

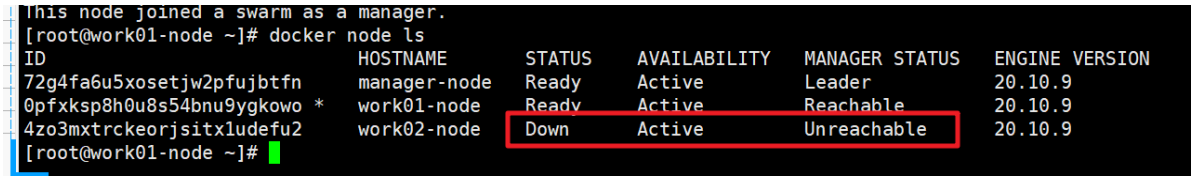
```
docker swarm join-token manager
```

三主0从

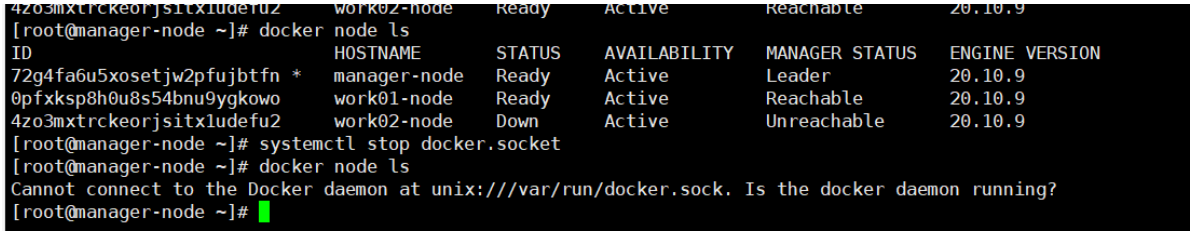
```
[root@manager-node ~]# docker node ls
ID                            HOSTNAME       STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
72g4fa6u5xosetjw2pfujbtfn *   manager-node   Ready     Active         Leader           20.10.9
4zo3mxtrckeorjsitx1udefu2     work02-node    Ready     Active         Reachable        20.10.9
[root@manager-node ~]# docker node ls
ID                            HOSTNAME       STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
72g4fa6u5xosetjw2pfujbtfn *   manager-node   Ready     Active         Leader           20.10.9
0pfxksp8h0u8s54bnu9ygkowo     work01-node    Ready     Active         Reachable        20.10.9
4zo3mxtrckeorjsitx1udefu2     work02-node    Ready     Active         Reachable        20.10.9
[root@manager-node ~]#
```

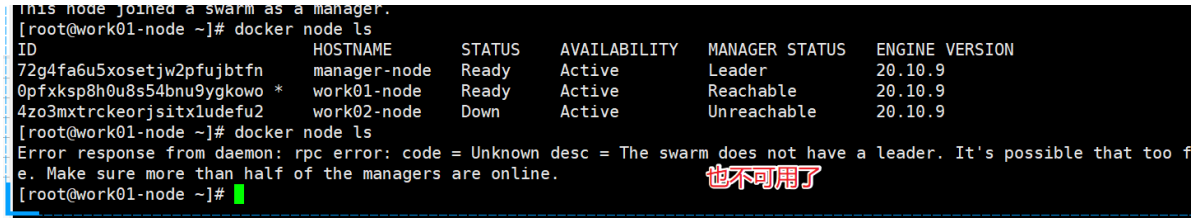
停止其中一个整个集群还是可用





但是停止掉两个后就不可用使用了





5.4 Service

(1)创建一个tomcat的service

```
docker service create --name my-tomcat tomcat
```

(2)查看当前swarm的service

```
docker service ls
```

(3)查看service的启动日志

```
docker service logs my-tomcat
```

(4)查看service的详情

```
docker service inspect my-tomcat
```

(5)查看my-tomcat运行在哪个node上

```
docker service ps my-tomcat
```

日志

```
ID                  NAME                IMAGE               NODE
DESIRED STATE       CURRENT STATE           ERROR               PORTS
u6o4mz4tj396        my-tomcat.1         tomcat:latest       worker01-node
Running             Running 3 minutes ago
```

(6)水平扩展service

```
docker service scale my-tomcat=3
docker service ls
docker service ps my-tomcat
```

日志：可以发现，其他node上都运行了一个my-tomcat的service

```
[root@manager-node ~]# docker service ps my-tomcat
ID                  NAME                IMAGE               NODE
DESIRED STATE       CURRENT STATE           ERROR               PORTS
u6o4mz4tj396        my-tomcat.1         tomcat:latest       worker01-node
Running             Running 8 minutes ago
v5O5wdu3fxqo        my-tomcat.2         tomcat:latest       manager-node
Running             Running 46 seconds ago
wpbsilp62sc0        my-tomcat.3         tomcat:latest       worker02-node
Running             Running 49 seconds ago
```

此时到worker01-node上：docker ps，可以发现container的name和service名称不一样，这点要知道

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
bc4b9bb097b8        tomcat:latest       "catalina.sh run"   10 minutes ago
Up 10 minutes       8080/tcp            my-tomcat.1.u6o4mz4tj3969a1p3mquagxok
```

> (7)如果某个node上的my-tomcat挂掉了，这时候会自动扩展

```
[worker01-node]
docker rm -f containerid

[manager-node]
docker service ls
docker service ps my-tomcat
```

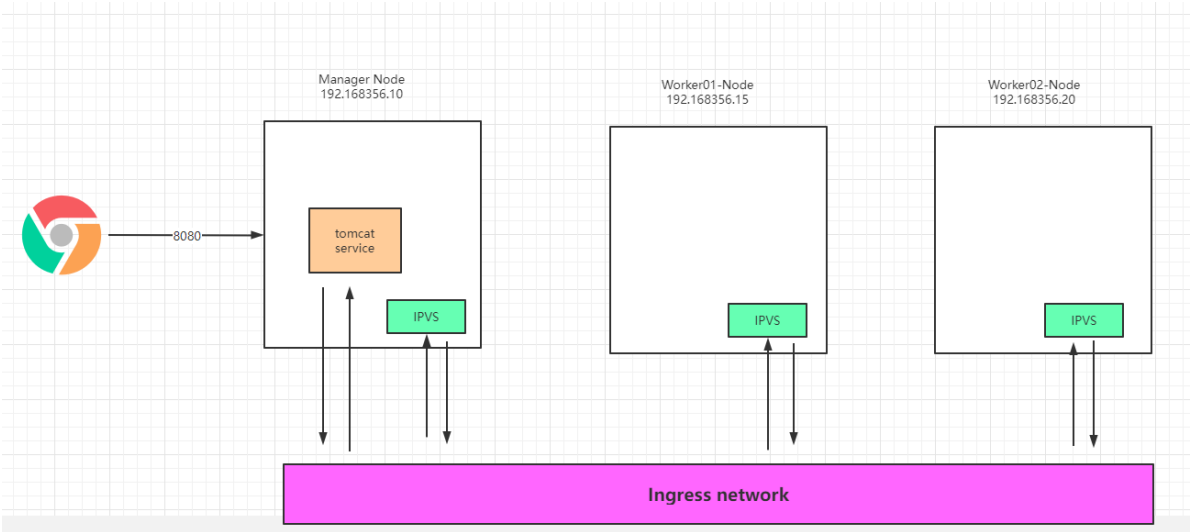
> (8)删除service

```
docker service rm my-tomcat
```

5.5 overlay

Overlay 在网络技术领域，指的是一种网络架构上叠加的虚拟化技术模式，其大体框架是对基础网络不进行大规模修改的条件下，实现应用在网络上的承载，并能与其它网络业务分离，并且以基于IP的基础网络技术为主

VXLAN（Virtual eXtensible LAN）技术是当前最为主流的Overlay标准



5.5 WordPress实战

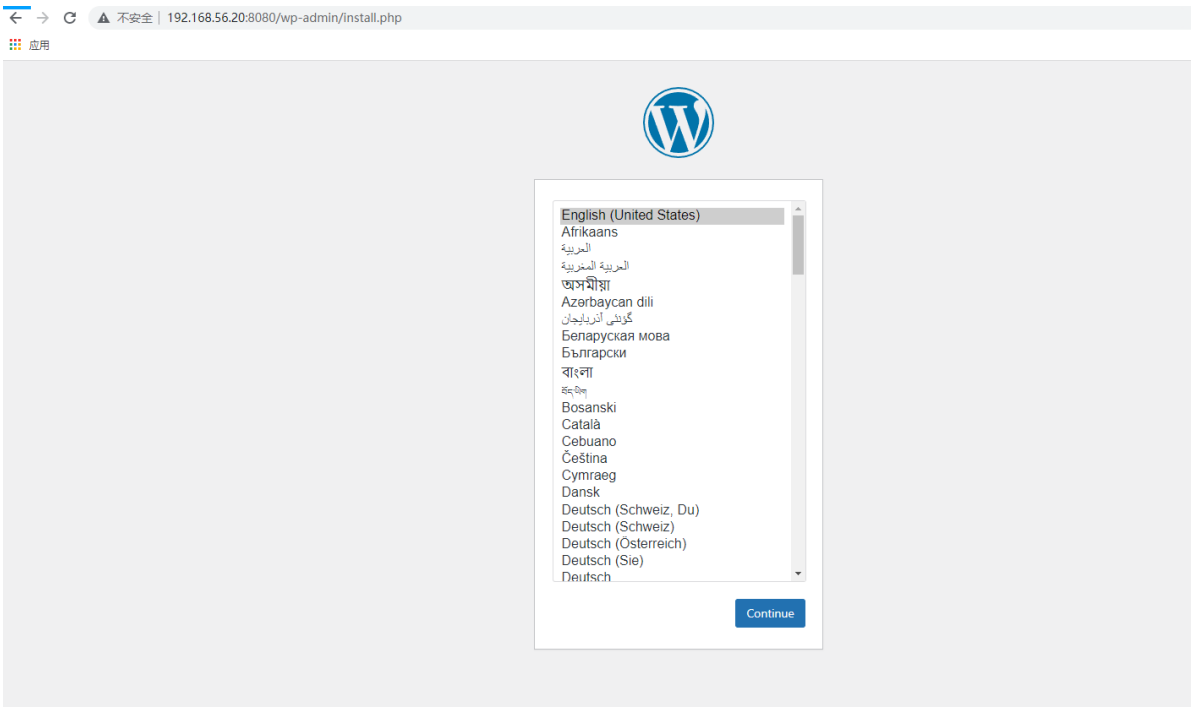
1> 创建MySQL service

```
docker service create --name mysql --mount
type=volume,source=v1,destination=/var/lib/mysql --env
MYSQL_ROOT_PASSWORD=examplepass --env MYSQL_DATABASE=db_wordpress --network my-
overlay-net mysql:5.6
```

2>创建WordPress的Service

```
docker service create --name wordpress --env WORDPRESS_DB_USER=root --env
WORDPRESS_DB_PASSWORD=examplepass --env WORDPRESS_DB_HOST=mysql:3306 --env
WORDPRESS_DB_NAME=db_wordpress -p 8080:80 --network my-overlay-net wordpress
```

3>访问测试



4>查看my-overlay-net

```
[root@work02-node ~]# docker network inspect my-overlay-net
[
    {
        "Name": "my-overlay-net",
        "Id": "jenkwf4hjkr25u3ihubso1qmi",
        "Created": "2021-10-21T17:54:54.470525693Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.1.0/24",
                    "Gateway": "10.0.1.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "e247813859bb30a48e40d32d4633369983cd7a605f9f75d88fe0af8a0d251063": {
                "Name": "mysql.1.v5jbaonxos77nqqb00e0x9qlk",
                "EndpointID": "0de469f2f40db4387ab545978978d199b7e0d255f29a79a2754e95460a1dcb0a",
                "MacAddress": "02:42:0a:00:01:03",
                "IPv4Address": "10.0.1.3/24",
```