Docker

Java 是第一大编程语言和开发平台。它有助于企业降低成本、缩短开发周期、推动创新以及改善应用服务。如今全球有数百万开发人员运行着超过 51 亿个 Java 虚拟机，Java 仍是企业和开发人员的首选开发平台

课程内容的介绍

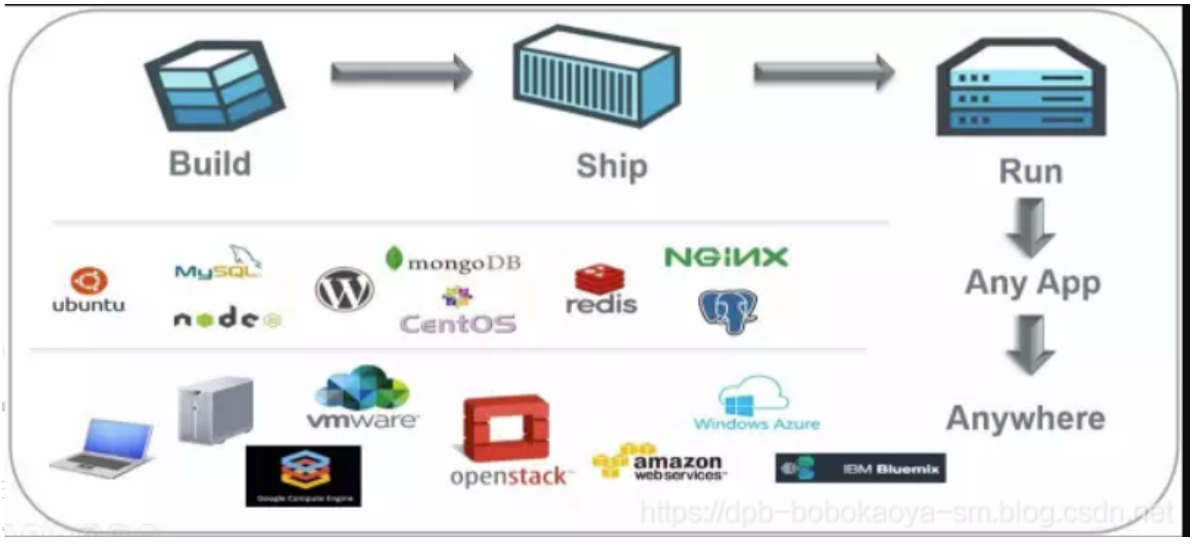
1. Docker介绍及安装
2. Docker常用命令介绍
3. Docker镜像介绍
4. Docker数据卷
5. Docker常用软件安装

一、Docker介绍及安装

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

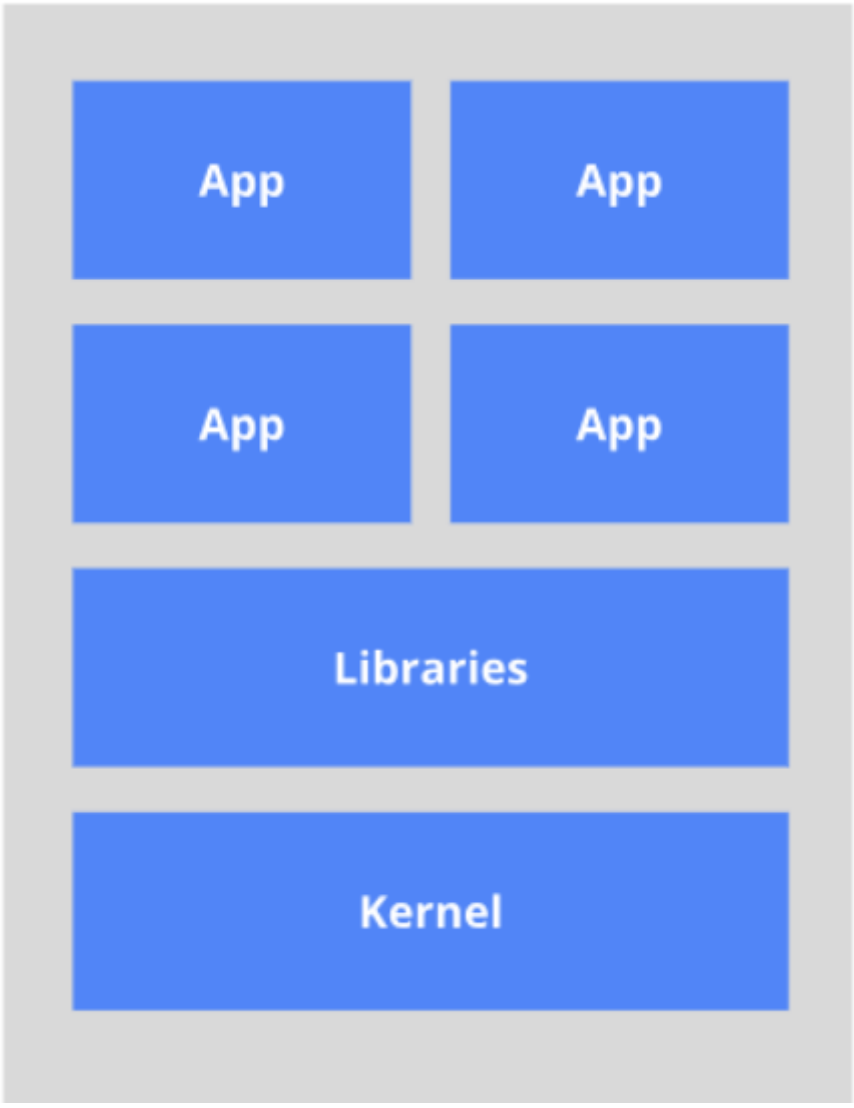
> 解决了运行环境和配置问题软件容器，方便做持续集成并有助于整体发布的容器虚拟化技术

2.Docker能干吗?

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。



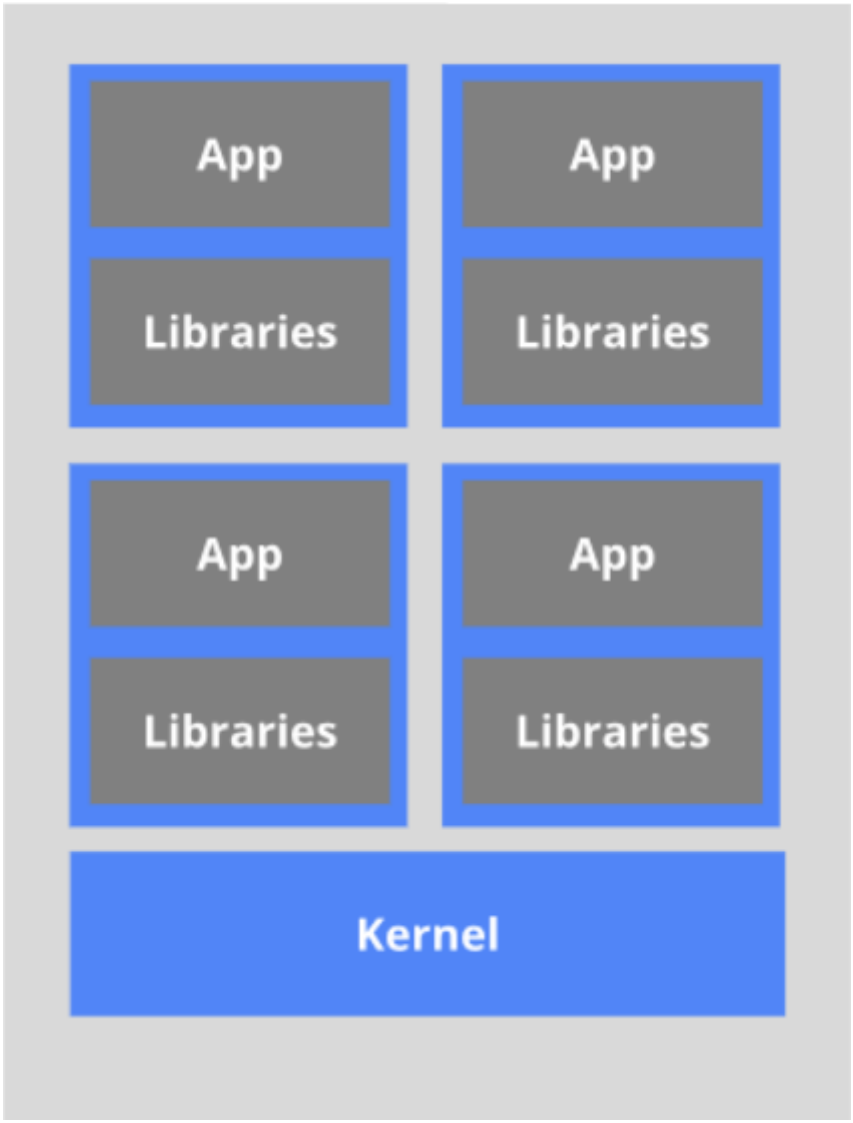
虚拟机的缺点:

1. 资源占用多
2. 冗余步骤多
3. 启动慢

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。

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

3.相关资源

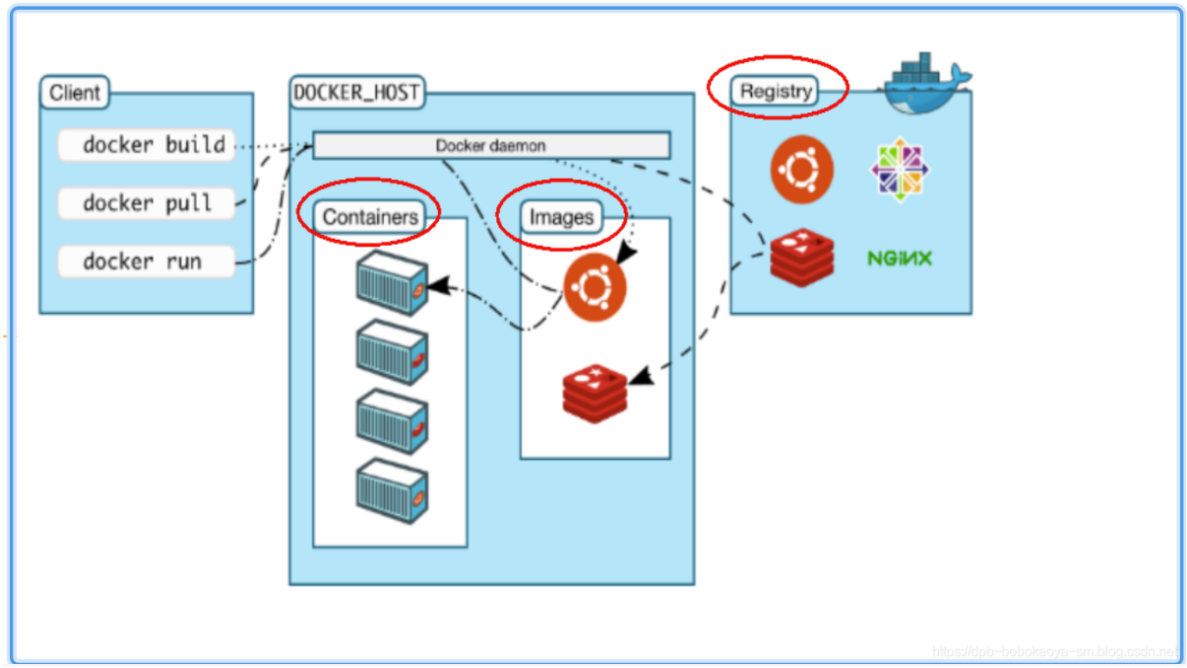
官网: http://www.docker.com
仓库: https://hub.docker.com

4.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）。

5.docker的基本组成



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

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。

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

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。

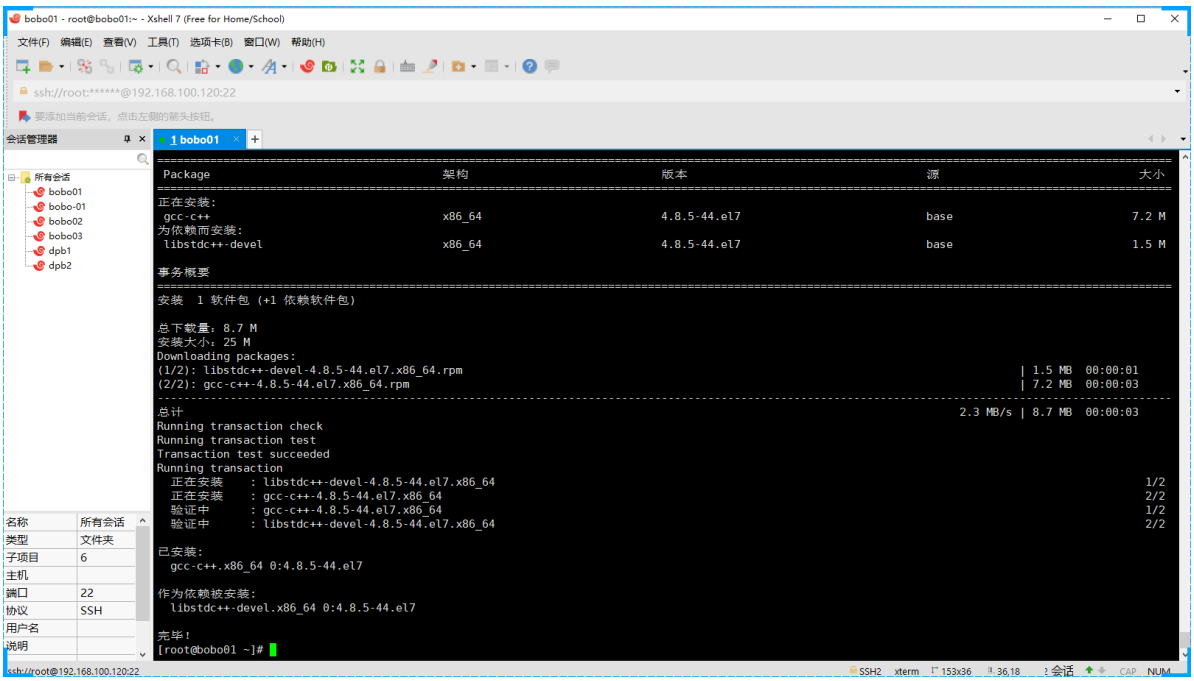
6. Docker安装

6.1 确认版本

```
[root@bobo01 ~]# cat /etc/redhat-release
CentOS Linux release 7.9.2009 (Core)
```

6.2 安装gcc相关

```
yum -y install gcc gcc-c++
```

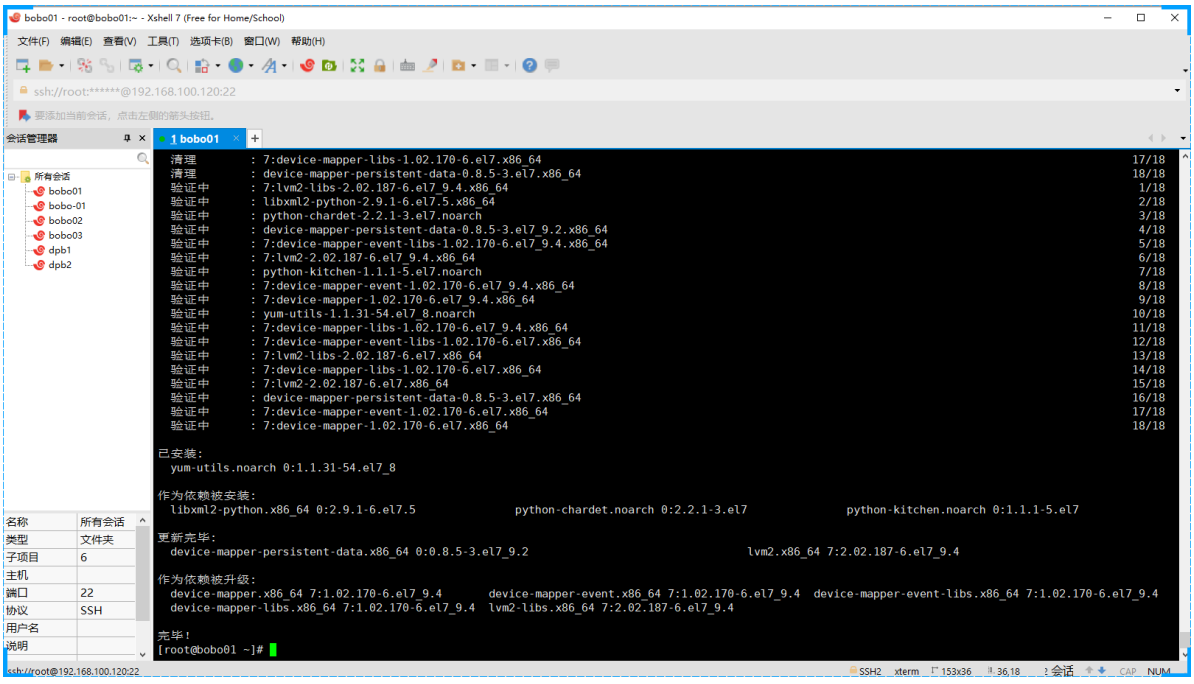


6.3 卸载旧版本

```
[root@bobo01 ~]# yum -y remove docker docker-common docker-selinux docker-engine
已加载插件：fastestmirror
参数 docker 没有匹配
参数 docker-common 没有匹配
参数 docker-selinux 没有匹配
参数 docker-engine 没有匹配
不删除任何软件包
```

6.4 安装软件包

```
yum install -y yum-utils device-mapper-persistent-data lvm2
```



6.5 设置镜像仓库

```
[root@bobo01 ~]# yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
已加载插件：fastestmirror
adding repo from: http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
grabbing file http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
```

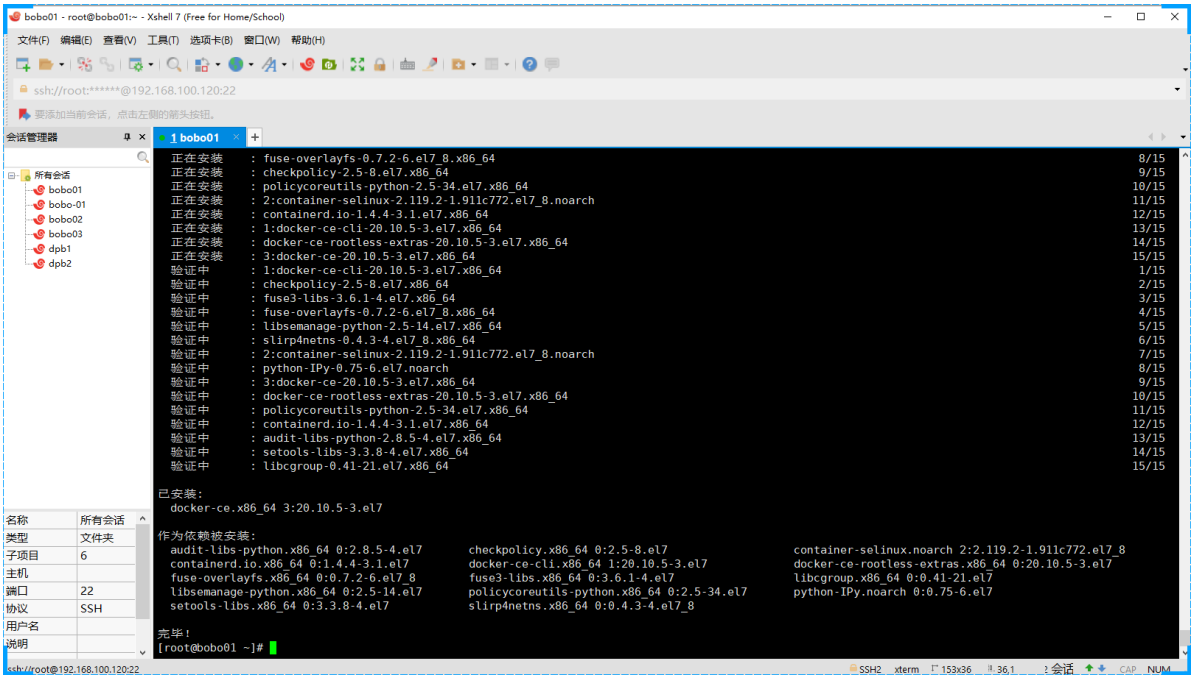
6.6 yum更新

```
yum makecache fast
```

6.7 Docker安装

CE 是官方提供的免费版本，学习足够了

```
yum -y install docker-ce
```



6.8 启动Docker

```
systemctl start docker
```

查看版本

```
[root@bobo01 ~]# docker version
Client: Docker Engine - Community
 Version:           20.10.5
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        55c4c88
 Built:             Tue Mar  2 20:33:55 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.5
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       363e9a8
  Built:            Tue Mar  2 20:32:17 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.4
  GitCommit:        05f951a3781f4f2c1911b05e61c160e9c30eaa8e
 runc:
  Version:          1.0.0-rc93
  GitCommit:        12644e614e25b05da6fd08a38ffa0cfe1903fdec
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

6.9 HelloWorld案例

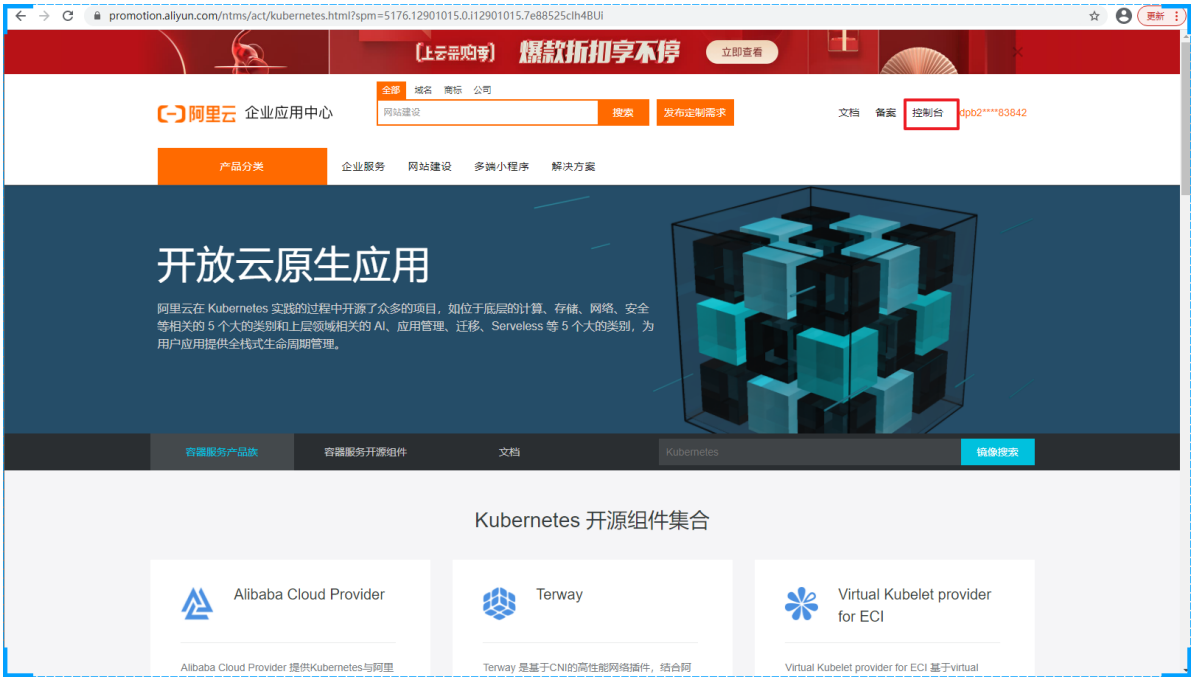
```
[root@bobo01 ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:308866a43596e83578c7dfa15e27a73011bdd402185a84c5cd7f32a88b501a24
Status: Downloaded newer image for hello-world:latest

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.
```

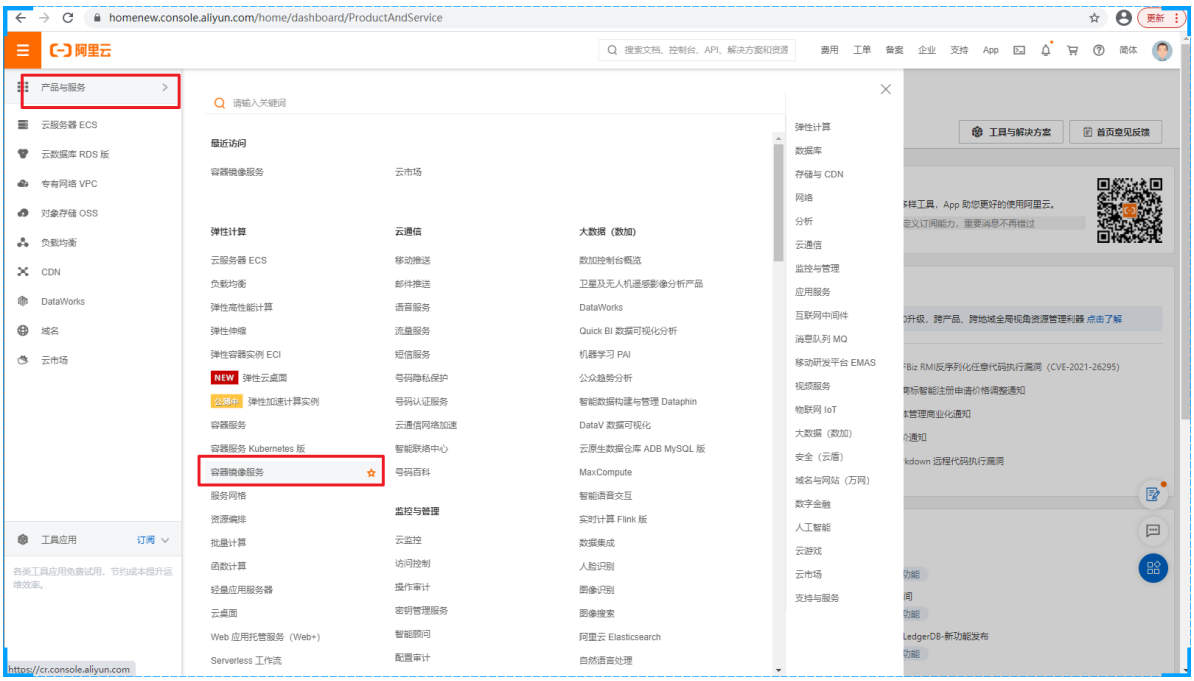
```
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

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/
```

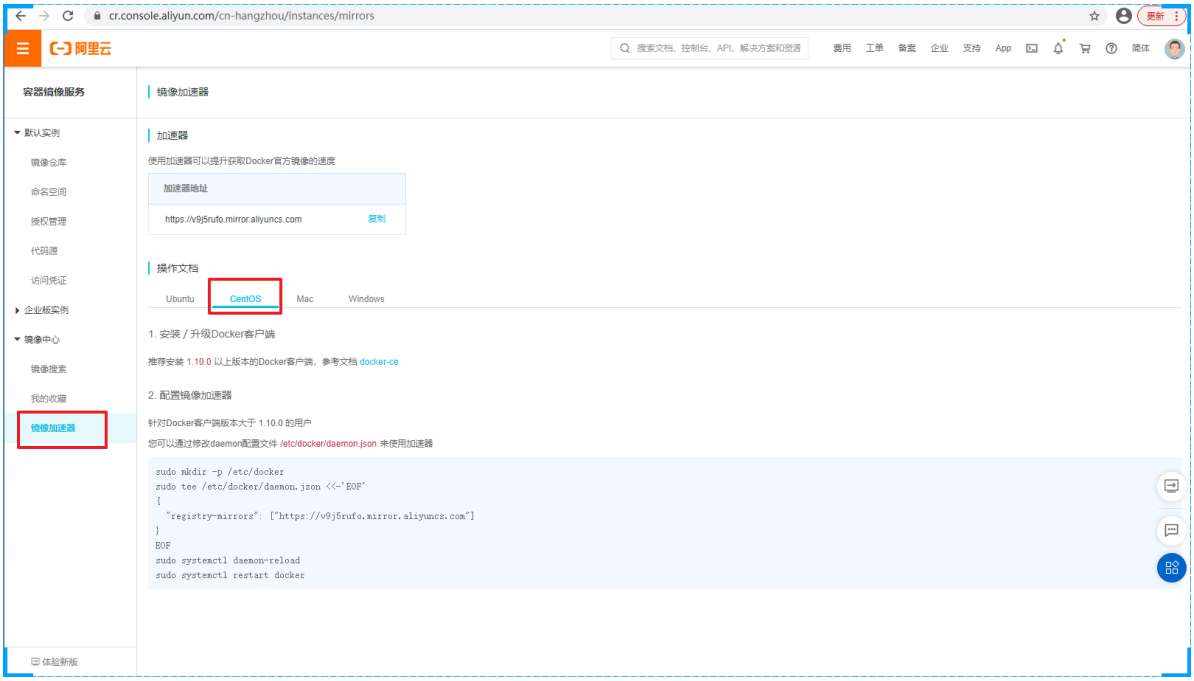
7. 阿里云镜像加速

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







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]#
```

8.Docker卸载

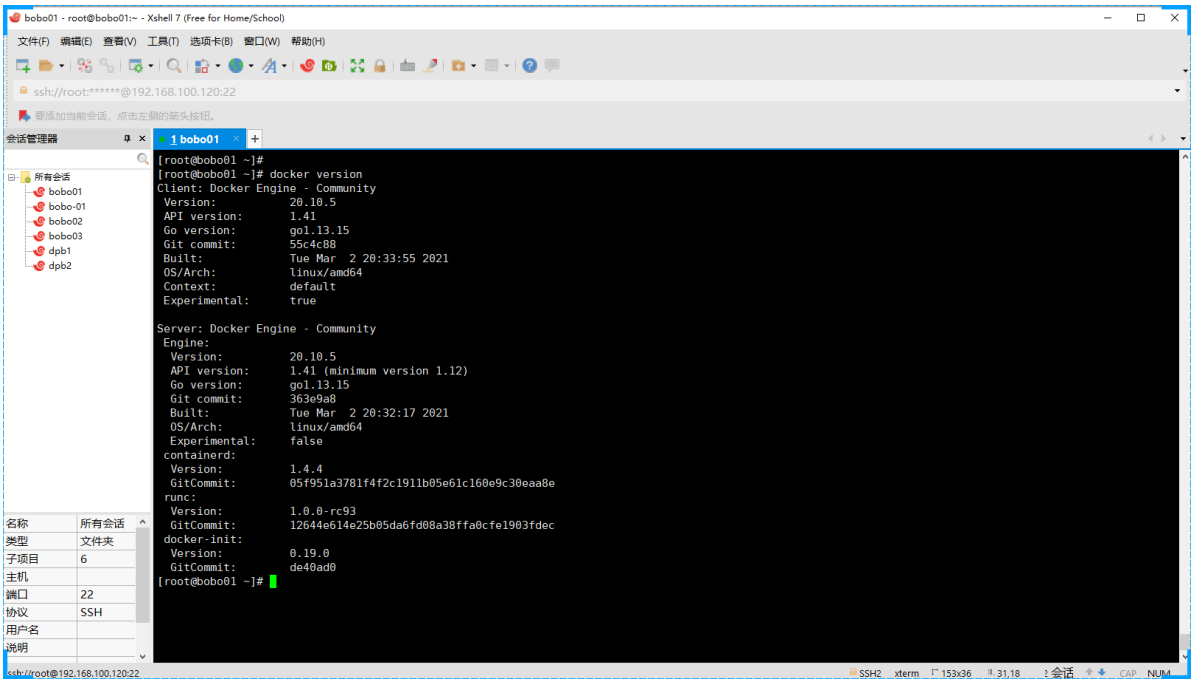
```
systemctl stop docker
yum -y remov docker-ce
rm -rf /var/lib/docker
```

二、Docker常用命令介绍

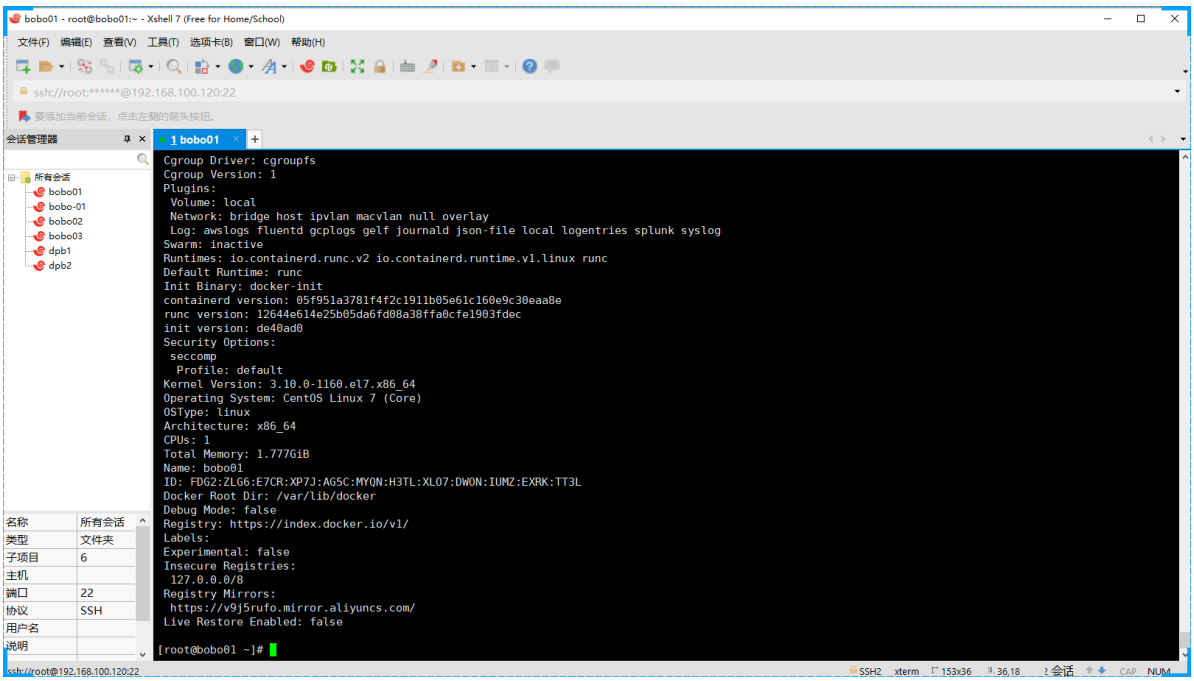
1. 帮助命令

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

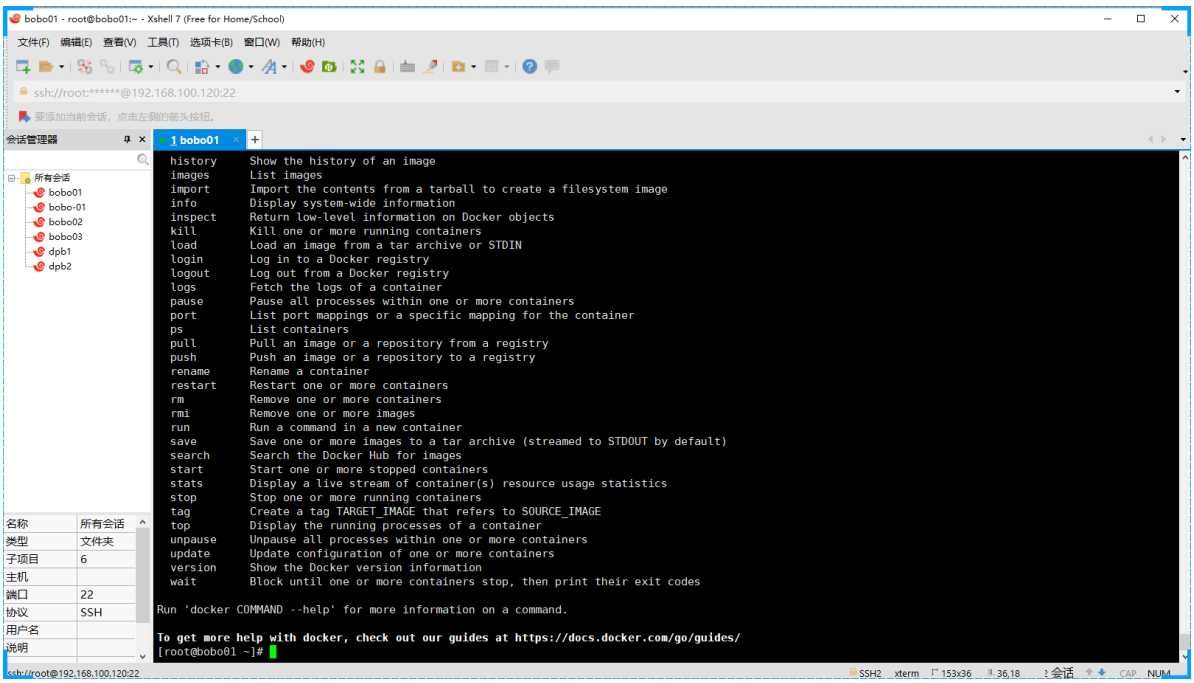
docker version



docker info



docker --help



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

2.1 docker images

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

镜像表格信息说明

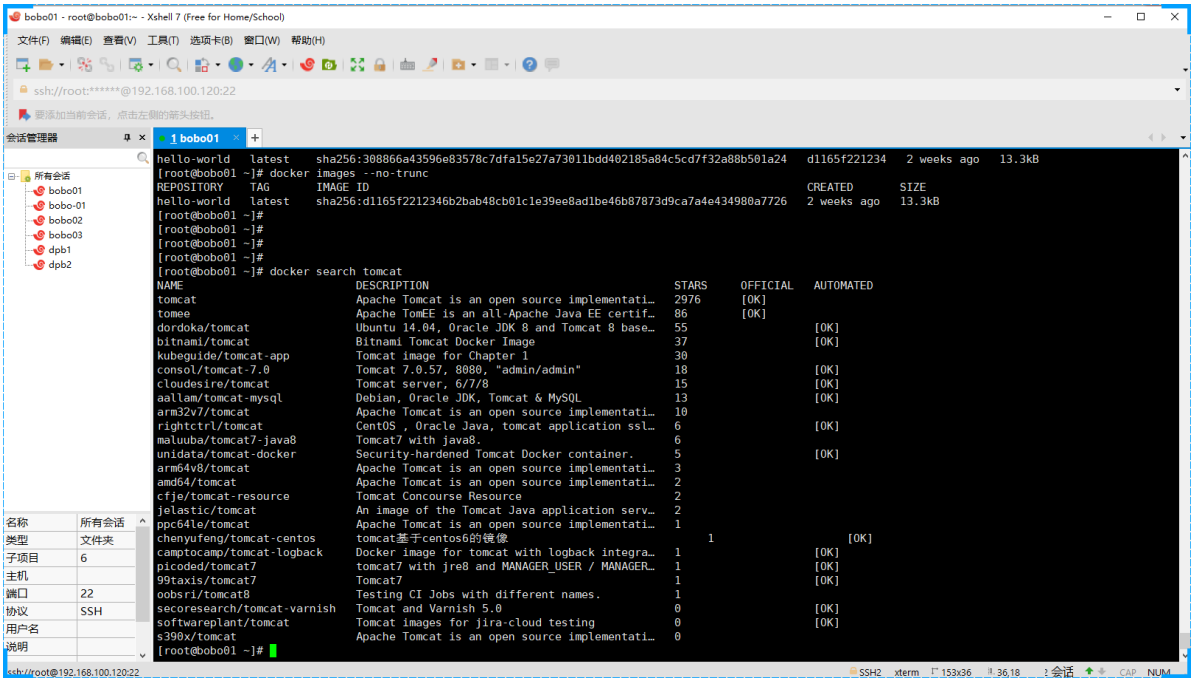
选项	说明
REPOSITORY	表示镜像的仓库源
TAG	镜像的标签
IMAGE ID	镜像ID
CREATED	镜像创建时间
SIZE	镜像大小

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 docker search

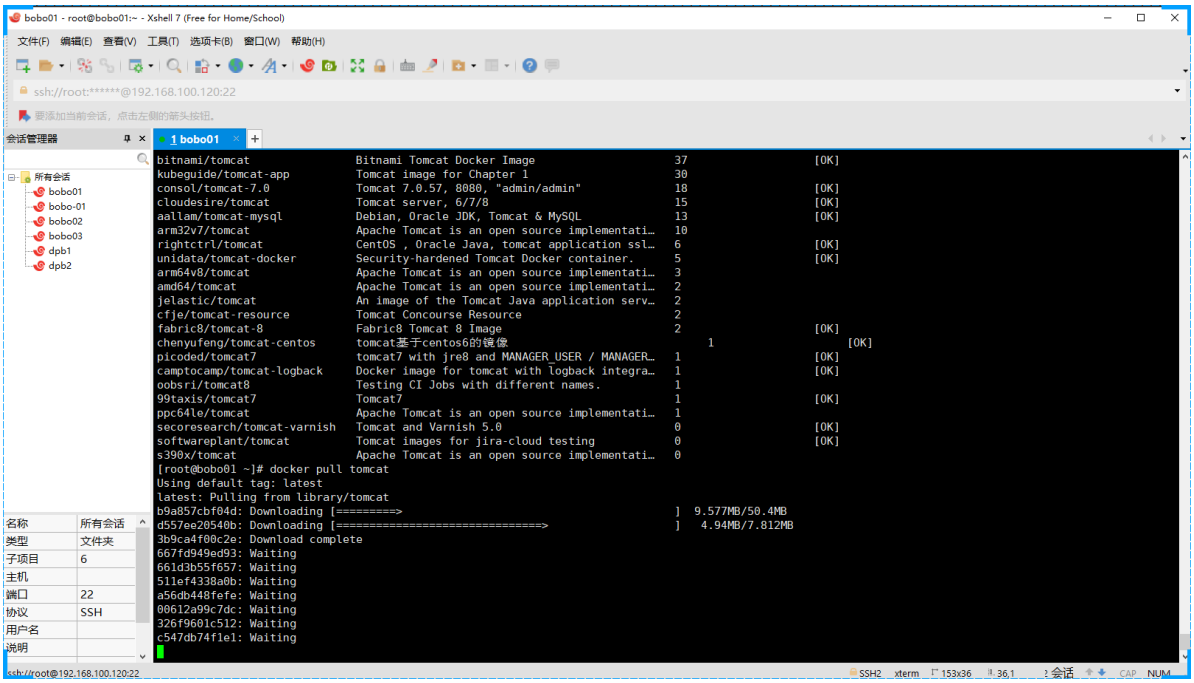
docker hub是Docker的在线仓库，我们可以通过docker search 在上面搜索我们需要的镜像



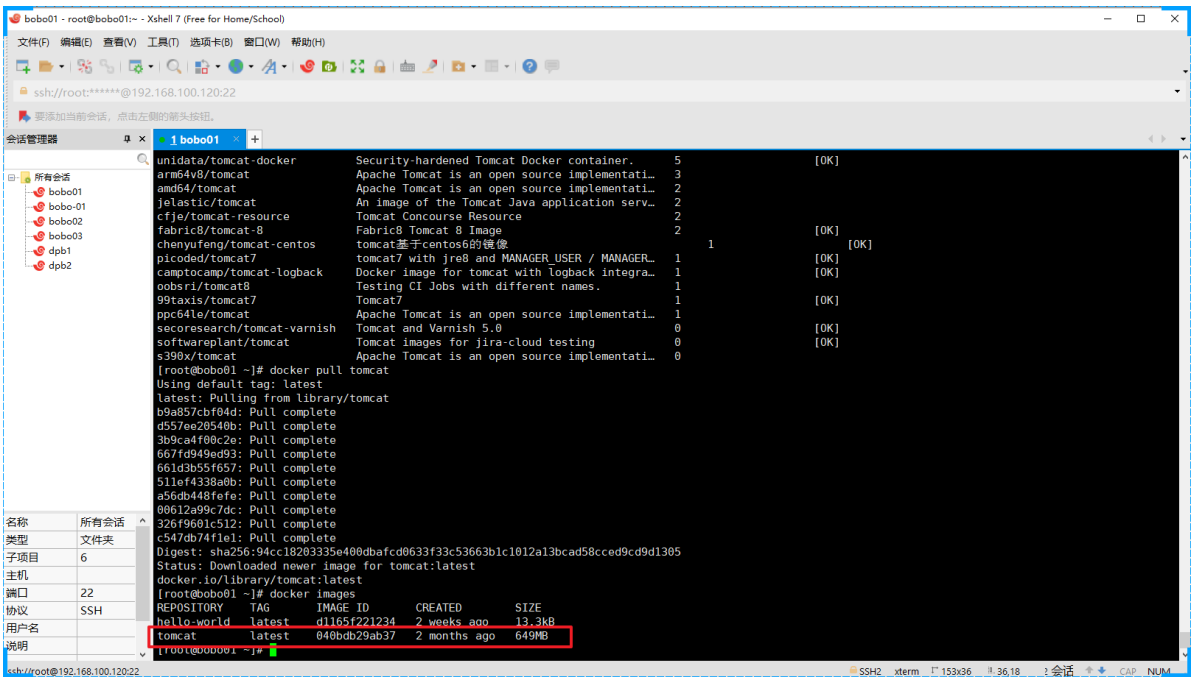
参数名称	描述
--no-trunc	显示完整的描述信息
--limit	分页显示
-f	过滤条件 docker search -f STARS=5 tomcat

2.3 Docker pull

从Docker hub 上下载镜像文件



等待下载完成即可



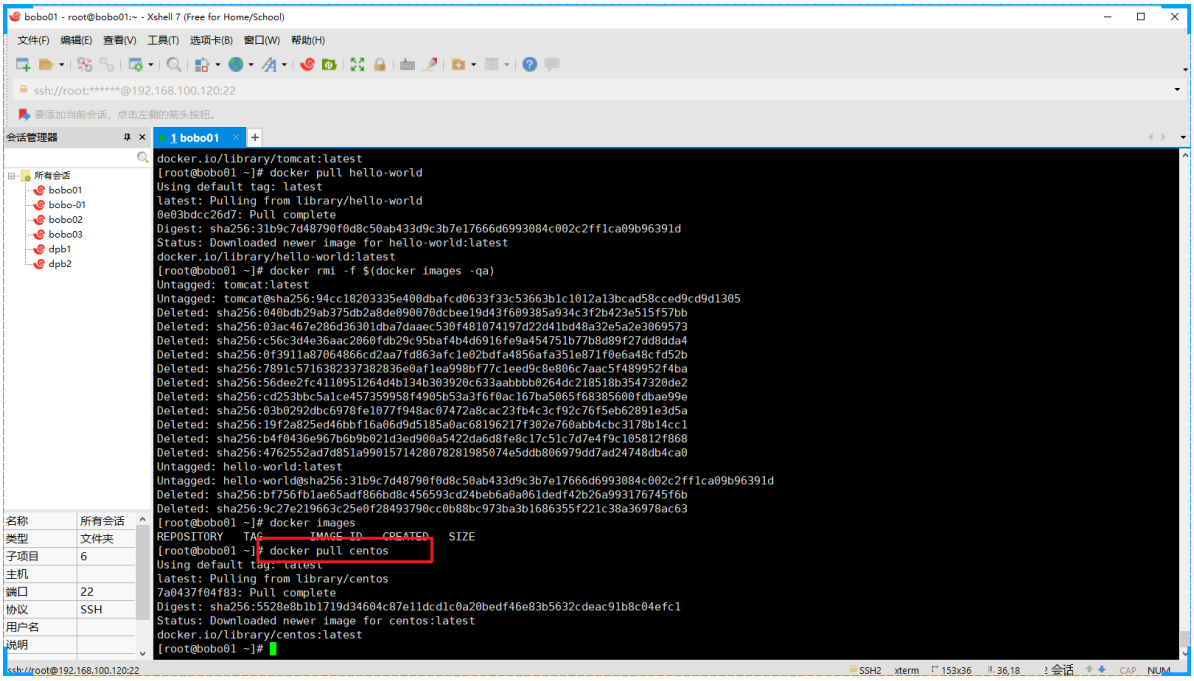
2.4 Docker rmi

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

3.容器命令

有镜像才能创建容器。



3.1 创建并启动容器

创建并启动一个容器的命令

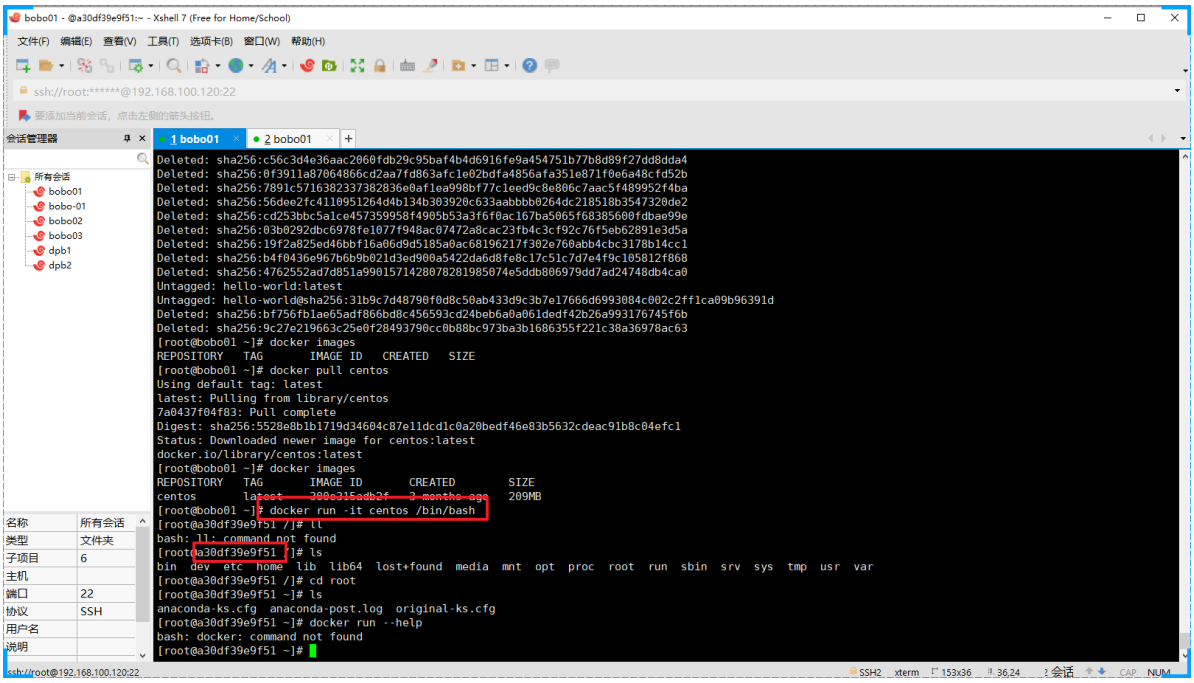
```
docker run [OPTIONS] IMAGE [COMMAND]
```

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
```



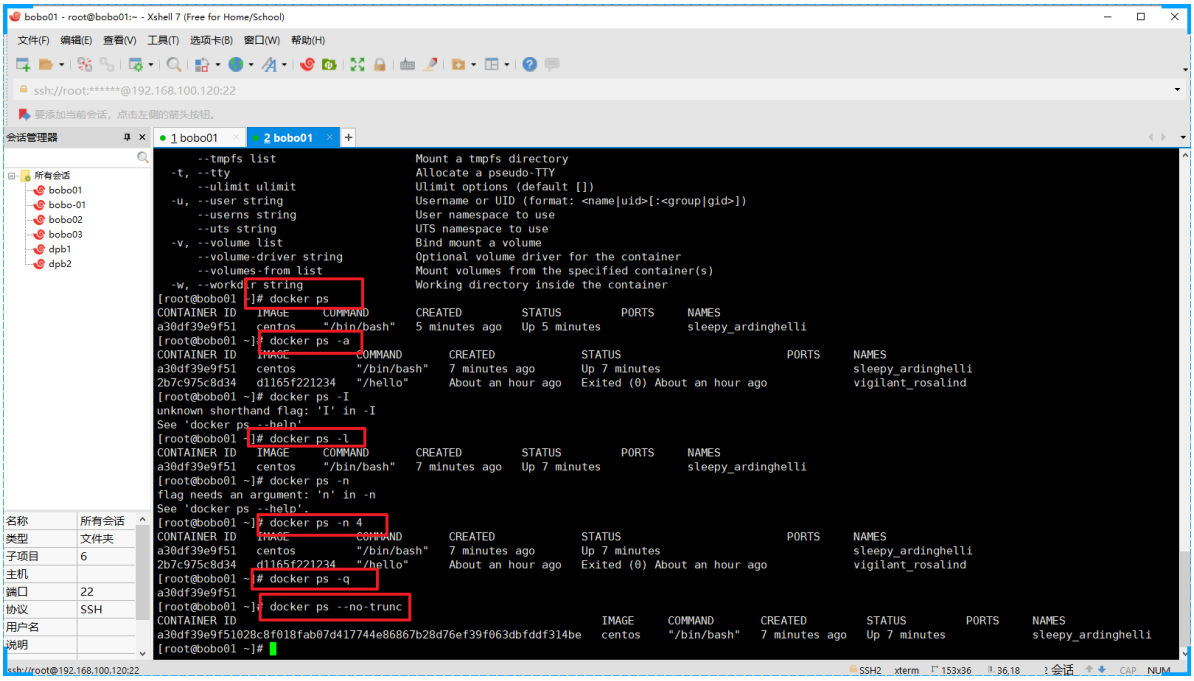
3.2 列举运行的容器

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]
```

OPTIONS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。



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

3.4 启动容器

```
docker start 容器ID或者容器名称
```

3.5 重启容器

```
docker restart 容器id或者名称
```

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
```

3.7 删除容器

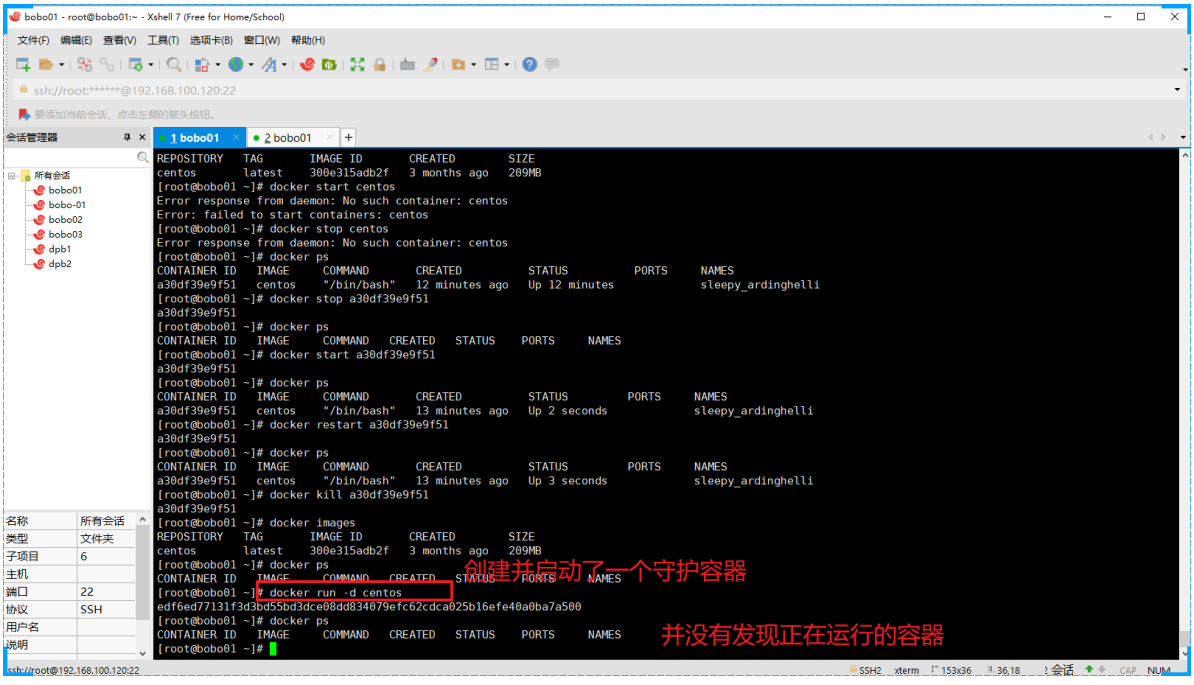
有时候容器使用完成就没有作用了，我们想要删除掉容器，这时可以通过rm命令

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

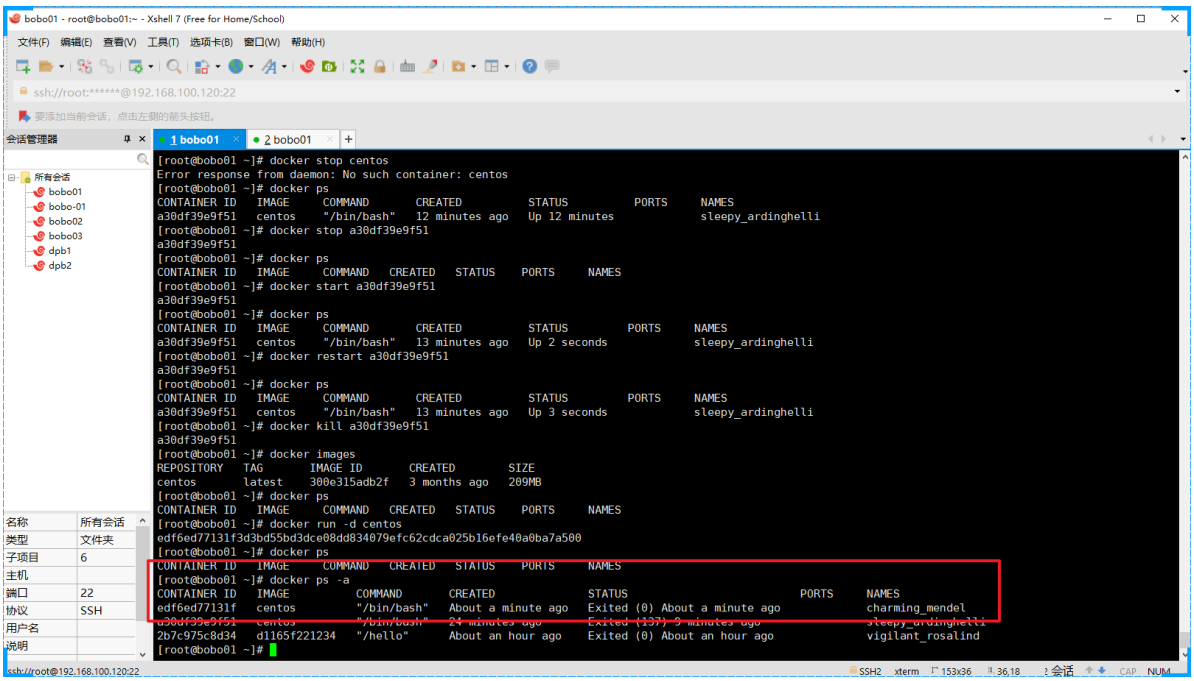
4. 其他命令

4.1 守护式容器

```
docker run -d 容器名称
```

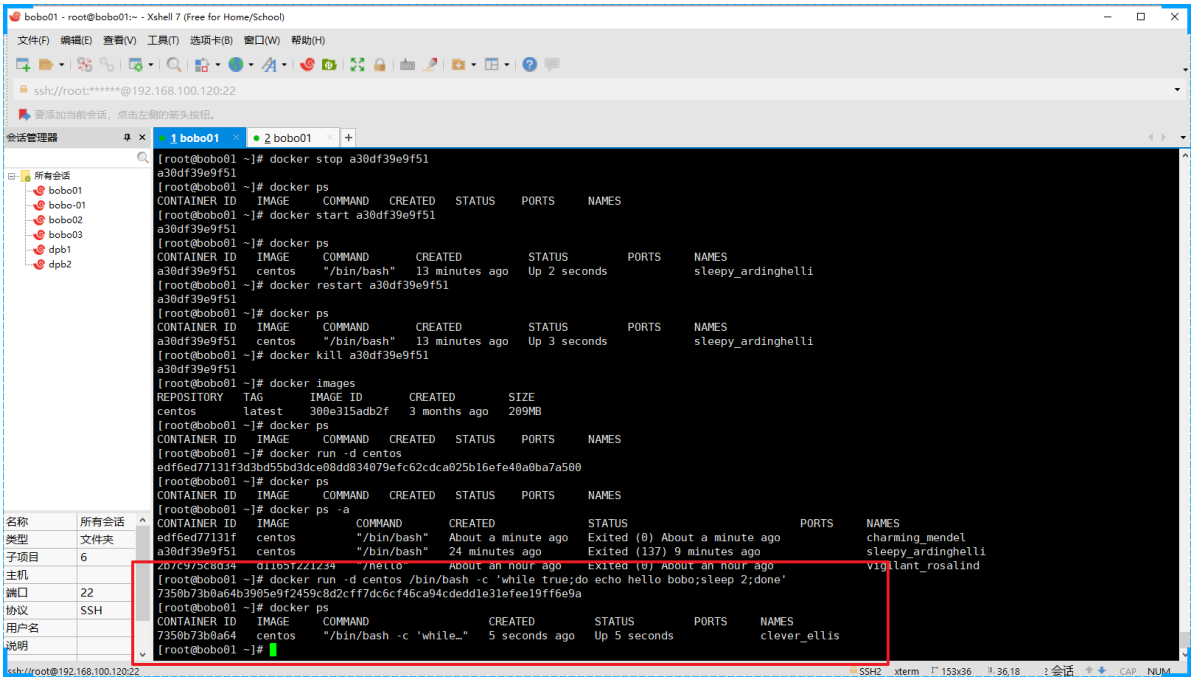


我们通过 docker ps -a 可以看到刚刚启动的容器已经退出了



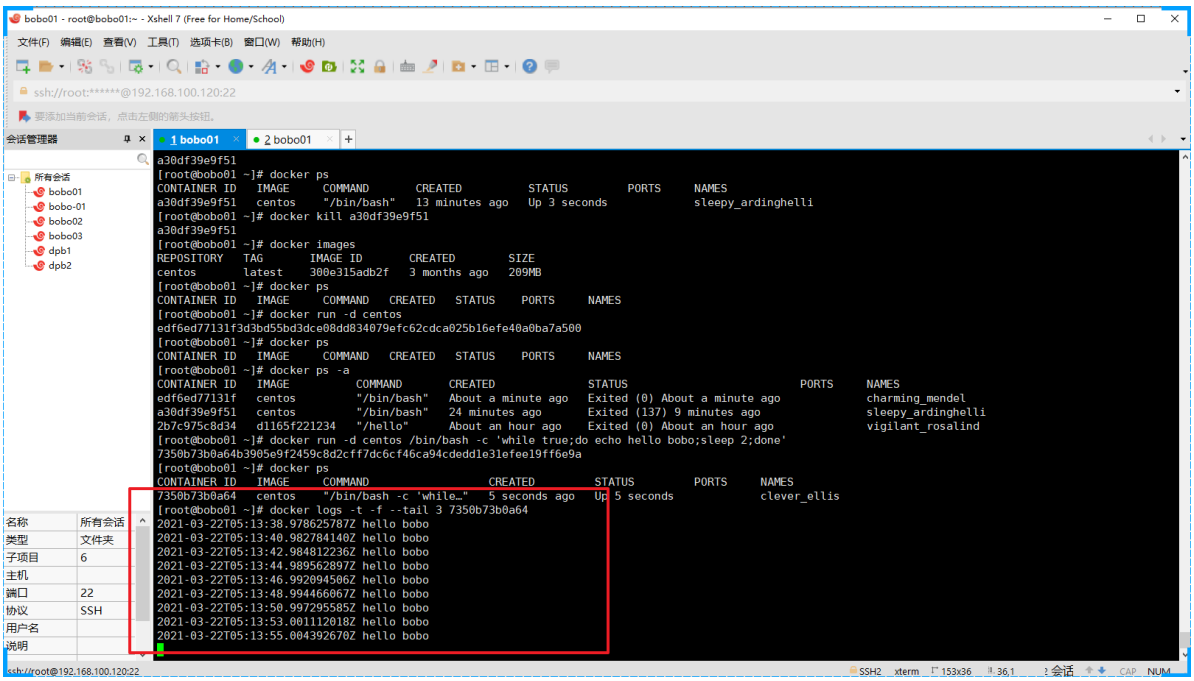
为了让守护式容器能够一直执行，我们可以在启动容器后在后台运行一个循环的脚本

```
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
```



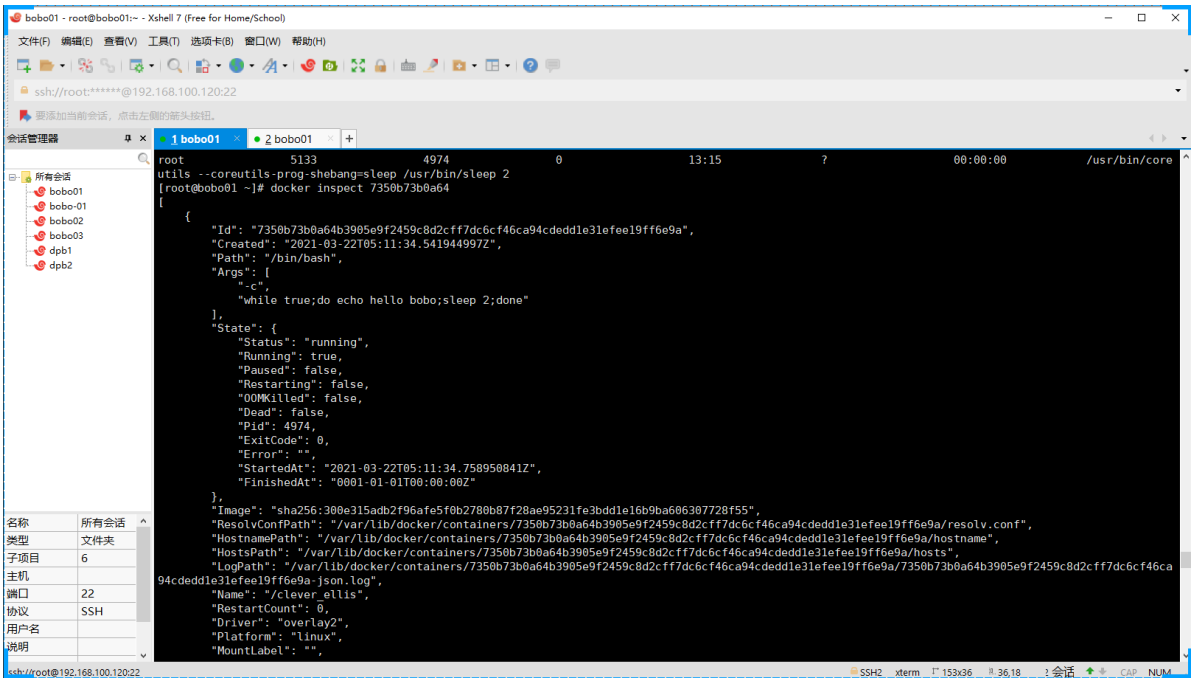
查看容器中运行的进程

```
docker top 容器ID
```

4.2 查看容器细节

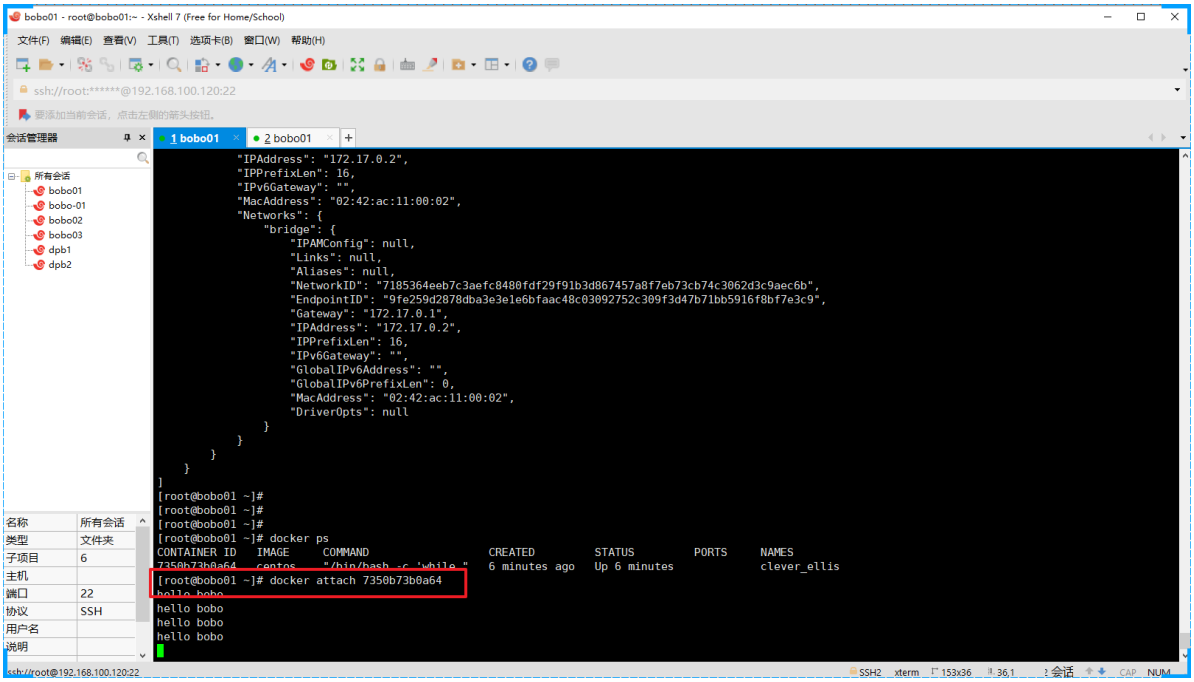
我们想要查看容器的细节可以通过inspect命令

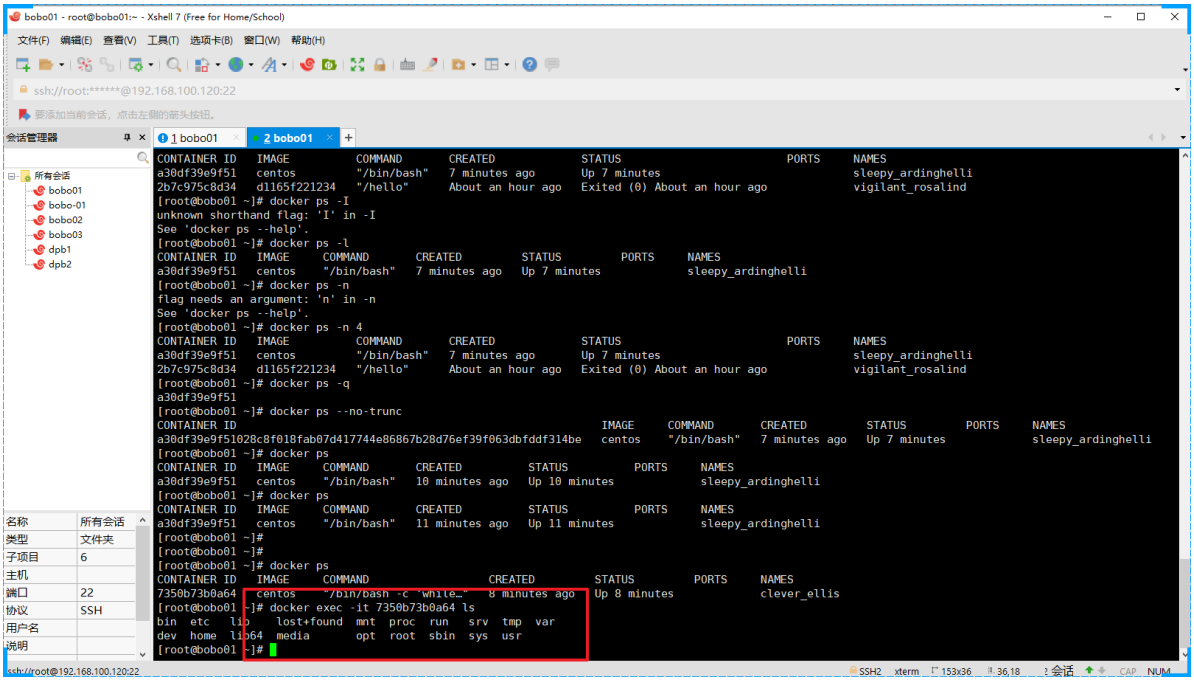
```
docker inspect 容器ID
```



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

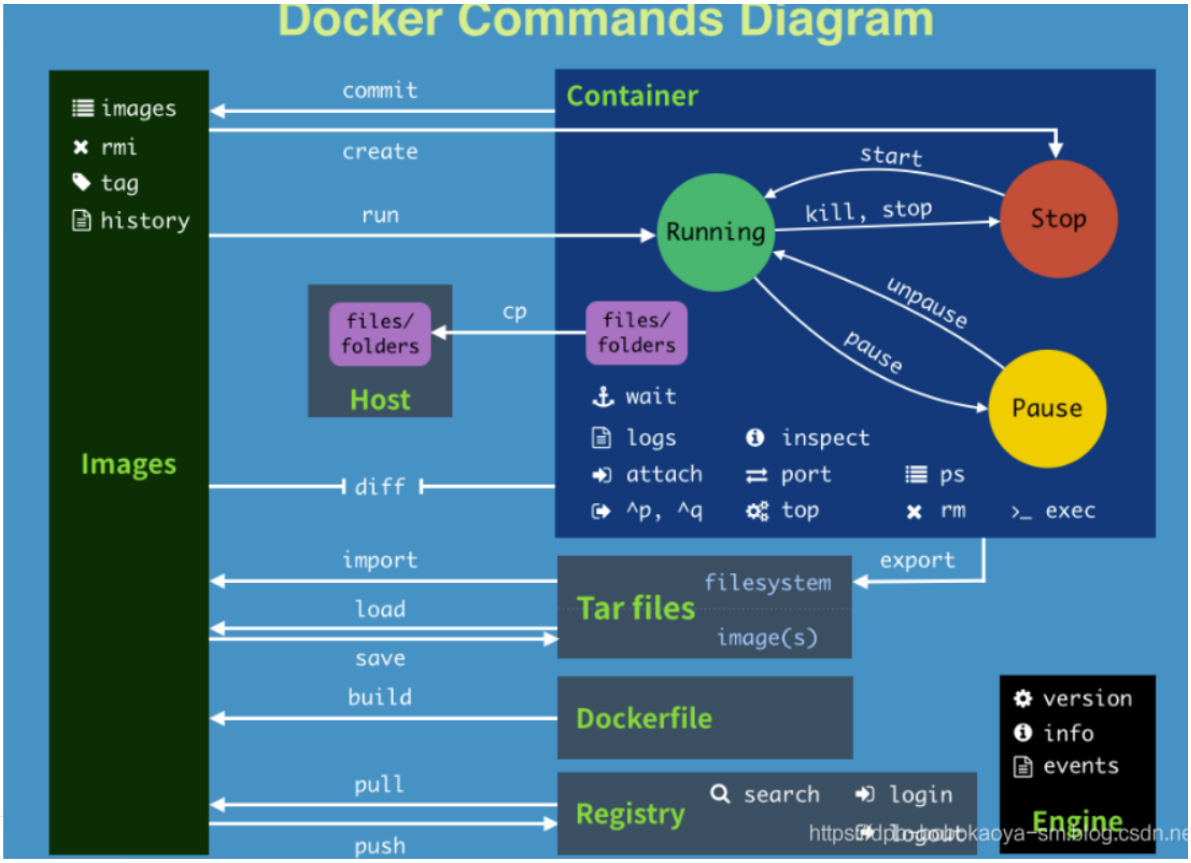




4.4 文件复制

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
```



三、Docker镜像介绍

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。

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

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。

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。

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。

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.镜像的操作

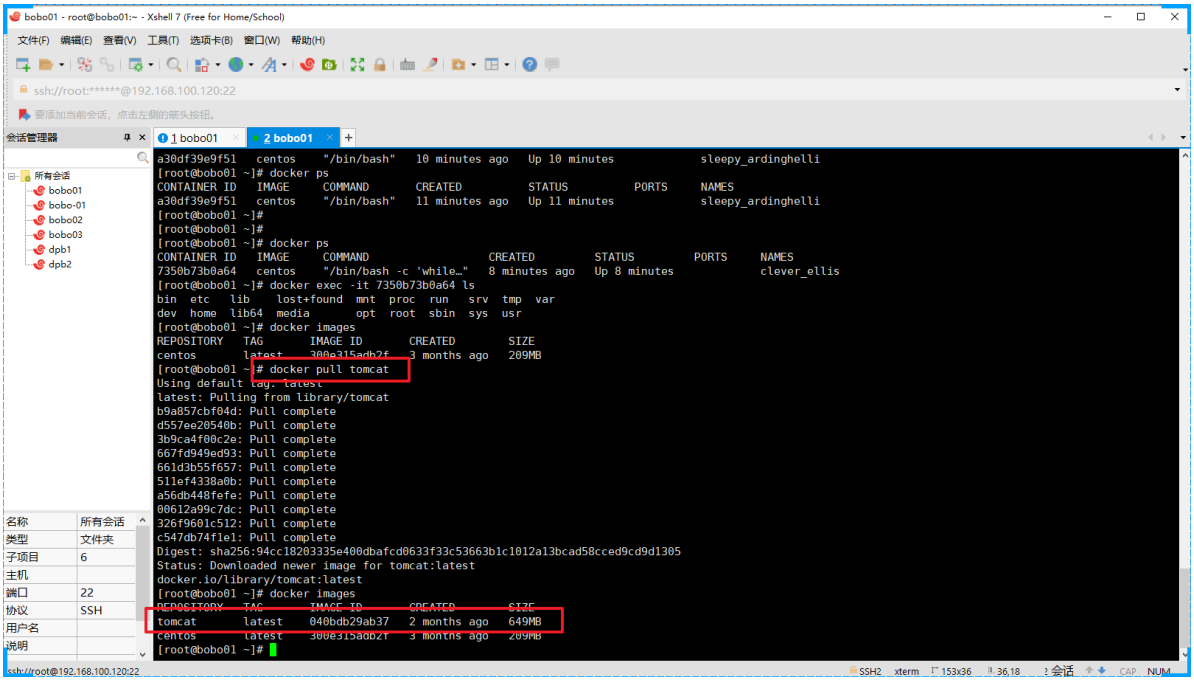
我们现在已经掌握了从docker hub上获取相关镜像，然后运行容器，并作出我们自己的处理，但有时候我们需要将我们自己的容器制作为对应的镜像，以便后面继续使用，这时我们就需要用到docker commit ...命令了，这节我们就通过案例来介绍下 docker commit ...命令的使用

```
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。

3.1 下载tomcat镜像

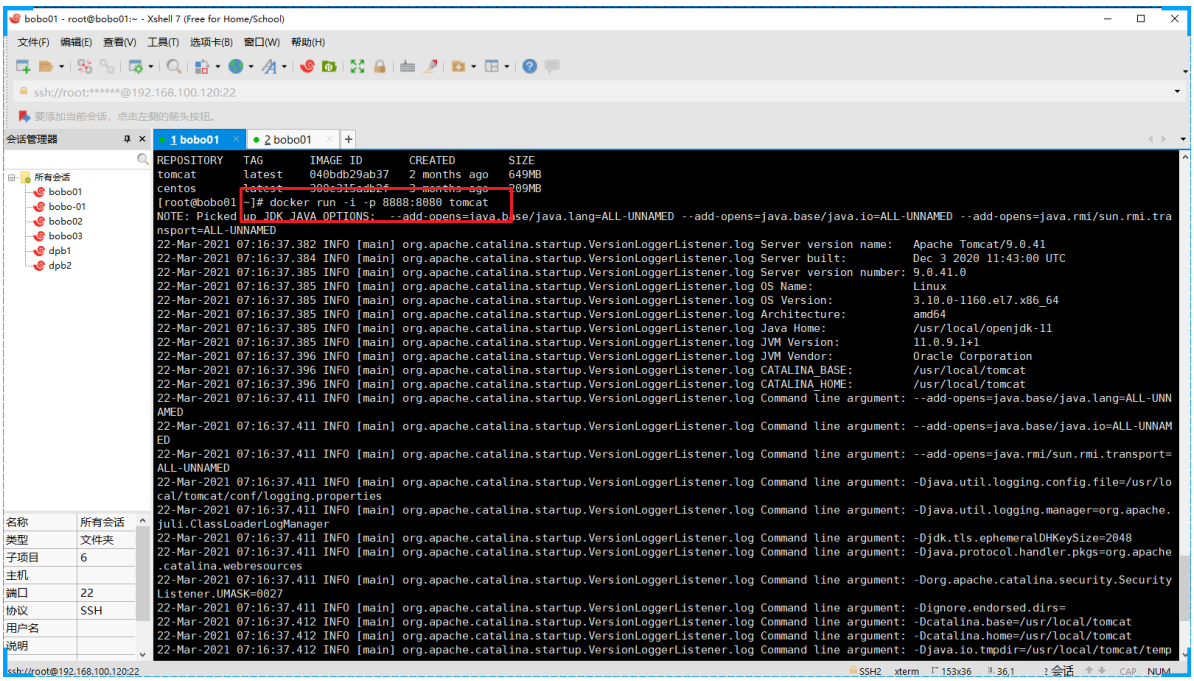


3.2 创建容器并启动

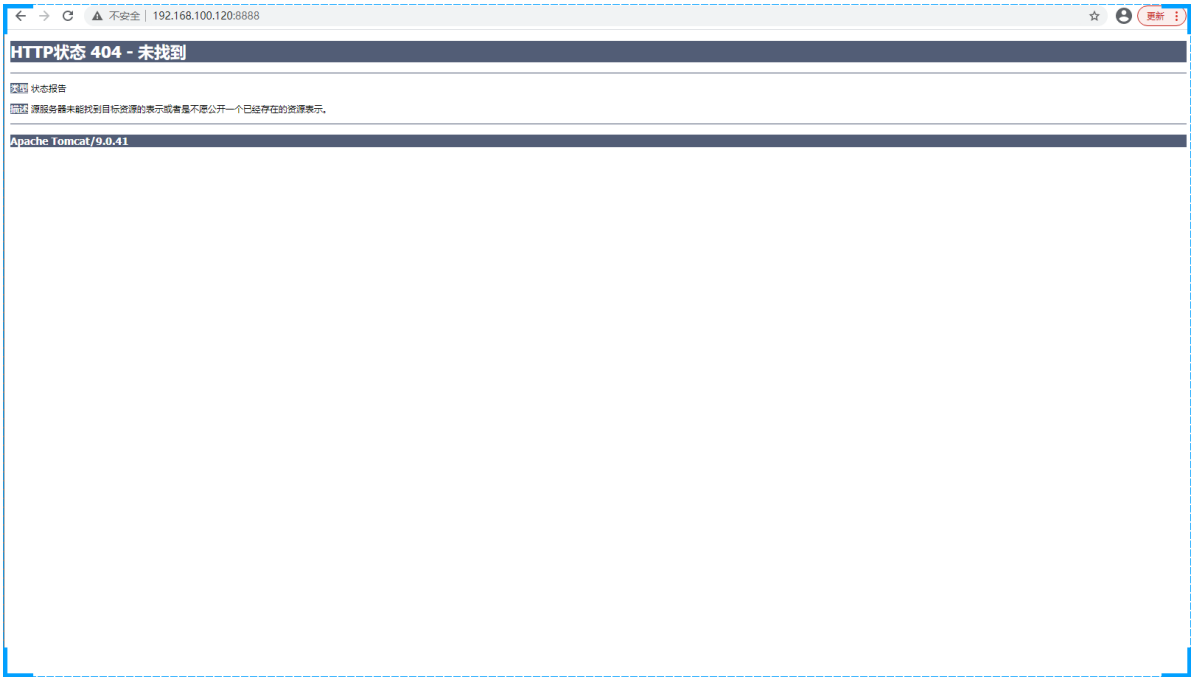
```
docker run -it -p 8888:8080 tomcat
```

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



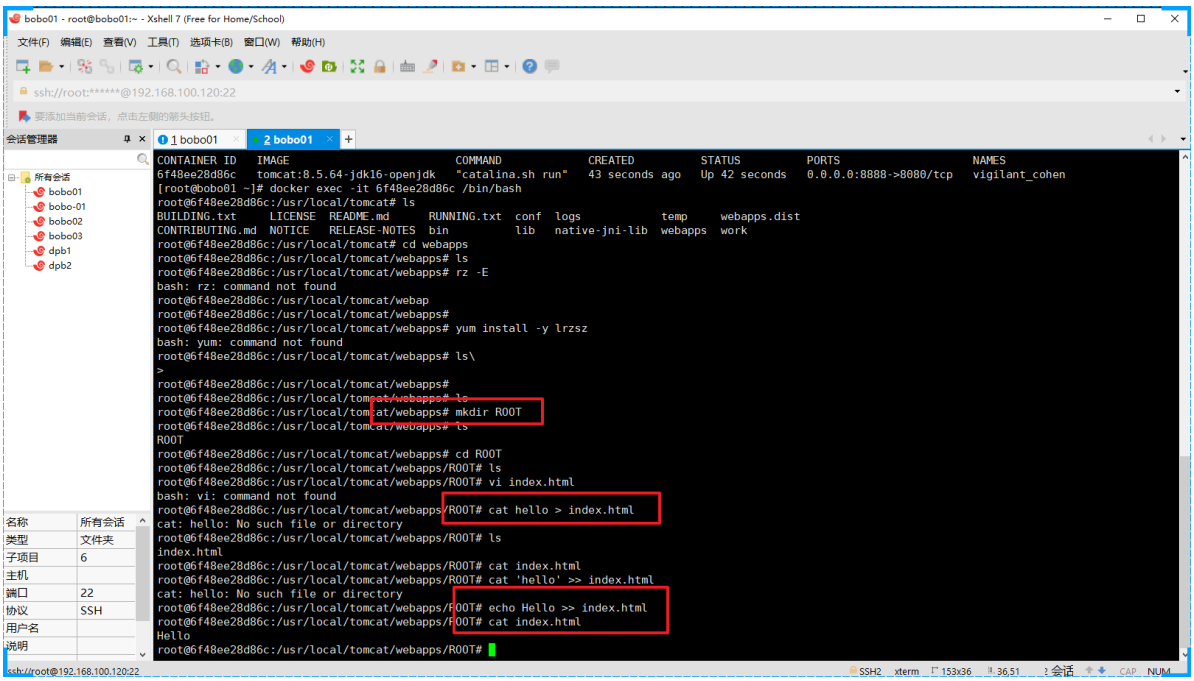
404

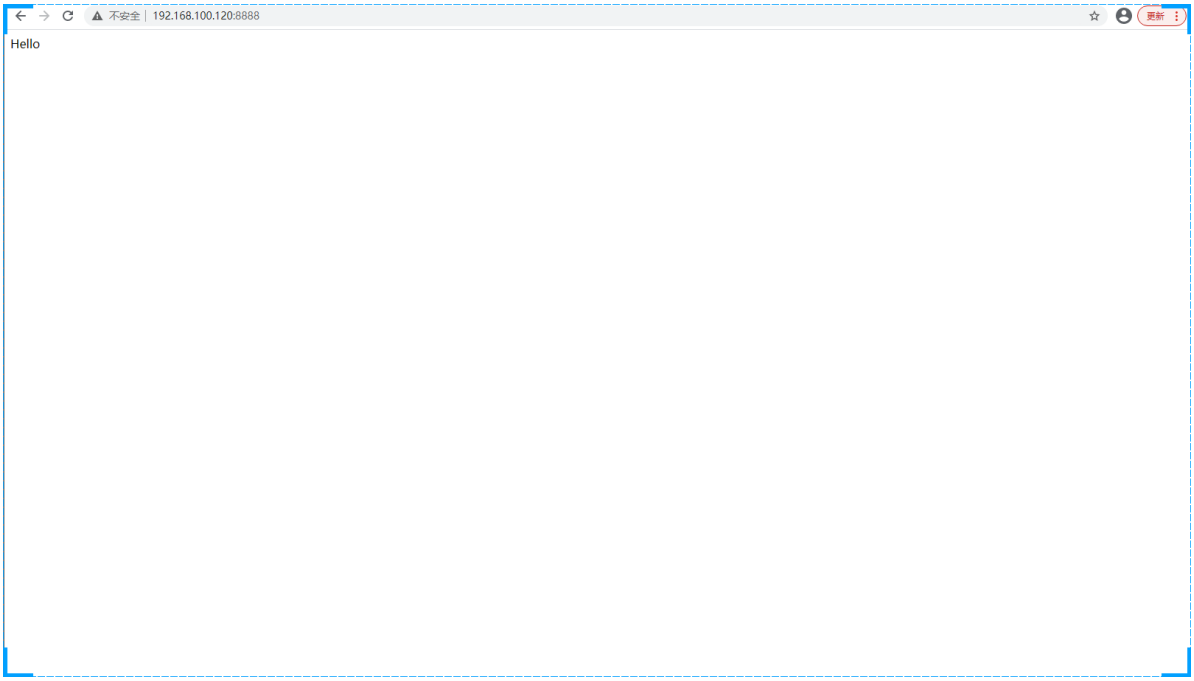


3.3 修改容器

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
```

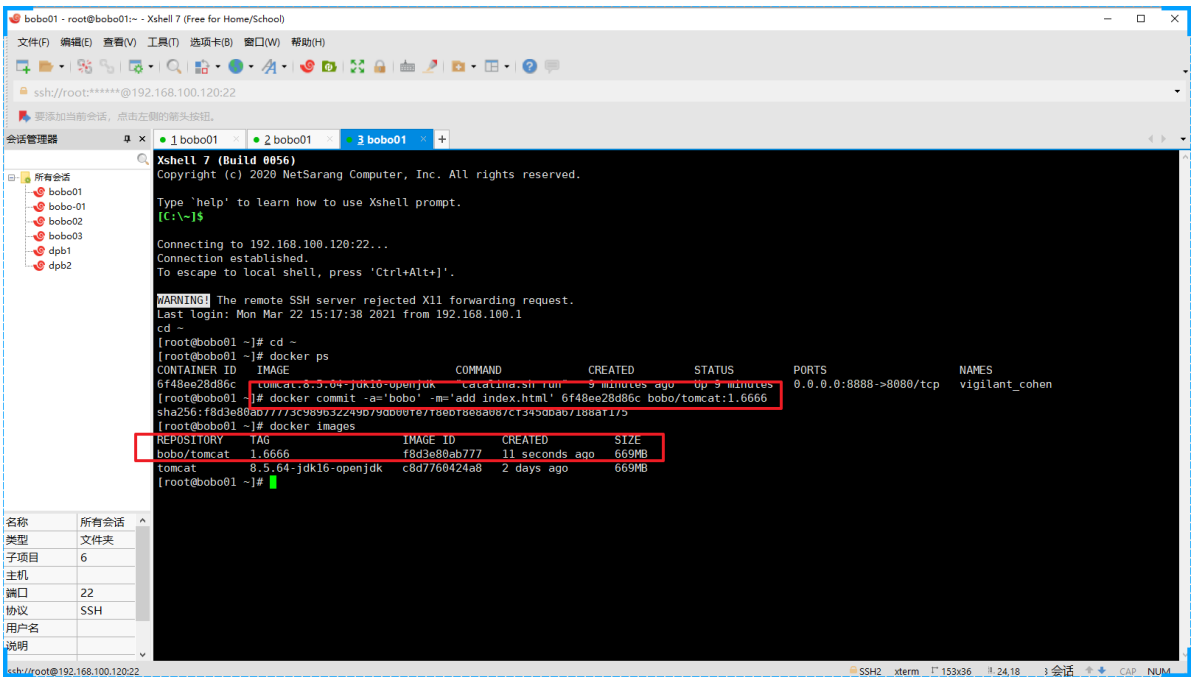




3.4 创建镜像

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

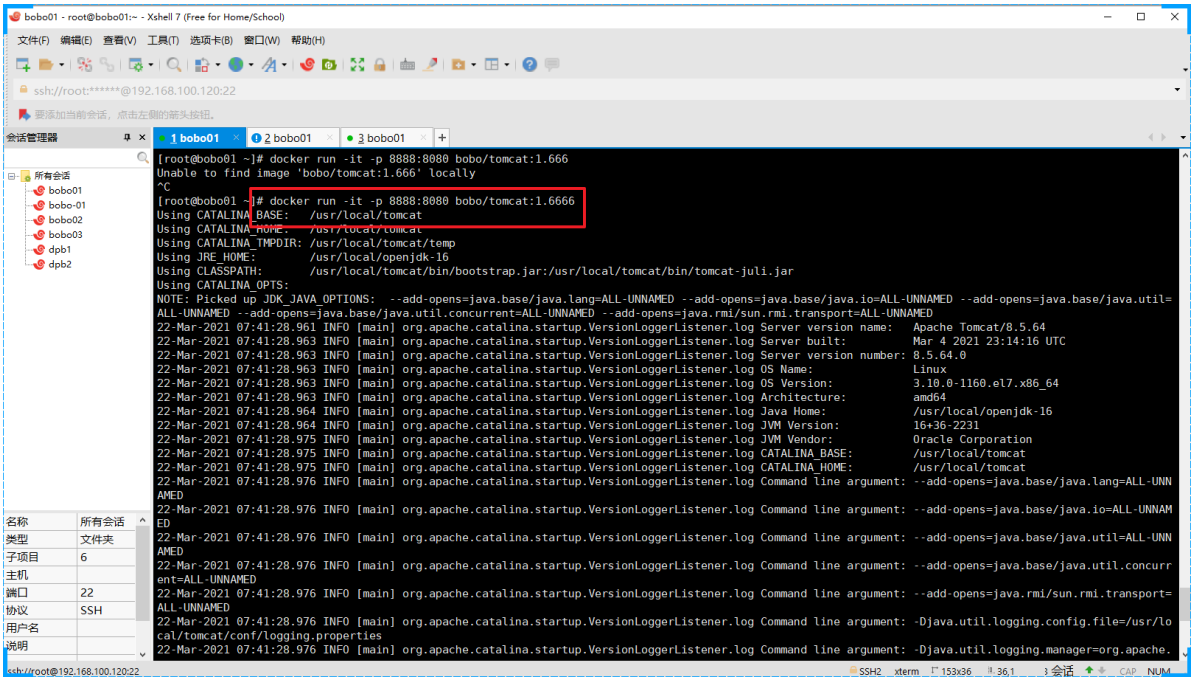
```
docker commit -a='bobo' -m='add index.html' 容器ID bobo/tomcat:1.666
```

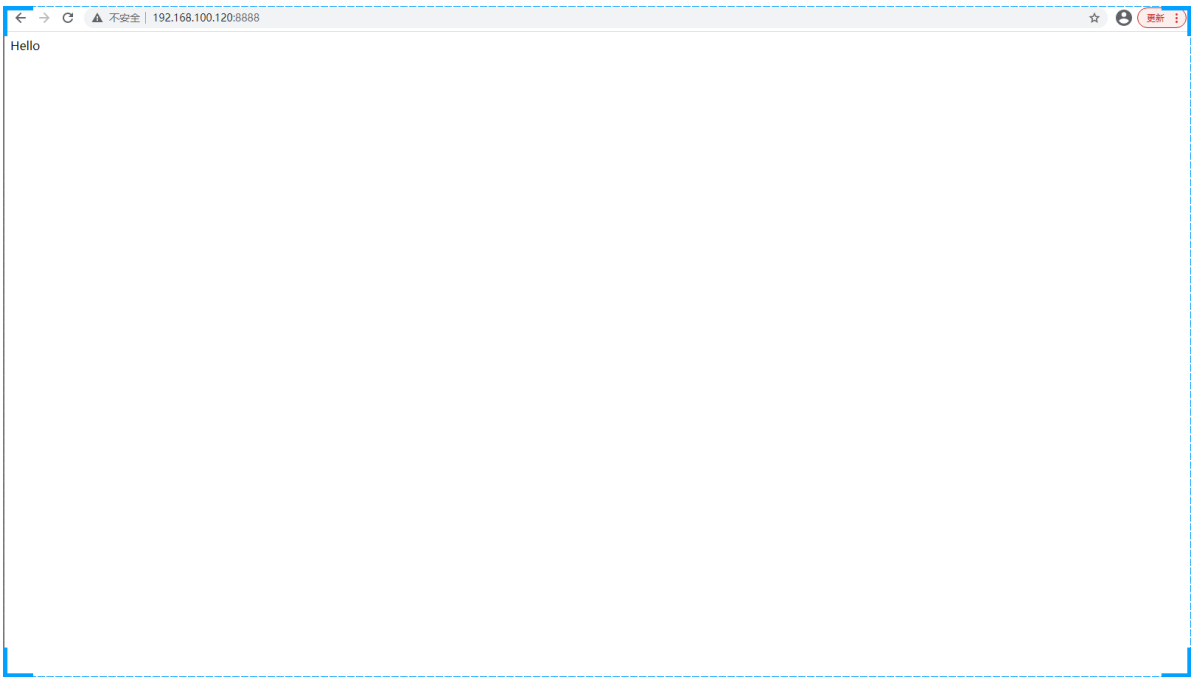


3.5 启动新的镜像

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
```





四、Docker数据卷

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

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。

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

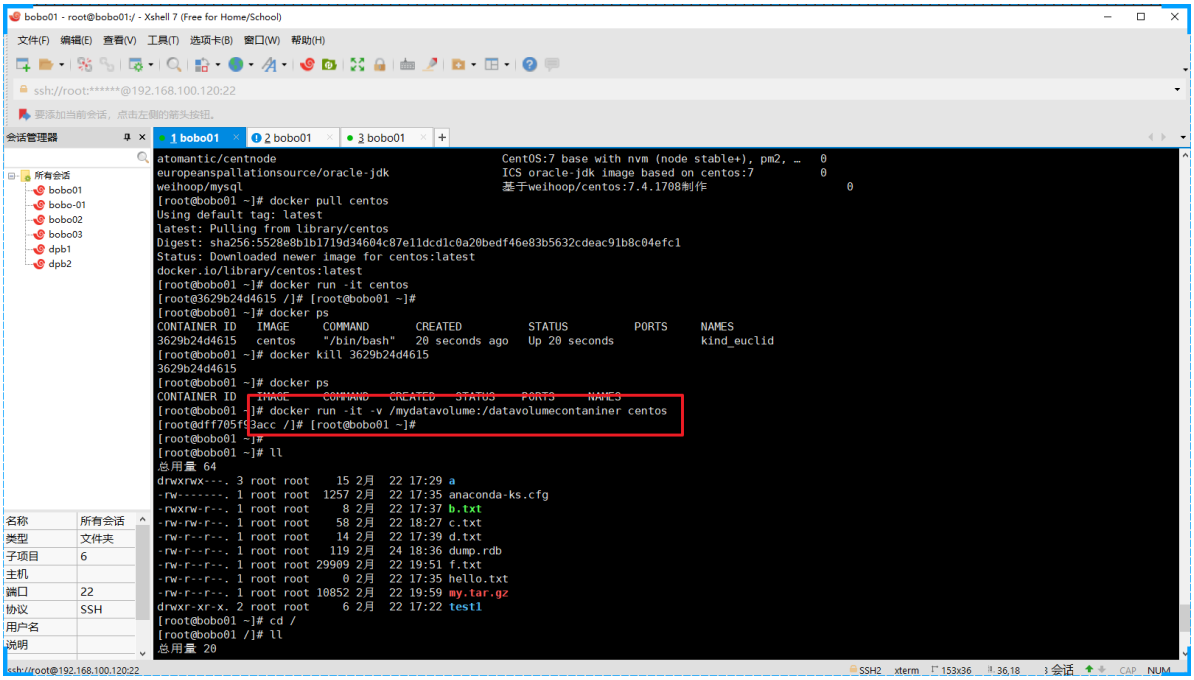
> 持久化，容器间继承和共享数据

1.3 数据卷使用

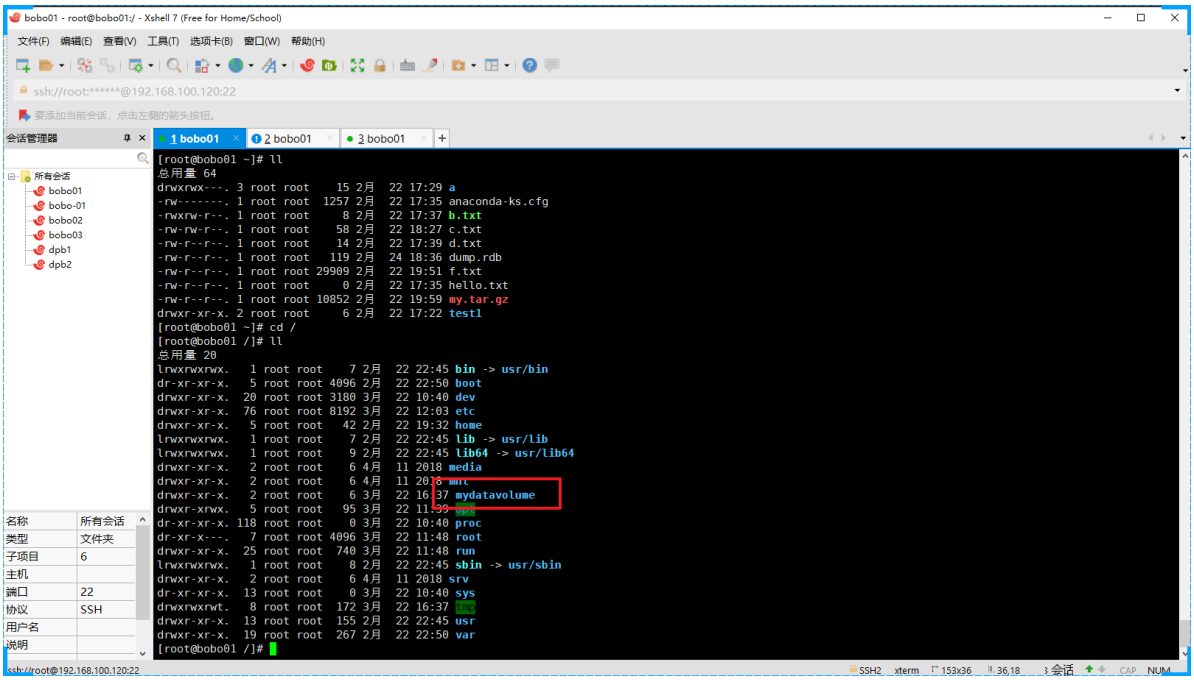
1.3.1 直接添加

运行一个centos容器

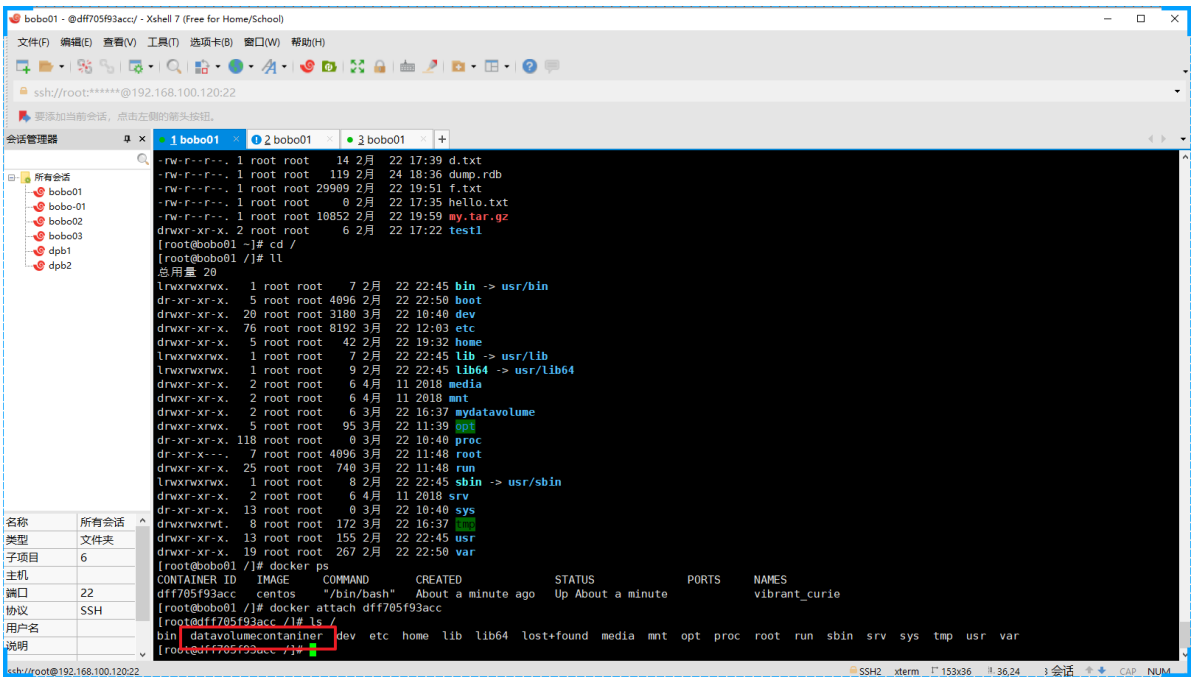
```
docker run -it -v /宿主机绝对路径:/容器内目录 镜像名
```



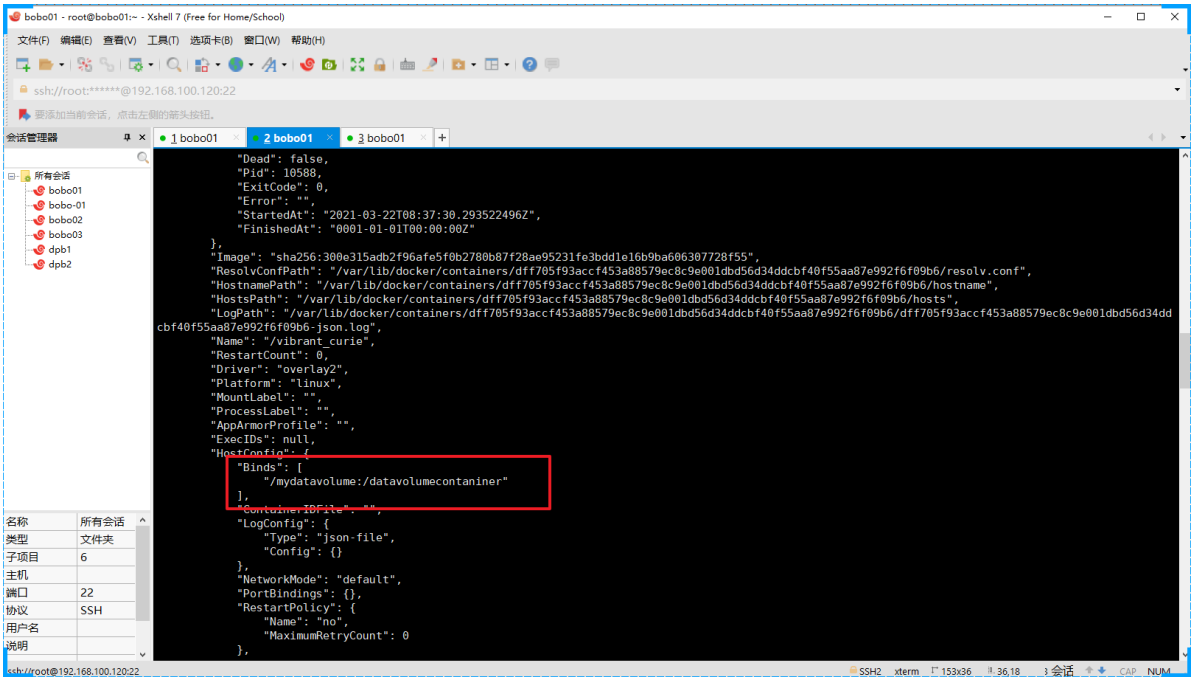
在宿主机的根目录下会多出对应的文件夹



然后在容器的根目录下也会出现对应的文件夹

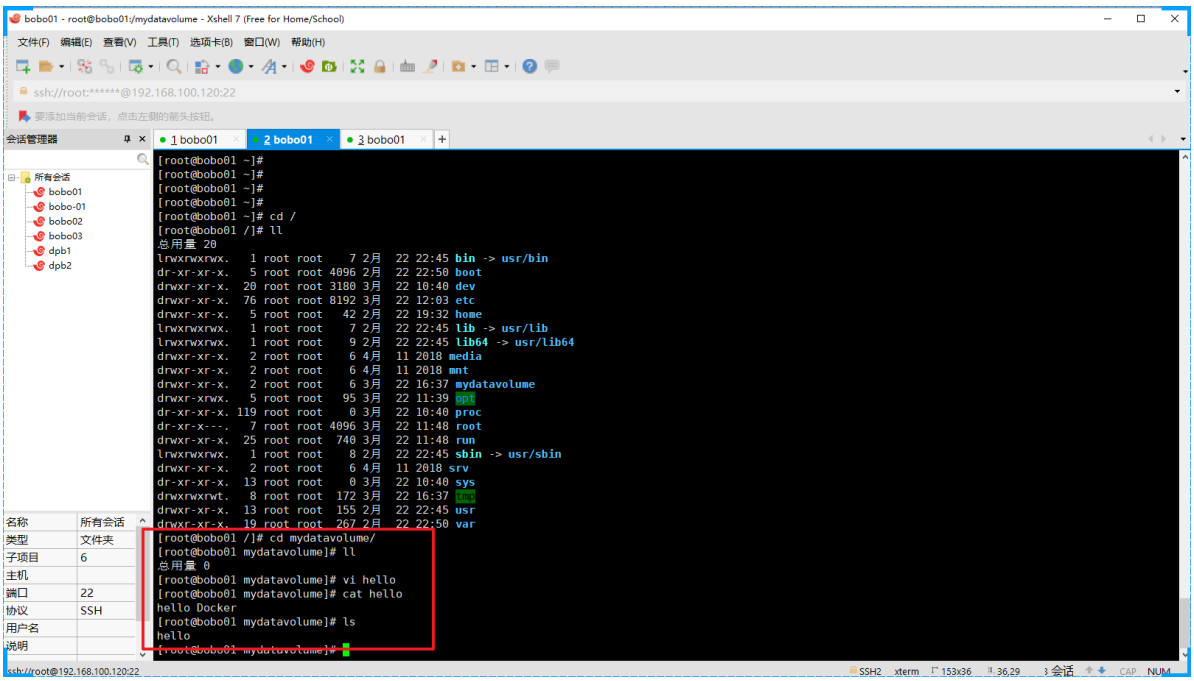


通过inspect命令可以查询容器的详情

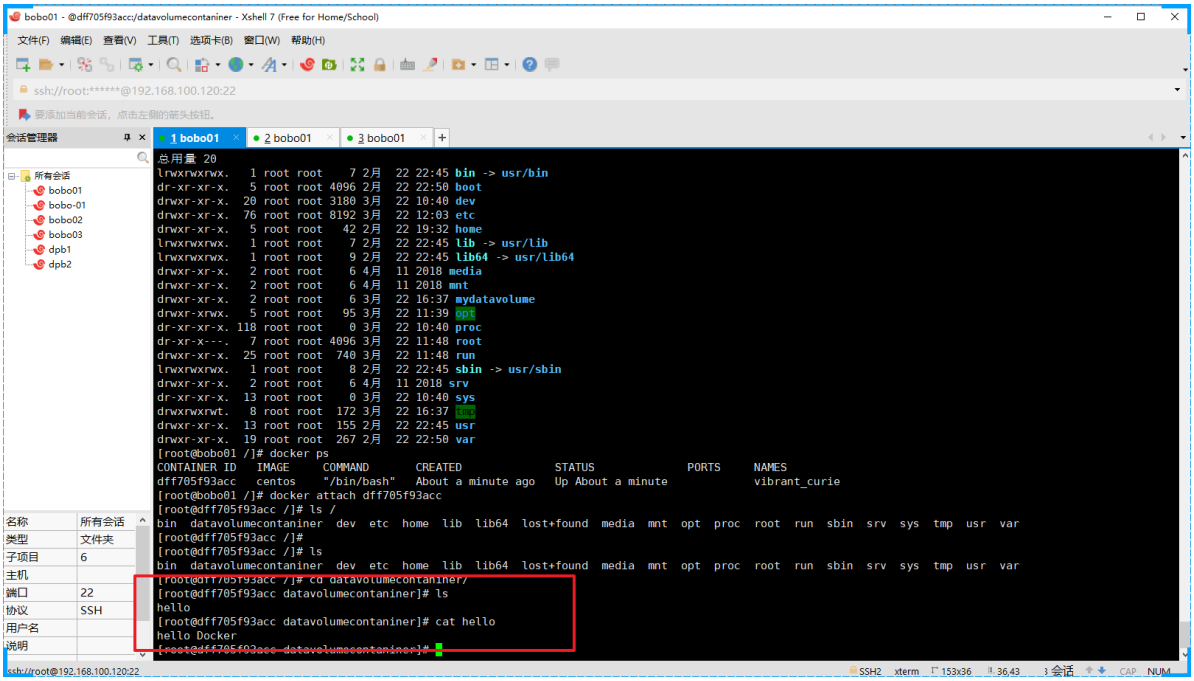


数据共享的操作

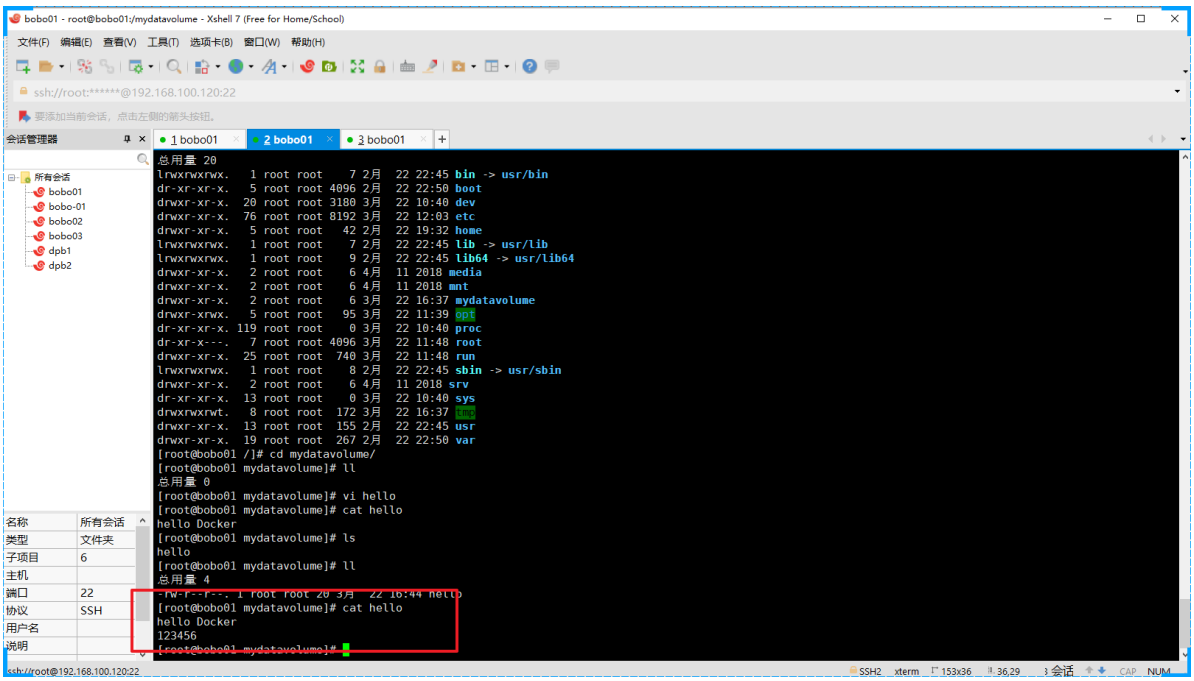
宿主机添加对应的文件



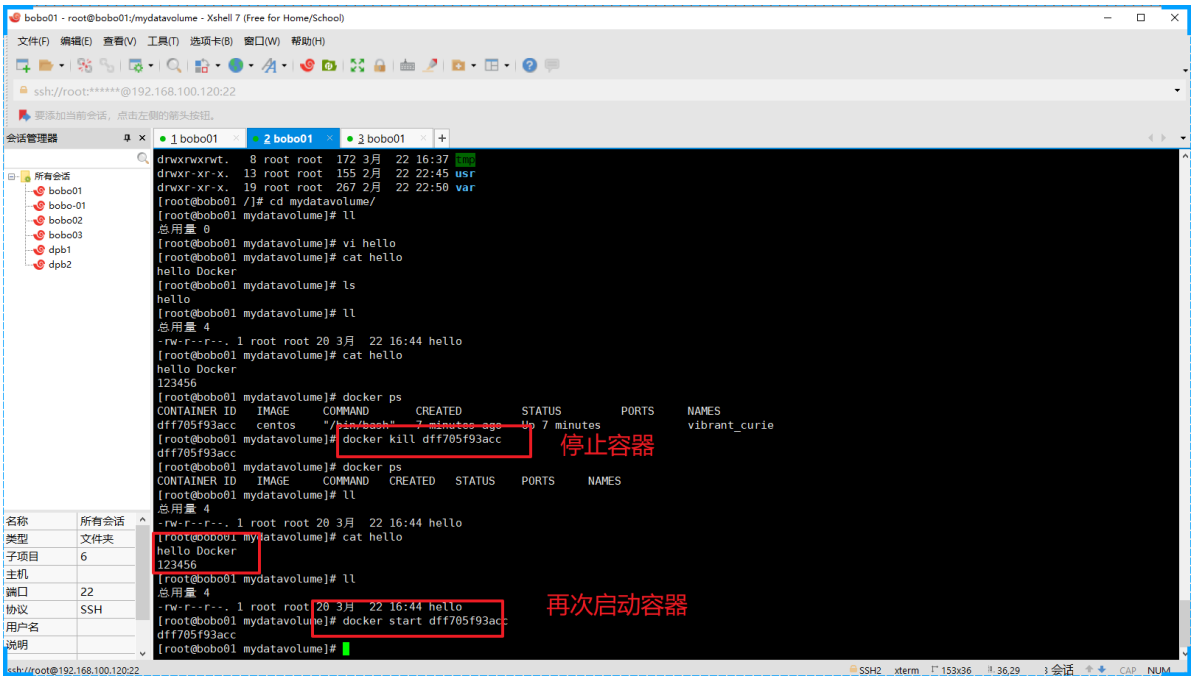
容器中查看

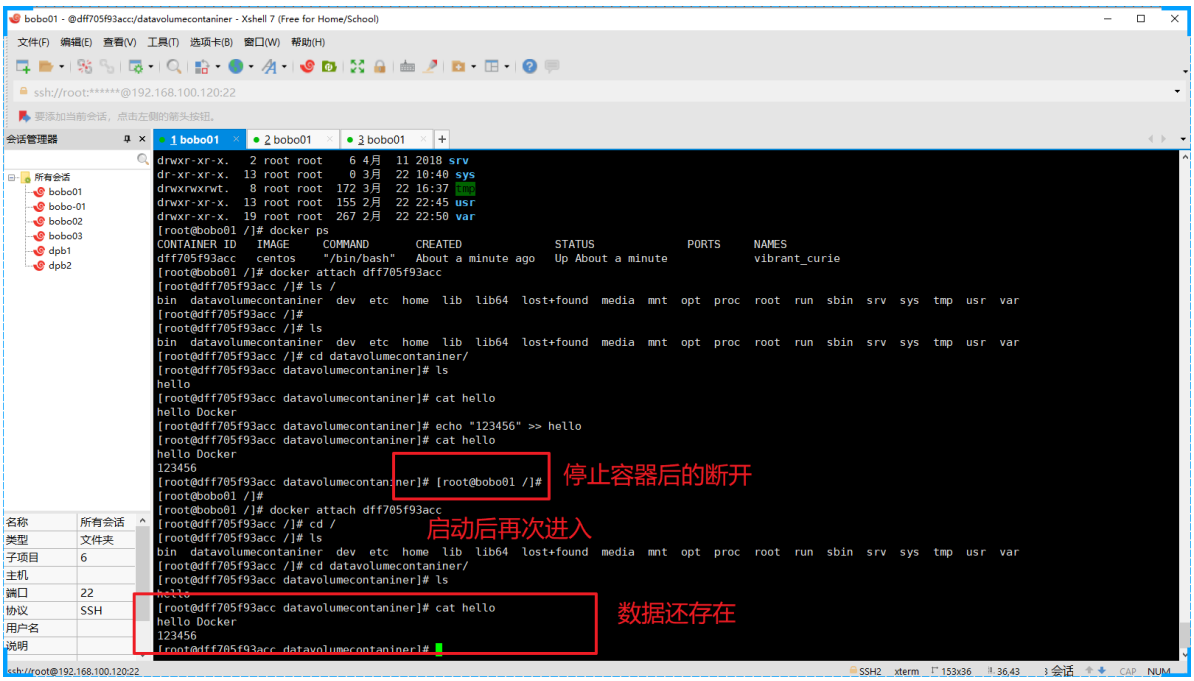


容器中可以同步看到，然后在容器中修改数据

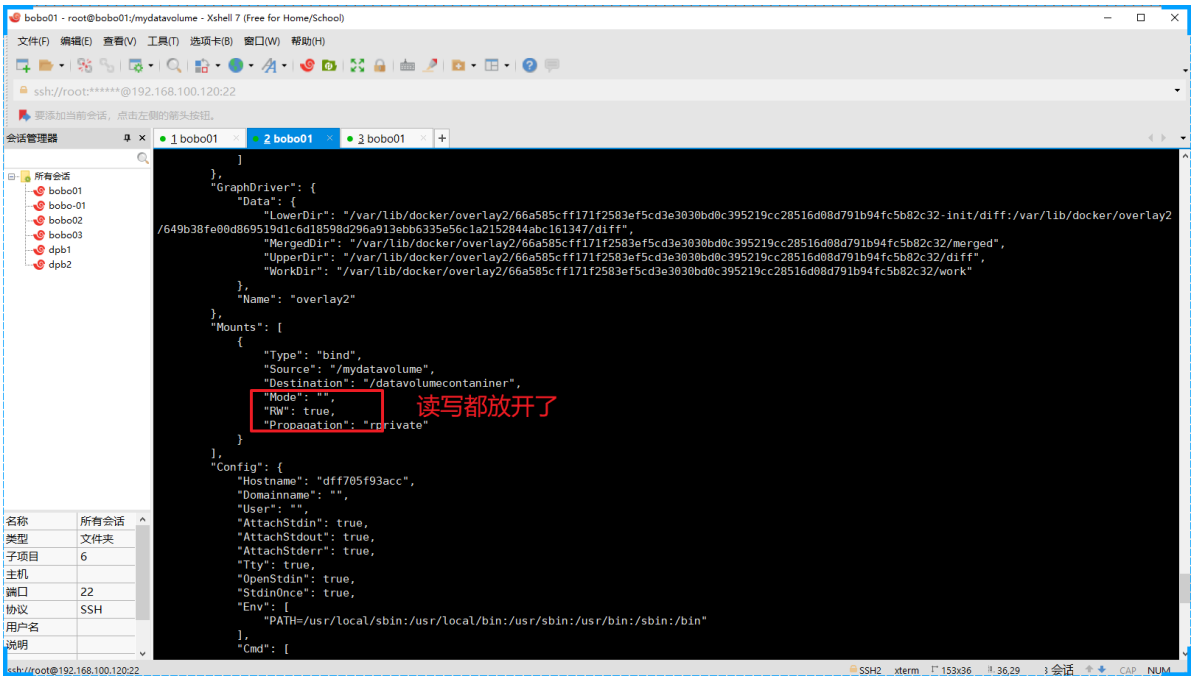


停止掉容器后，数据依然存在



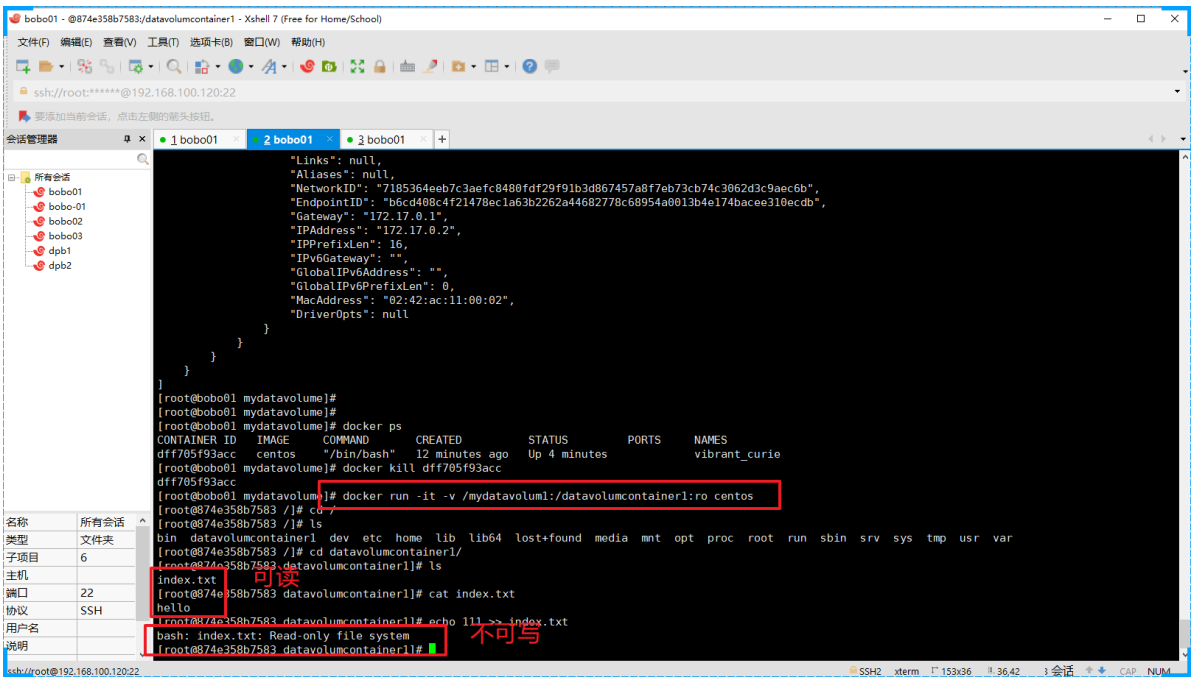


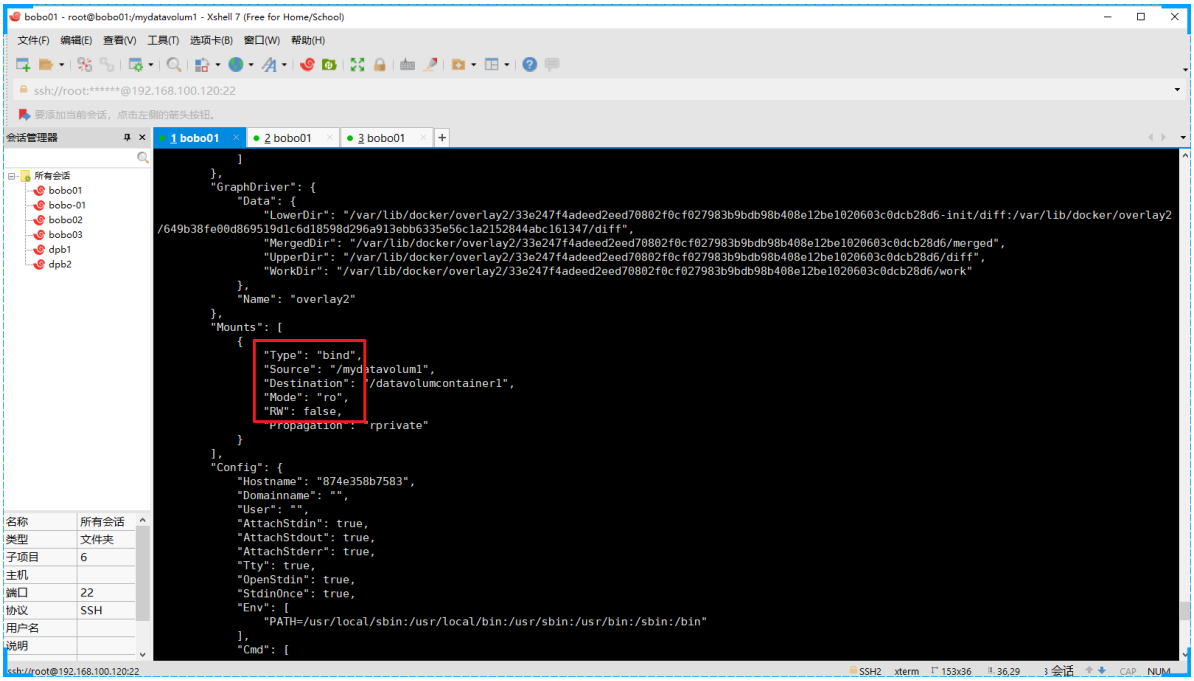
权限控制：不允许在容器中修改



修改权限

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





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
```

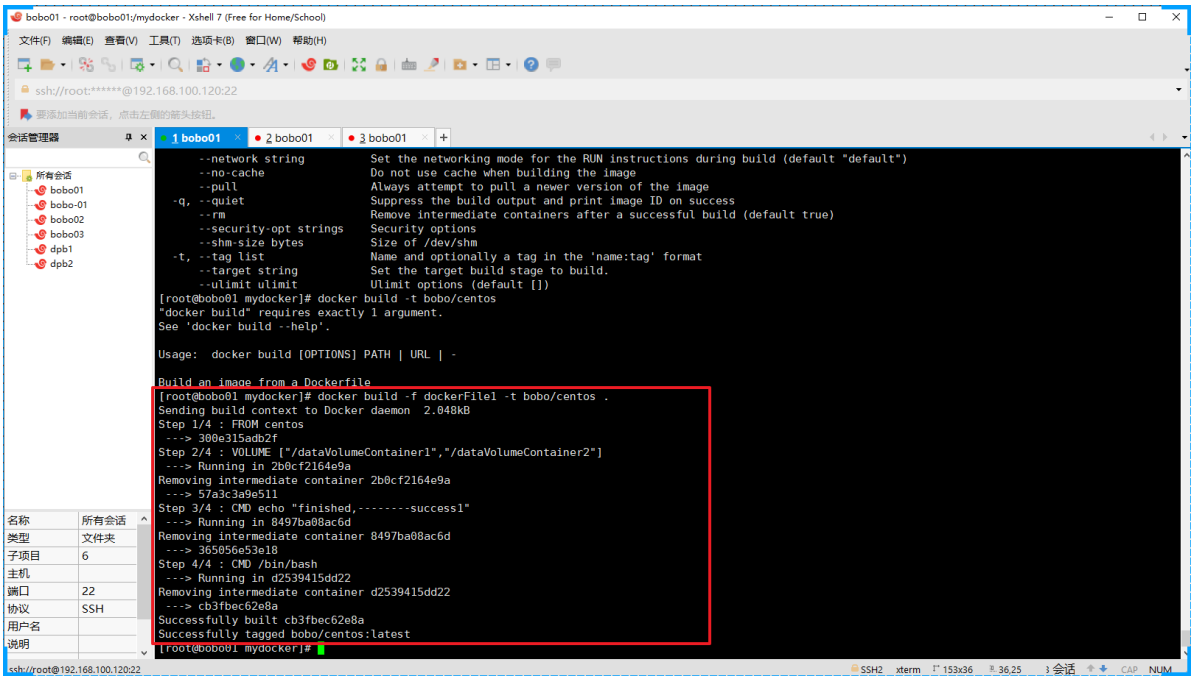
根据这个DockerFile构建我们的镜像文件

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

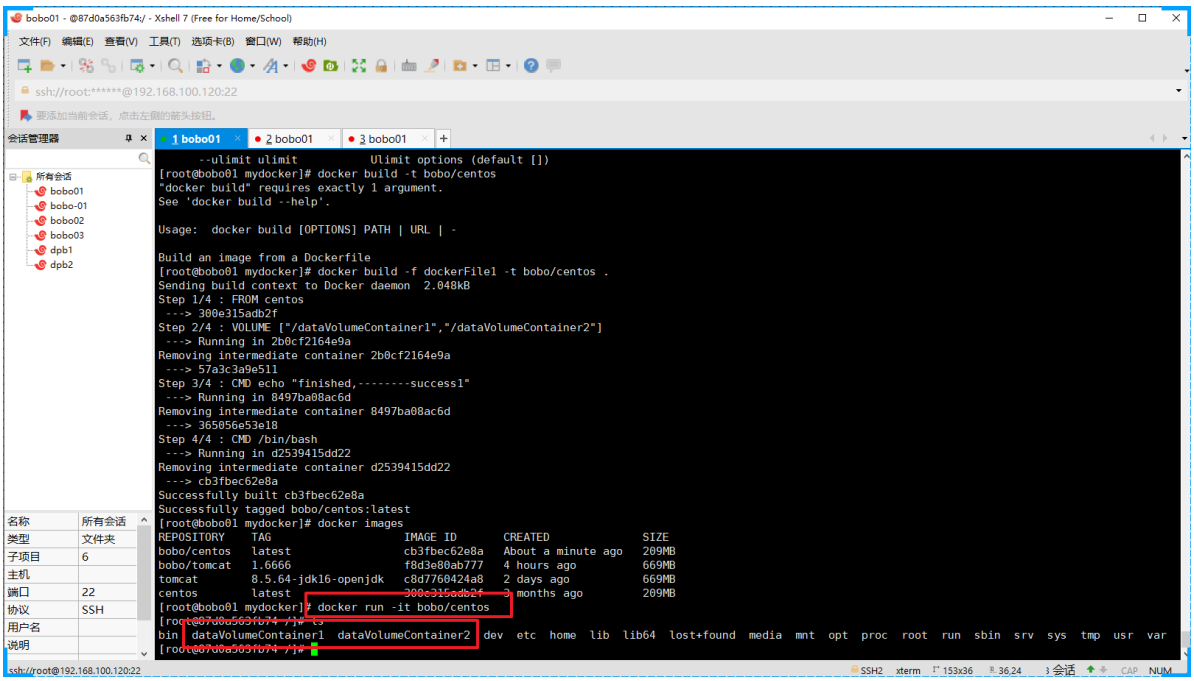
-f DockerFile文件的路径

-t 标签

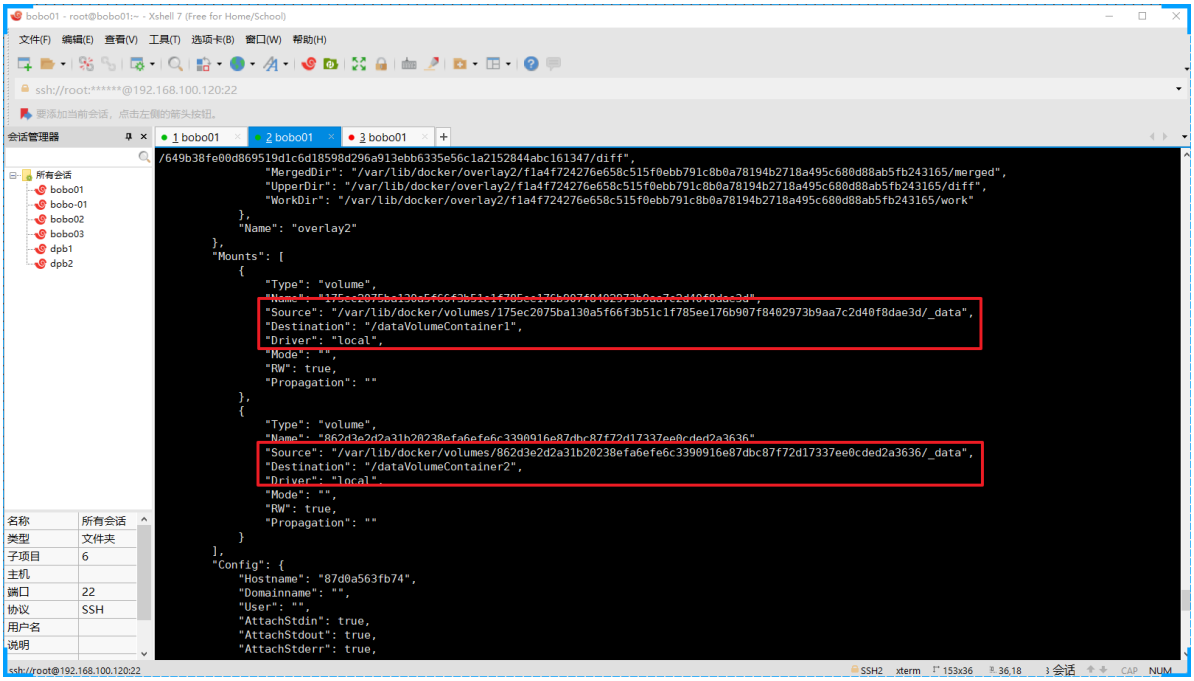
. 当前路径



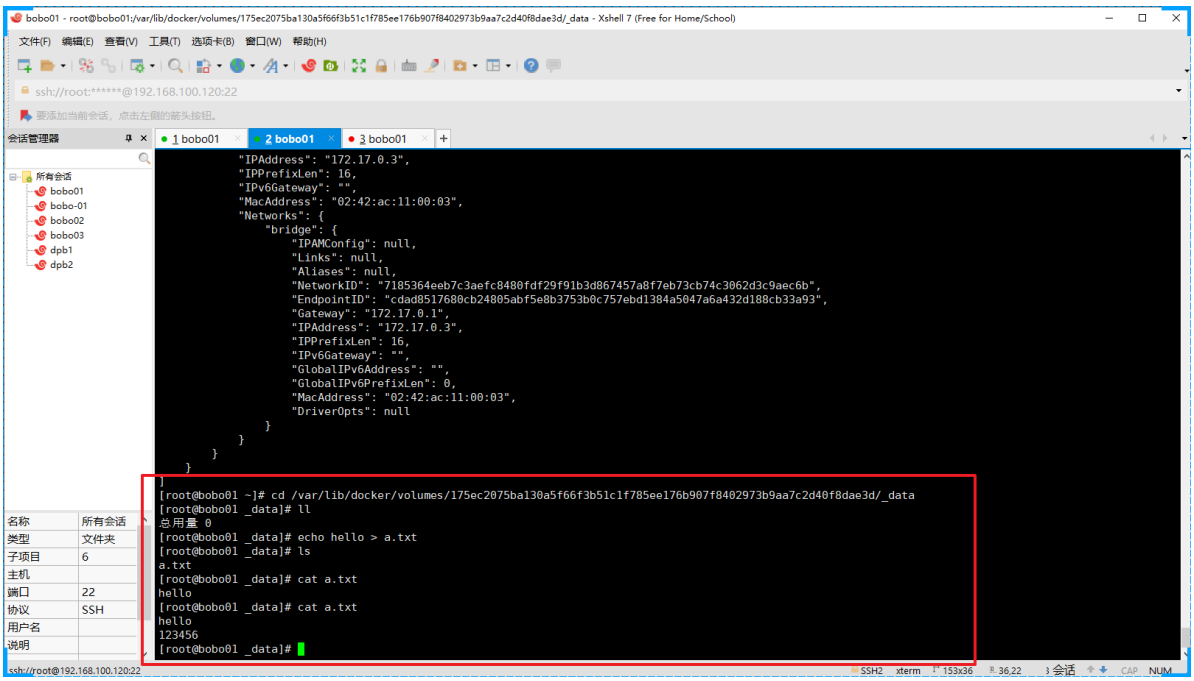
根据新创建的镜像文件创建一个容器，启动后我们可以看到在容器中创建的有对应的目录

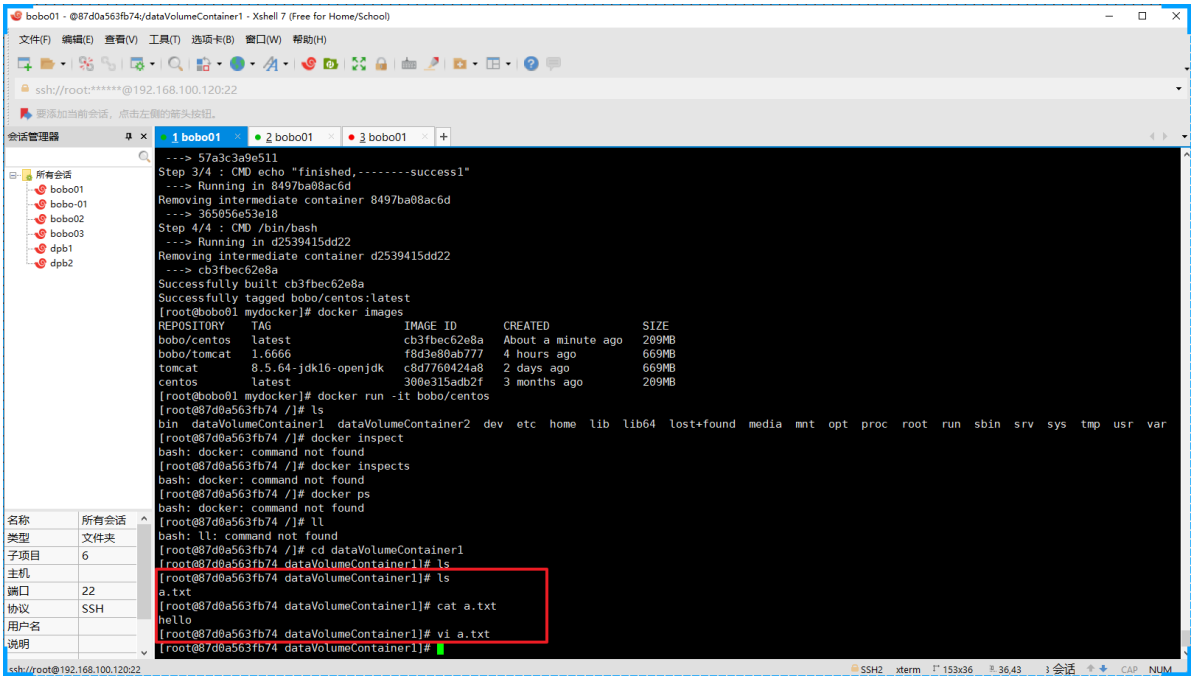


这两个目录和宿主机的映射目录在哪呢？这时我们可以通过 inspect命令查看



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



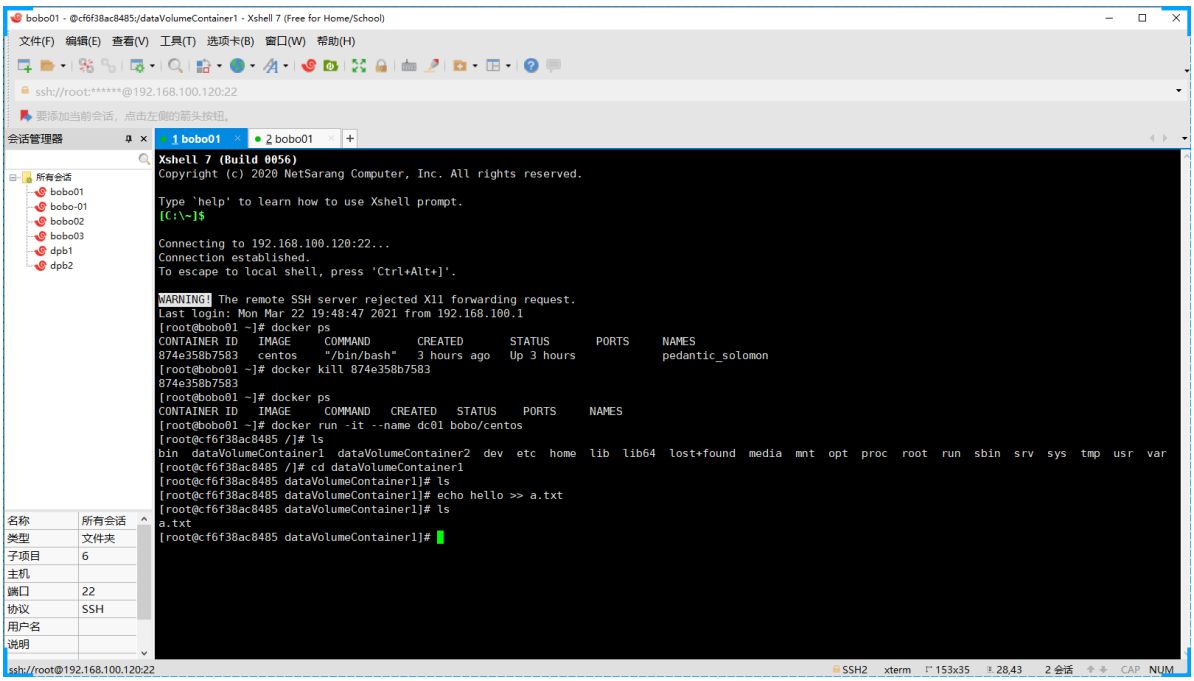


2. 数据卷容器

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。

2.1 启动一个父容器

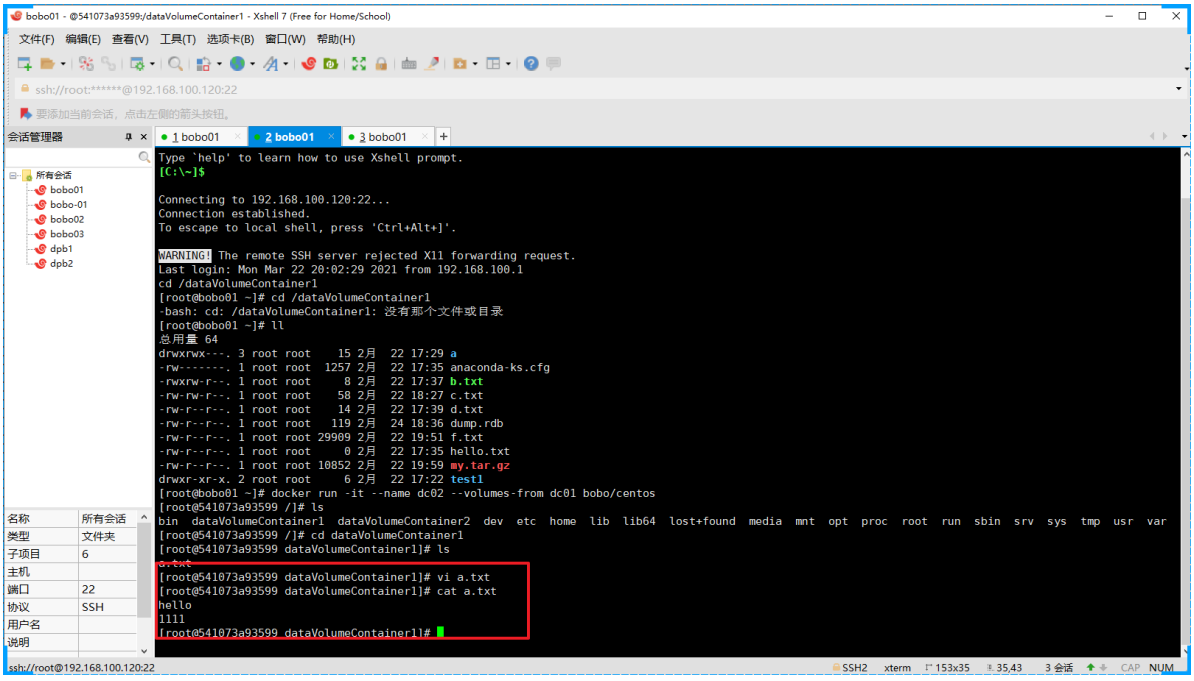
```
docker run -it --name dc01 bobo/centos
```

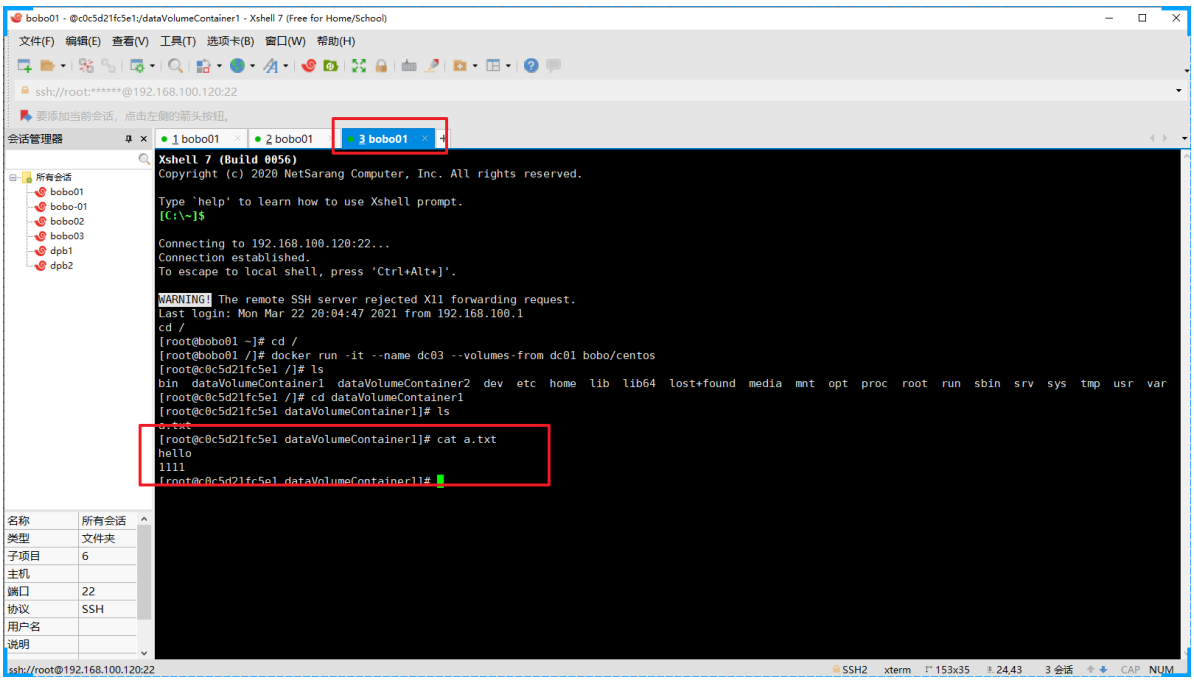


2.2 创建两个子容器

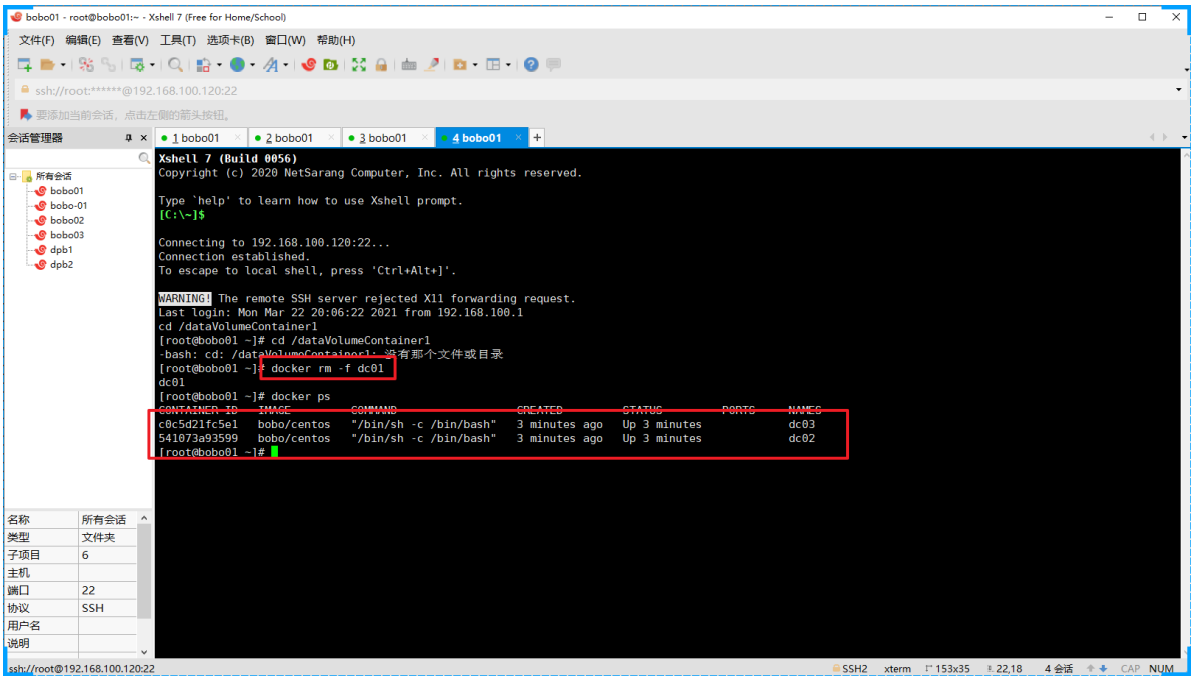
```
docker run -it --name dc02 --volumes-from dc01 bobo/centos
docker run -it --name dc03 --volumes-from dc01 bobo/centos
```

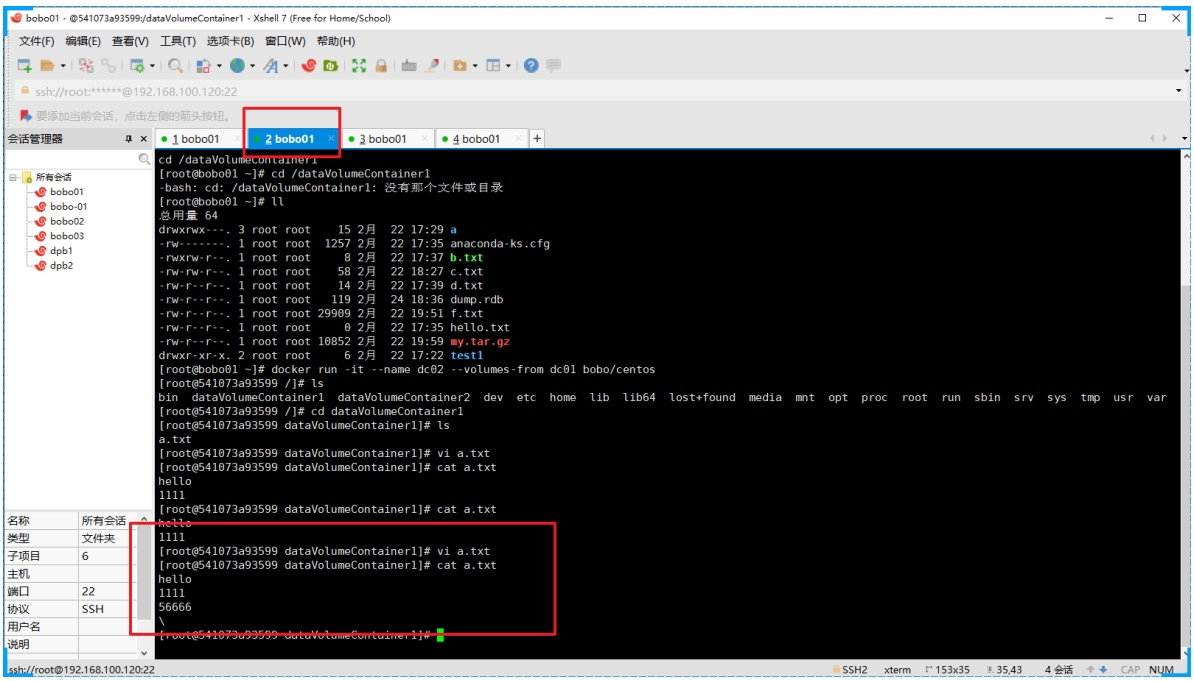
创建了两个子容器后，首先都可以看到dc01中的共享资源。第二个在dc01中修改了共享资源文件后，在两个容器中也是可见的。

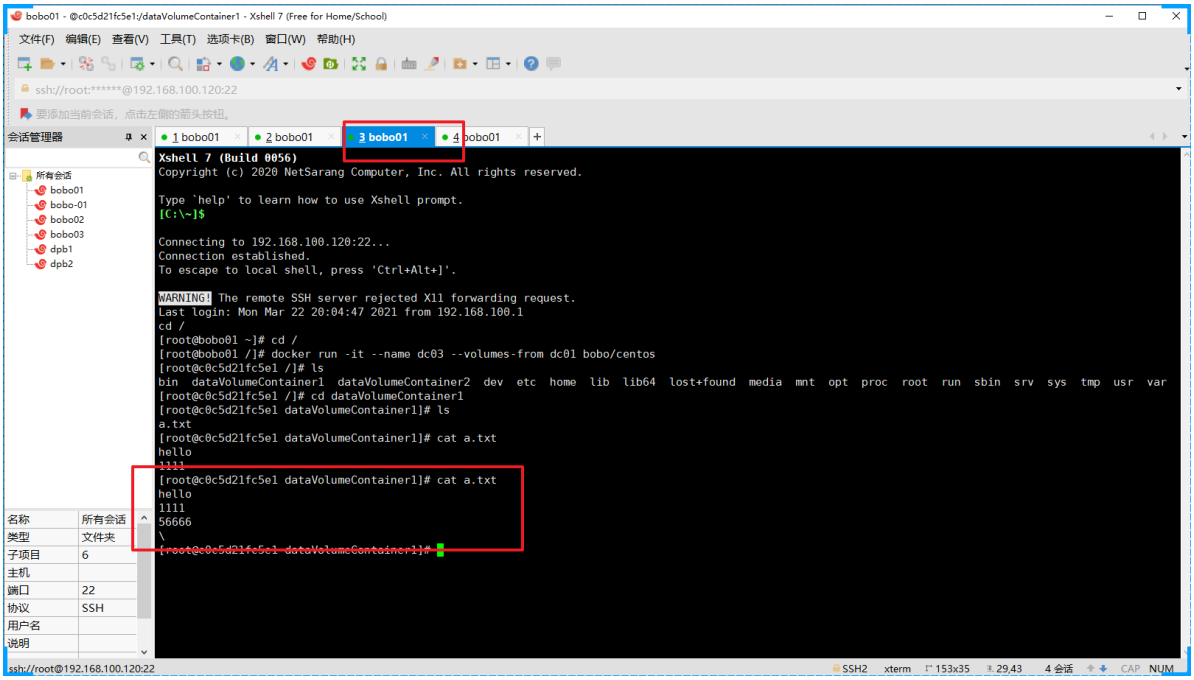




注意，删除dc01后，dc02和dc03之间数据还是共享的



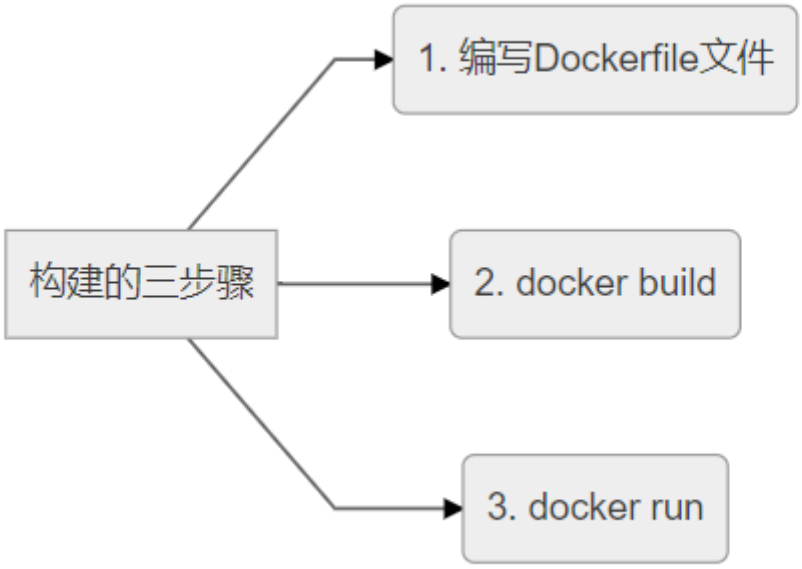




注意：容器之间配置信息的传递，数据卷的生命周期一直持续到没有容器使用它为止。

3. DockerFile

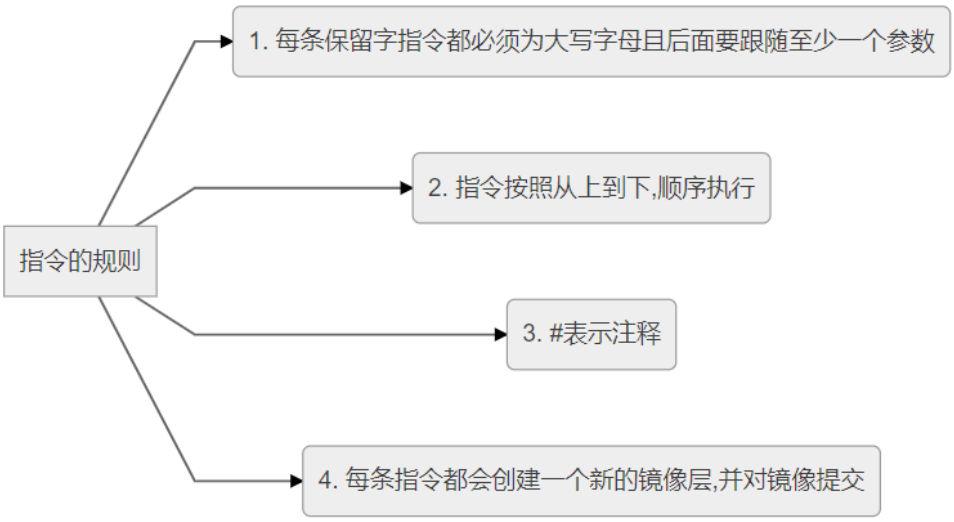
DockerFile是用来构建Docker镜像的 构建文件 ，是由一系列 命令 和 参数 构成的 脚本 。




```
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"]
```

3.1. 构建过程

Dockerfile中的指令需要满足如下的规则



3.2 执行流程

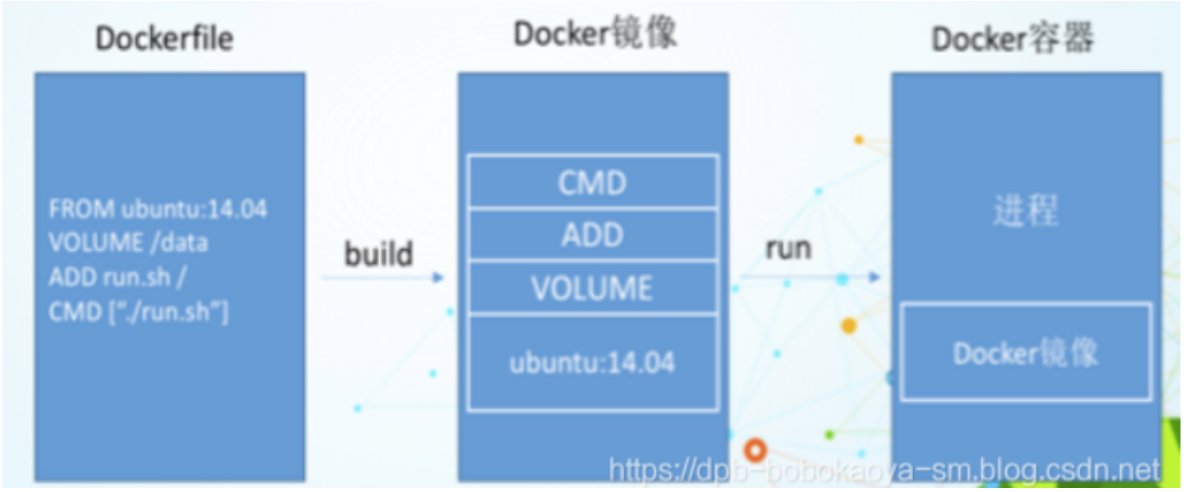
docker执行一个Dockerfile脚本的流程大致如下：

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。

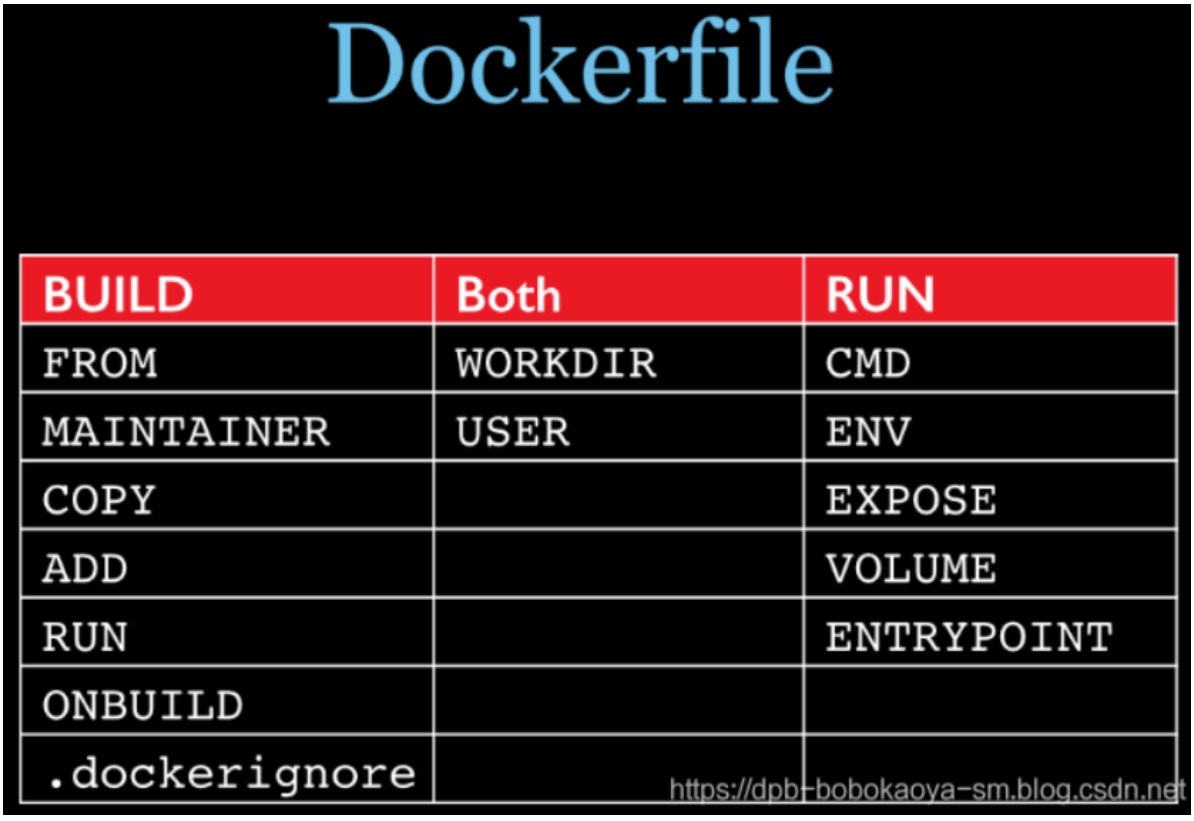
Dockerfile面向开发，Docker镜像成为交付标准，Docker容器则涉及部署与运维，三者缺一不可，合力充当Docker体系的基石。



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



3.3 DockerFile案例

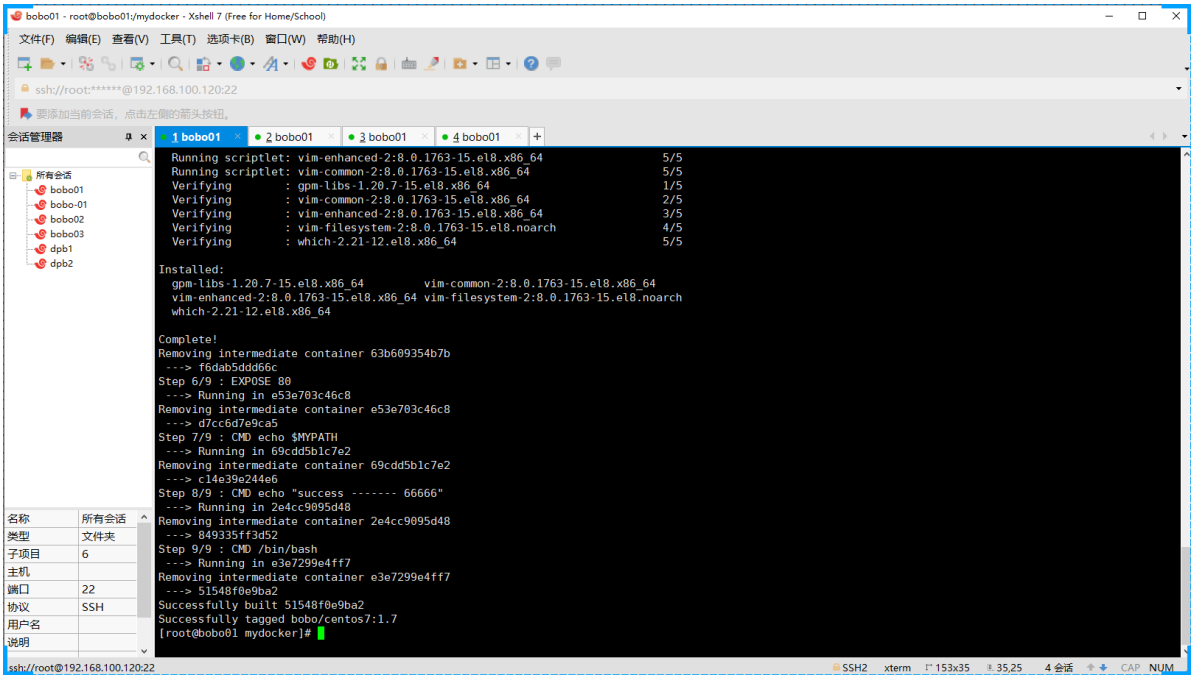
我们从官方pull下来的centos是没有vim命令的，我们可以自定义一个镜像，功能比官方提供的强大一点即可

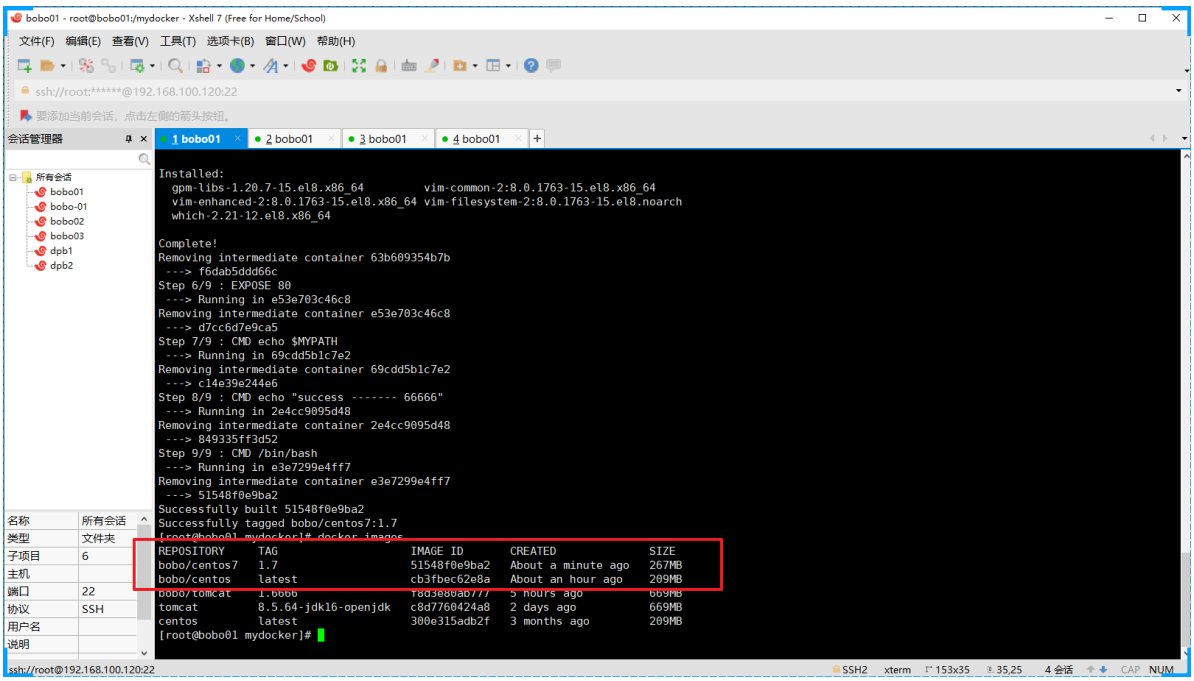
创建DockerFile文件

```
FROM centos
MAINTAINER bobo<dengpbs163.com>


ENV MYPATH /usr/local
WORKDIR $MYPATH

RUN yum -y install vim

EXPOSE 80
CMD echo $MYPATH
CMD echo "success -------- 66666"
CMD /bin/bash
```

构建镜像文件

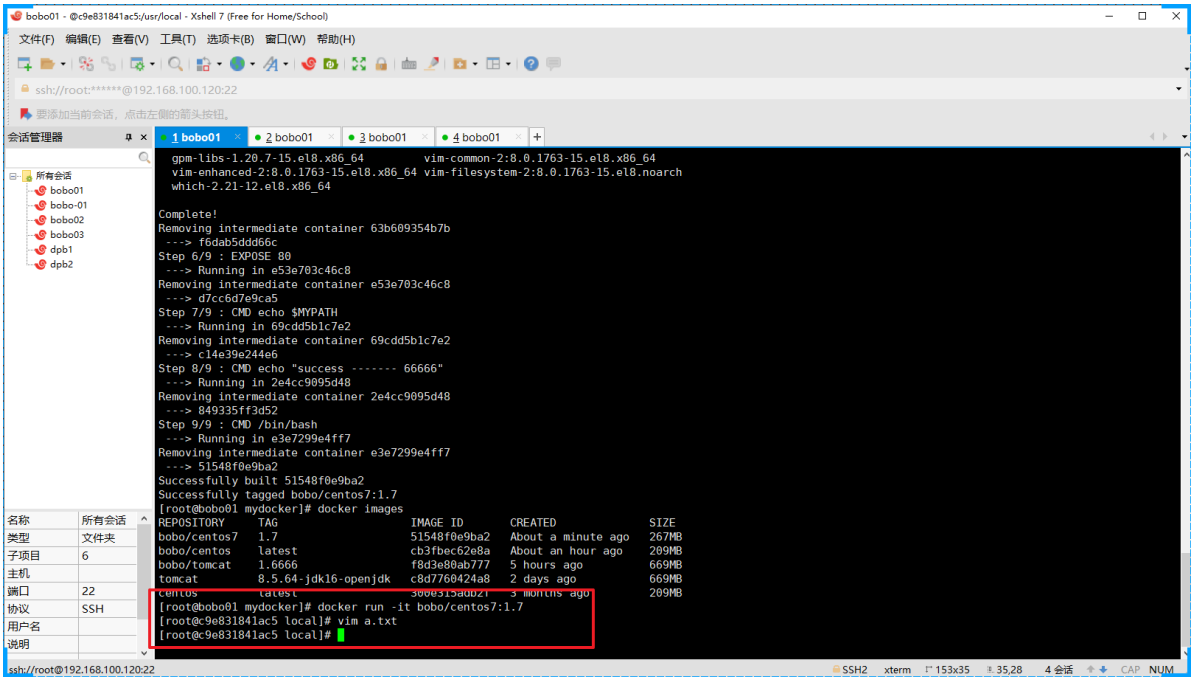
```
docker build -f DockerFile文件路径 -t 镜像名称:tag .
```





运行容器

```
docker run -it bobo/centos7:1.7
```

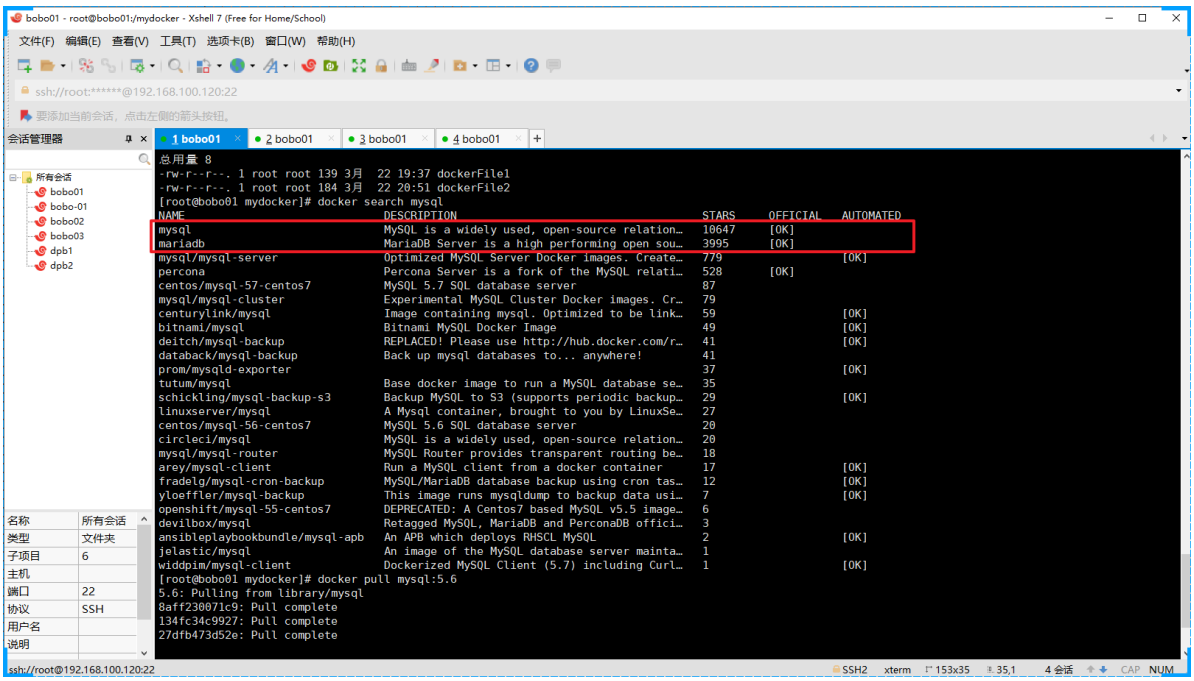


五、Docker常用软件安装

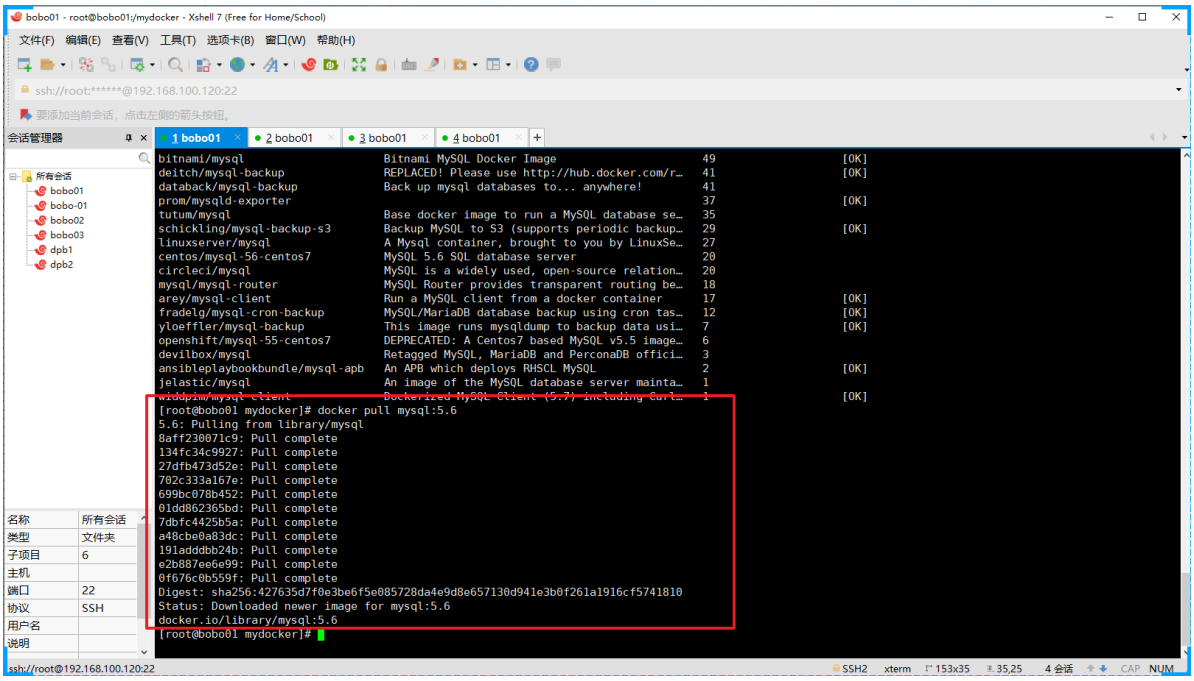
1. MySQL的安装

search命令查询

```
docker search mysql
```



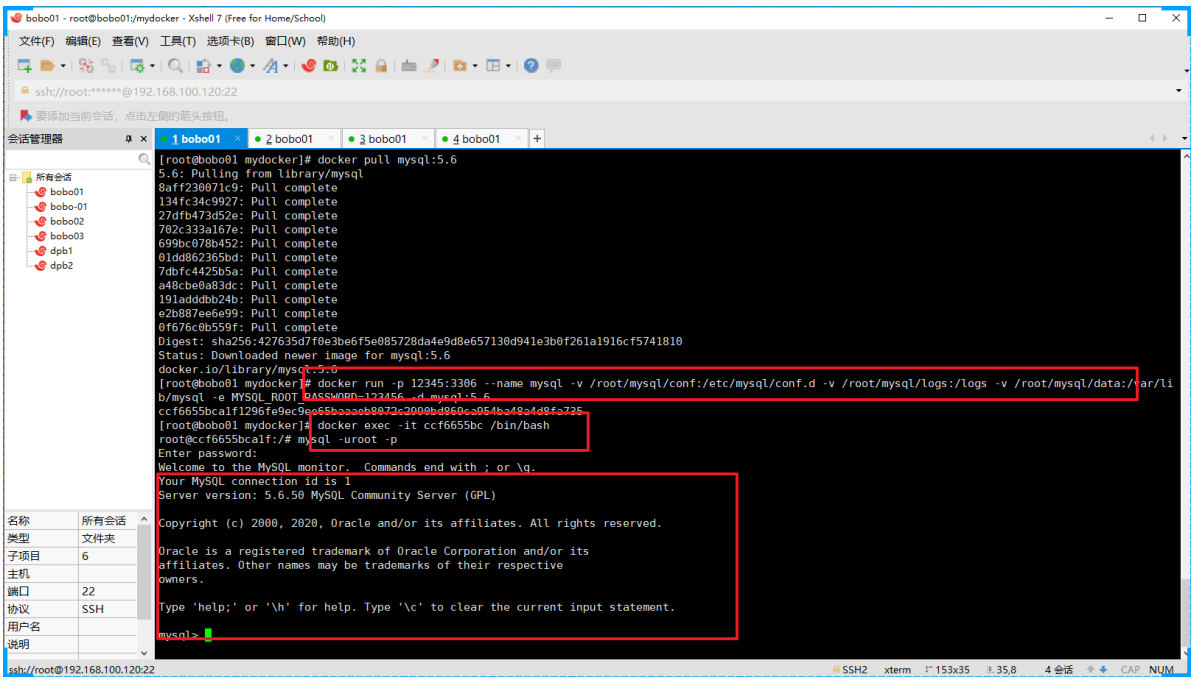
然后下载对应的mysql镜像



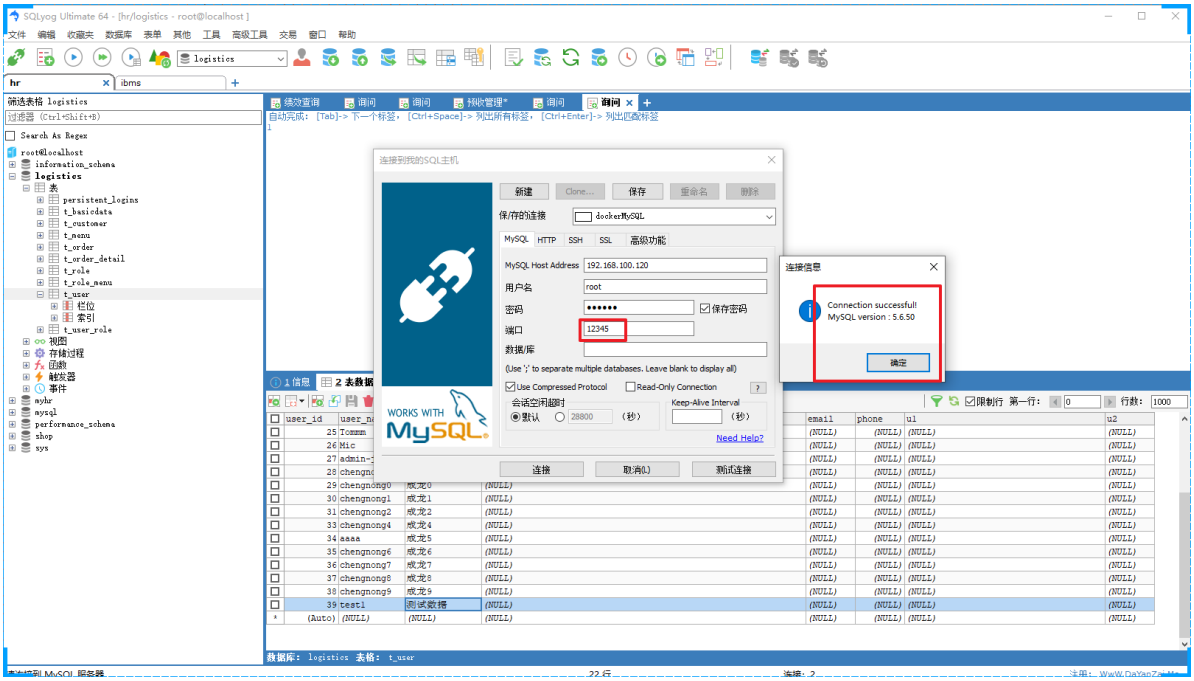
构建容器

```
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

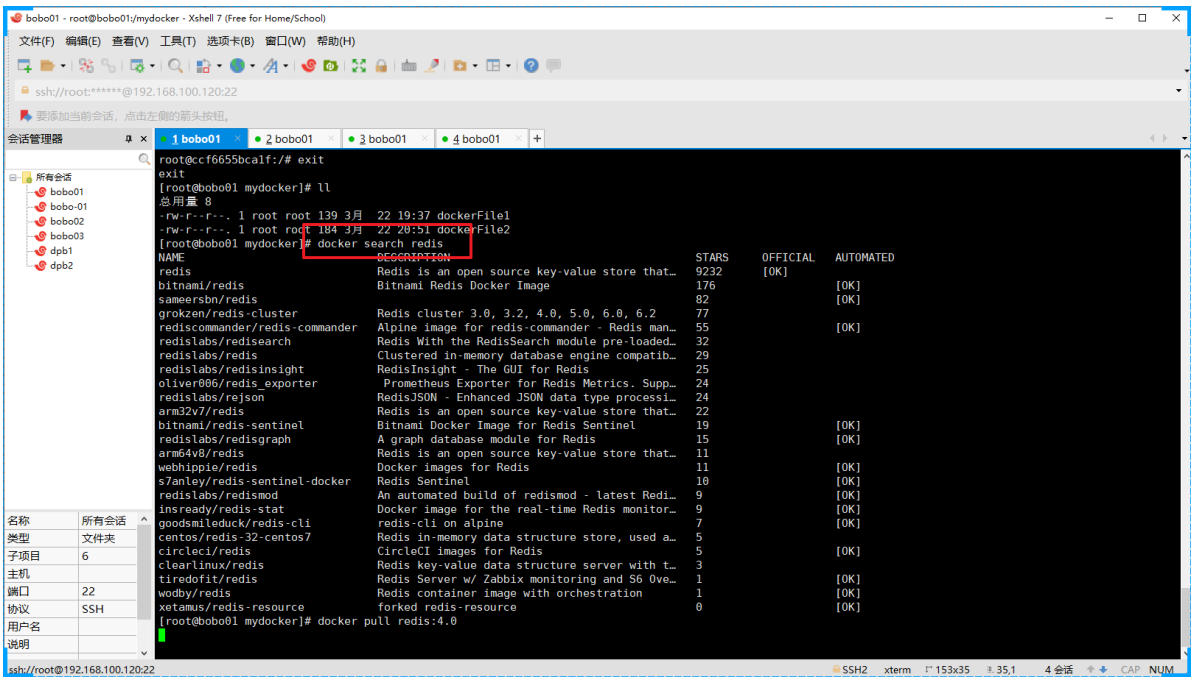


我们也可以在Windows平台中通过MySQL数据库客户端来连接容器中的数据库

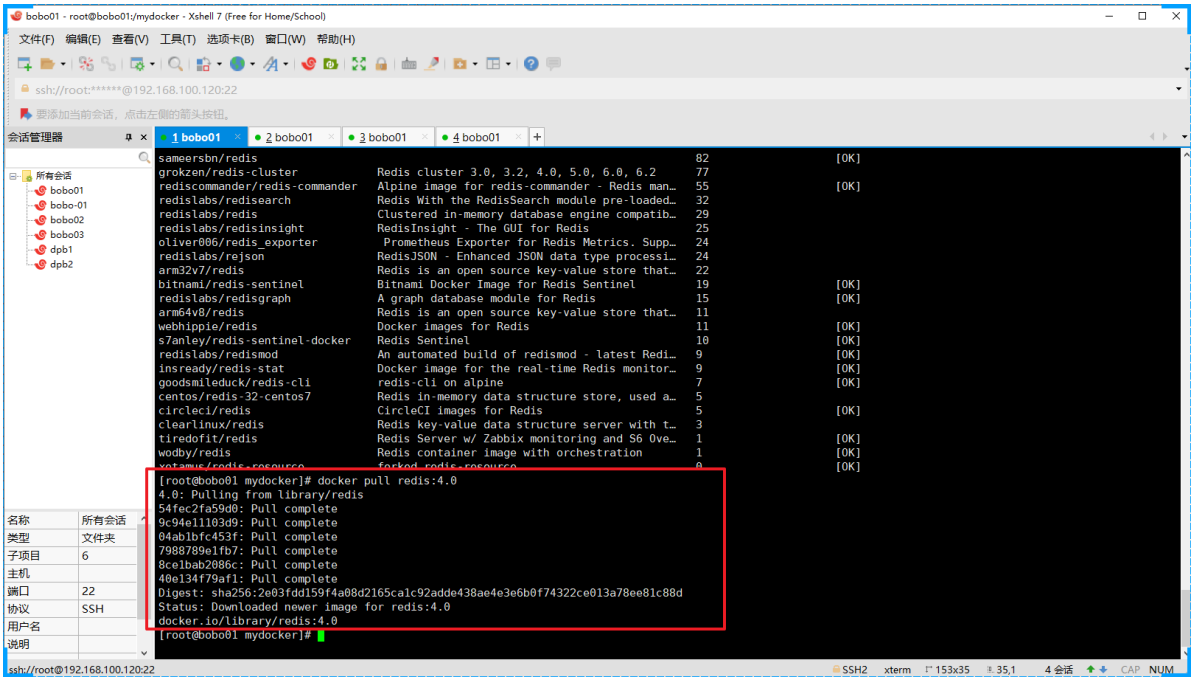


2. Redis的安装

搜索Redis

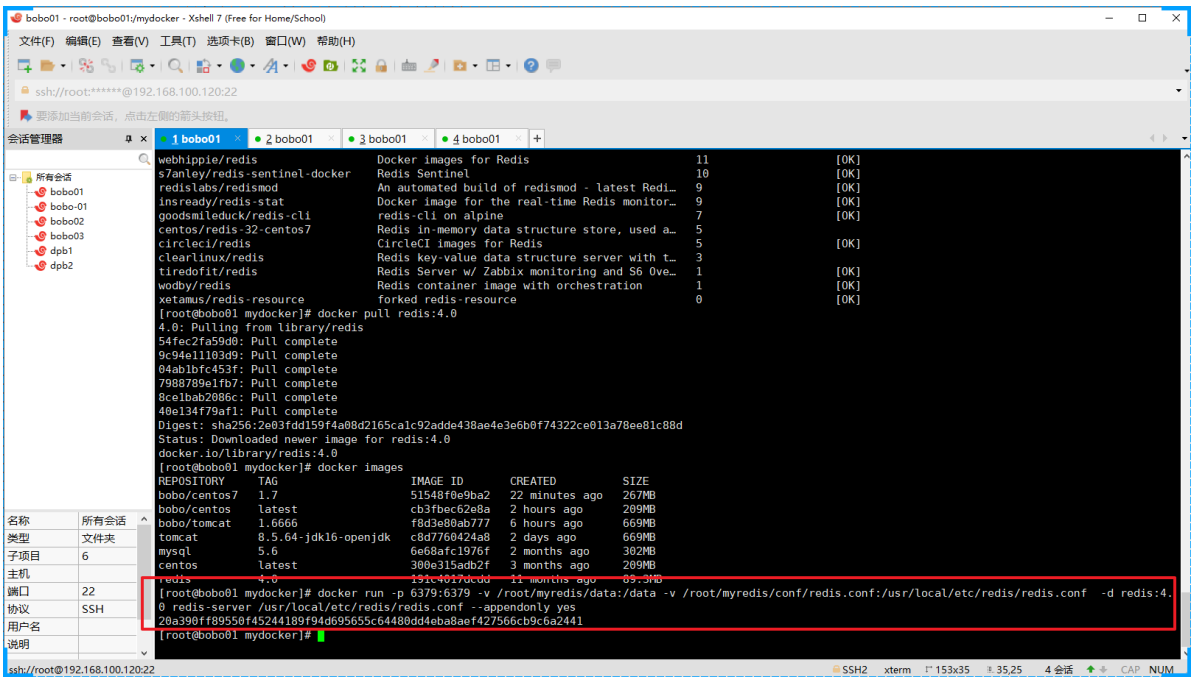


下载对应镜像文件



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
```



redis的配置文件

```
# Redis configuration file example.

#

# Note that in order to read the configuration file, Redis must be

# started with the file path as first argument:

#

# ./redis-server /path/to/redis.conf



# Note on units: when memory size is needed, it is possible to specify

# it in the usual form of 1k 5GB 4M and so forth:

#

# 1k => 1000 bytes

# 1kb => 1024 bytes

# 1m => 1000000 bytes

# 1mb => 1024*1024 bytes

# 1g => 1000000000 bytes

# 1gb => 1024*1024*1024 bytes

#
```

```
# units are case insensitive so 1GB 1Gb 1gB are all the same.

################################## INCLUDES ###################################


# Include one or more other config files here.  This is useful if you

# have a standard template that goes to all Redis servers but also need

# to customize a few per-server settings.  Include files can include

# other files, so use this wisely.

#

# Notice option "include" won't be rewritten by command "CONFIG REWRITE"

# from admin or Redis Sentinel. Since Redis always uses the last processed

# line as value of a configuration directive, you'd better put includes

# at the beginning of this file to avoid overwriting config change at runtime.

#

# If instead you are interested in using includes to override configuration

# options, it is better to use include as the last line.

#

# include /path/to/local.conf

# include /path/to/other.conf


################################## NETWORK #####################################


# By default, if no "bind" configuration directive is specified, Redis listens

# for connections from all the network interfaces available on the server.

# It is possible to listen to just one or multiple selected interfaces using

# the "bind" configuration directive, followed by one or more IP addresses.

#

# Examples:

#

# bind 192.168.1.100 10.0.0.1
```

```
# bind 127.0.0.1 ::1
#
# ~~~ WARNING ~~~ If the computer running Redis is directly exposed to the
# internet, binding to all the interfaces is dangerous and will expose the
# instance to everybody on the internet. So by default we uncomment the
# following bind directive, that will force Redis to listen only into
# the IPv4 lookback interface address (this means Redis will be able to
# accept connections only from clients running into the same computer it
# is running).
#
# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES
# JUST COMMENT THE FOLLOWING LINE.
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#bind 127.0.0.1


# Protected mode is a layer of security protection, in order to avoid that
# Redis instances left open on the internet are accessed and exploited.
#
# When protected mode is on and if:
#
# 1) The server is not binding explicitly to a set of addresses using the
#    "bind" directive.
# 2) No password is configured.
#
# The server only accepts connections from clients connecting from the
# IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix domain
# sockets.
#
# By default protected mode is enabled. You should disable it only if
```

```
# you are sure you want clients from other hosts to connect to Redis

# even if no authentication is configured, nor a specific set of interfaces

# are explicitly listed using the "bind" directive.

protected-mode yes



# Accept connections on the specified port, default is 6379 (IANA #815344).

# If port 0 is specified Redis will not listen on a TCP socket.

port 6379



# TCP listen() backlog.

#

# In high requests-per-second environments you need an high backlog in order

# to avoid slow clients connections issues. Note that the Linux kernel

# will silently truncate it to the value of /proc/sys/net/core/somaxconn so

# make sure to raise both the value of somaxconn and tcp_max_syn_backlog

# in order to get the desired effect.

tcp-backlog 511



# Unix socket.

#

# Specify the path for the Unix socket that will be used to listen for

# incoming connections. There is no default, so Redis will not listen

# on a unix socket when not specified.

#

# unixsocket /tmp/redis.sock

# unixsocketperm 700



# Close the connection after a client is idle for N seconds (0 to disable)

timeout 0
```

```
# TCP keepalive.
#
# If non-zero, use SO_KEEPALIVE to send TCP ACKs to clients in absence
# of communication. This is useful for two reasons:
#
# 1) Detect dead peers.
# 2) Take the connection alive from the point of view of network
#    equipment in the middle.
#
# On Linux, the specified value (in seconds) is the period used to send ACKs.
# Note that to close the connection the double of the time is needed.
# On other kernels the period depends on the kernel configuration.
#
# A reasonable value for this option is 300 seconds, which is the new
# Redis default starting with Redis 3.2.1.
tcp-keepalive 300

################################# GENERAL #####################################

# By default Redis does not run as a daemon. Use 'yes' if you need it.
# Note that Redis will write a pid file in /var/run/redis.pid when daemonized.
#daemonize no

# If you run Redis from upstart or systemd, Redis can interact with your
# supervision tree. Options:
#   supervised no      - no supervision interaction
#   supervised upstart - signal upstart by putting Redis into SIGSTOP mode
#   supervised systemd - signal systemd by writing READY=1 to $NOTIFY_SOCKET
```

```
#   supervised auto    - detect upstart or systemd method based on
#                        UPSTART_JOB or NOTIFY_SOCKET environment variables
# Note: these supervision methods only signal "process is ready."
#       They do not enable continuous liveness pings back to your supervisor.
supervised no

# If a pid file is specified, Redis writes it where specified at startup
# and removes it at exit.
#
# When the server runs non daemonized, no pid file is created if none is
# specified in the configuration. When the server is daemonized, the pid file
# is used even if not specified, defaulting to "/var/run/redis.pid".
#
# Creating a pid file is best effort: if Redis is not able to create it
# nothing bad happens, the server will start and run normally.
pidfile /var/run/redis_6379.pid

# Specify the server verbosity level.
# This can be one of:
# debug (a lot of information, useful for development/testing)
# verbose (many rarely useful info, but not a mess like the debug level)
# notice (moderately verbose, what you want in production probably)
# warning (only very important / critical messages are logged)
loglevel notice

# Specify the log file name. Also the empty string can be used to force
# Redis to log on the standard output. Note that if you use standard
# output for logging but daemonize, logs will be sent to /dev/null
logfile ""
```

```
# To enable logging to the system logger, just set 'syslog-enabled' to yes,

# and optionally update the other syslog parameters to suit your needs.

# syslog-enabled no


# Specify the syslog identity.

# syslog-ident redis


# Specify the syslog facility. Must be USER or between LOCAL0-LOCAL7.

# syslog-facility local0


# Set the number of databases. The default database is DB 0, you can select

# a different one on a per-connection basis using SELECT <dbid> where

# dbid is a number between 0 and 'databases'-1

databases 16


################################ SNAPSHOTTING  ################################

#

# Save the DB on disk:

#

#   save <seconds> <changes>

#

#   Will save the DB if both the given number of seconds and the given

#   number of write operations against the DB occurred.

#

#   In the example below the behaviour will be to save:

#   after 900 sec (15 min) if at least 1 key changed

#   after 300 sec (5 min) if at least 10 keys changed

#   after 60 sec if at least 10000 keys changed
```

```
#
#   Note: you can disable saving completely by commenting out all "save" lines.
#
#   It is also possible to remove all the previously configured save
#   points by adding a save directive with a single empty string argument
#   like in the following example:
#
#   save ""

save 120 1
save 300 10
save 60 10000

# By default Redis will stop accepting writes if RDB snapshots are enabled
# (at least one save point) and the latest background save failed.
# This will make the user aware (in a hard way) that data is not persisting
# on disk properly, otherwise chances are that no one will notice and some
# disaster will happen.
#
# If the background saving process will start working again Redis will
# automatically allow writes again.
#
# However if you have setup your proper monitoring of the Redis server
# and persistence, you may want to disable this feature so that Redis will
# continue to work as usual even if there are problems with disk,
# permissions, and so forth.
stop-writes-on-bgsave-error yes

# Compress string objects using LZF when dump .rdb databases?
```

```
# For default that's set to 'yes' as it's almost always a win.

# If you want to save some CPU in the saving child set it to 'no' but

# the dataset will likely be bigger if you have compressible values or keys.

rdbcompression yes



# Since version 5 of RDB a CRC64 checksum is placed at the end of the file.

# This makes the format more resistant to corruption but there is a performance

# hit to pay (around 10%) when saving and loading RDB files, so you can disable
it

# for maximum performances.

#

# RDB files created with checksum disabled have a checksum of zero that will

# tell the loading code to skip the check.

rdbchecksum yes



# The filename where to dump the DB

dbfilename dump.rdb



# The working directory.

#

# The DB will be written inside this directory, with the filename specified

# above using the 'dbfilename' configuration directive.

#

# The Append Only File will also be created inside this directory.

#

# Note that you must specify a directory here, not a file name.

dir ./



################################ REPLICATION #################################
```

```
# Master-Slave replication. Use slaveof to make a Redis instance a copy of
# another Redis server. A few things to understand ASAP about Redis replication.
#
# 1) Redis replication is asynchronous, but you can configure a master to
#    stop accepting writes if it appears to be not connected with at least
#    a given number of slaves.
# 2) Redis slaves are able to perform a partial resynchronization with the
#    master if the replication link is lost for a relatively small amount of
#    time. You may want to configure the replication backlog size (see the next
#    sections of this file) with a sensible value depending on your needs.
# 3) Replication is automatic and does not need user intervention. After a
#    network partition slaves automatically try to reconnect to masters
#    and resynchronize with them.
#
# slaveof <masterip> <masterport>

# If the master is password protected (using the "requirepass" configuration
# directive below) it is possible to tell the slave to authenticate before
# starting the replication synchronization process, otherwise the master will
# refuse the slave request.
#
# masterauth <master-password>

# When a slave loses its connection with the master, or when the replication
# is still in progress, the slave can act in two different ways:
#
# 1) if slave-serve-stale-data is set to 'yes' (the default) the slave will
#    still reply to client requests, possibly with out of date data, or the
#    data set may just be empty if this is the first synchronization.
```

```
#
# 2) if slave-serve-stale-data is set to 'no' the slave will reply with
#    an error "SYNC with master in progress" to all the kind of commands
#    but to INFO and SLAVEOF.
#
slave-serve-stale-data yes

# You can configure a slave instance to accept writes or not. Writing against
# a slave instance may be useful to store some ephemeral data (because data
# written on a slave will be easily deleted after resync with the master) but
# may also cause problems if clients are writing to it because of a
# misconfiguration.
#
# Since Redis 2.6 by default slaves are read-only.
#
# Note: read only slaves are not designed to be exposed to untrusted clients
# on the internet. It's just a protection layer against misuse of the instance.
# Still a read only slave exports by default all the administrative commands
# such as CONFIG, DEBUG, and so forth. To a limited extent you can improve
# security of read only slaves using 'rename-command' to shadow all the
# administrative / dangerous commands.
slave-read-only yes

# Replication SYNC strategy: disk or socket.
#
# -------------------------------------------------------
# WARNING: DISKLESS REPLICATION IS EXPERIMENTAL CURRENTLY
# -------------------------------------------------------
#
```

```
# New slaves and reconnecting slaves that are not able to continue the
replication

# process just receiving differences, need to do what is called a "full

# synchronization". An RDB file is transmitted from the master to the slaves.

# The transmission can happen in two different ways:

#

# 1) Disk-backed: The Redis master creates a new process that writes the RDB

#                 file on disk. Later the file is transferred by the parent

#                 process to the slaves incrementally.

# 2) Diskless: The Redis master creates a new process that directly writes the

#              RDB file to slave sockets, without touching the disk at all.

#

# With disk-backed replication, while the RDB file is generated, more slaves

# can be queued and served with the RDB file as soon as the current child
producing

# the RDB file finishes its work. With diskless replication instead once

# the transfer starts, new slaves arriving will be queued and a new transfer

# will start when the current one terminates.

#

# When diskless replication is used, the master waits a configurable amount of

# time (in seconds) before starting the transfer in the hope that multiple
slaves

# will arrive and the transfer can be parallelized.

#

# With slow disks and fast (large bandwidth) networks, diskless replication

# works better.

repl-diskless-sync no



# When diskless replication is enabled, it is possible to configure the delay

# the server waits in order to spawn the child that transfers the RDB via socket
```

```
# to the slaves.

#

# This is important since once the transfer starts, it is not possible to serve

# new slaves arriving, that will be queued for the next RDB transfer, so the
server

# waits a delay in order to let more slaves arrive.

#

# The delay is specified in seconds, and by default is 5 seconds. To disable

# it entirely just set it to 0 seconds and the transfer will start ASAP.

repl-diskless-sync-delay 5



# Slaves send PINGs to server in a predefined interval. It's possible to change

# this interval with the repl_ping_slave_period option. The default value is 10

# seconds.

#

# repl-ping-slave-period 10



# The following option sets the replication timeout for:

#

# 1) Bulk transfer I/O during SYNC, from the point of view of slave.

# 2) Master timeout from the point of view of slaves (data, pings).

# 3) Slave timeout from the point of view of masters (REPLCONF ACK pings).

#

# It is important to make sure that this value is greater than the value

# specified for repl-ping-slave-period otherwise a timeout will be detected

# every time there is low traffic between the master and the slave.

#

# repl-timeout 60



# Disable TCP_NODELAY on the slave socket after SYNC?
```

```
#
# If you select "yes" Redis will use a smaller number of TCP packets and
# less bandwidth to send data to slaves. But this can add a delay for
# the data to appear on the slave side, up to 40 milliseconds with
# Linux kernels using a default configuration.
#
# If you select "no" the delay for data to appear on the slave side will
# be reduced but more bandwidth will be used for replication.
#
# By default we optimize for low latency, but in very high traffic conditions
# or when the master and slaves are many hops away, turning this to "yes" may
# be a good idea.
repl-disable-tcp-nodelay no

# Set the replication backlog size. The backlog is a buffer that accumulates
# slave data when slaves are disconnected for some time, so that when a slave
# wants to reconnect again, often a full resync is not needed, but a partial
# resync is enough, just passing the portion of data the slave missed while
# disconnected.
#
# The bigger the replication backlog, the longer the time the slave can be
# disconnected and later be able to perform a partial resynchronization.
#
# The backlog is only allocated once there is at least a slave connected.
#
# repl-backlog-size 1mb

# After a master has no longer connected slaves for some time, the backlog
# will be freed. The following option configures the amount of seconds that
```

```
# need to elapse, starting from the time the last slave disconnected, for

# the backlog buffer to be freed.

#

# A value of 0 means to never release the backlog.

#

# repl-backlog-ttl 3600


# The slave priority is an integer number published by Redis in the INFO output.

# It is used by Redis Sentinel in order to select a slave to promote into a

# master if the master is no longer working correctly.

#

# A slave with a low priority number is considered better for promotion, so

# for instance if there are three slaves with priority 10, 100, 25 Sentinel will

# pick the one with priority 10, that is the lowest.

#

# However a special priority of 0 marks the slave as not able to perform the

# role of master, so a slave with priority of 0 will never be selected by

# Redis Sentinel for promotion.

#

# By default the priority is 100.

slave-priority 100


# It is possible for a master to stop accepting writes if there are less than

# N slaves connected, having a lag less or equal than M seconds.

#

# The N slaves need to be in "online" state.

#

# The lag in seconds, that must be <= the specified value, is calculated from

# the last ping received from the slave, that is usually sent every second.
```

```
#
# This option does not GUARANTEE that N replicas will accept the write, but
# will limit the window of exposure for lost writes in case not enough slaves
# are available, to the specified number of seconds.
#
# For example to require at least 3 slaves with a lag <= 10 seconds use:
#
# min-slaves-to-write 3
# min-slaves-max-lag 10
#
# Setting one or the other to 0 disables the feature.
#
# By default min-slaves-to-write is set to 0 (feature disabled) and
# min-slaves-max-lag is set to 10.


# A Redis master is able to list the address and port of the attached
# slaves in different ways. For example the "INFO replication" section
# offers this information, which is used, among other tools, by
# Redis Sentinel in order to discover slave instances.
# Another place where this info is available is in the output of the
# "ROLE" command of a masteer.
#
# The listed IP and address normally reported by a slave is obtained
# in the following way:
#
#   IP: The address is auto detected by checking the peer address
#   of the socket used by the slave to connect with the master.
#
#   Port: The port is communicated by the slave during the replication
```

```
#   handshake, and is normally the port that the slave is using to
#   list for connections.
#
# However when port forwarding or Network Address Translation (NAT) is
# used, the slave may be actually reachable via different IP and port
# pairs. The following two options can be used by a slave in order to
# report to its master a specific set of IP and port, so that both INFO
# and ROLE will report those values.
#
# There is no need to use both the options if you need to override just
# the port or the IP address.
#
# slave-announce-ip 5.5.5.5
# slave-announce-port 1234


################################## SECURITY ###################################


# Require clients to issue AUTH <PASSWORD> before processing any other
# commands.  This might be useful in environments in which you do not trust
# others with access to the host running redis-server.
#
# This should stay commented out for backward compatibility and because most
# people do not need auth (e.g. they run their own servers).
#
# Warning: since Redis is pretty fast an outside user can try up to
# 150k passwords per second against a good box. This means that you should
# use a very strong password otherwise it will be very easy to break.
#
# requirepass foobared
```

```
# Command renaming.
#
# It is possible to change the name of dangerous commands in a shared
# environment. For instance the CONFIG command may be renamed into something
# hard to guess so that it will still be available for internal-use tools
# but not available for general clients.
#
# Example:
#
# rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52
#
# It is also possible to completely kill a command by renaming it into
# an empty string:
#
# rename-command CONFIG ""
#
# Please note that changing the name of commands that are logged into the
# AOF file or transmitted to slaves may cause problems.

################################### LIMITS ####################################

# Set the max number of connected clients at the same time. By default
# this limit is set to 10000 clients, however if the Redis server is not
# able to configure the process file limit to allow for the specified limit
# the max number of allowed clients is set to the current file limit
# minus 32 (as Redis reserves a few file descriptors for internal uses).
#
# Once the limit is reached Redis will close all the new connections sending
```

```
# an error 'max number of clients reached'.
#
# maxclients 10000


# Don't use more memory than the specified amount of bytes.
# When the memory limit is reached Redis will try to remove keys
# according to the eviction policy selected (see maxmemory-policy).
#
# If Redis can't remove keys according to the policy, or if the policy is
# set to 'noeviction', Redis will start to reply with errors to commands
# that would use more memory, like SET, LPUSH, and so on, and will continue
# to reply to read-only commands like GET.
#
# This option is usually useful when using Redis as an LRU cache, or to set
# a hard memory limit for an instance (using the 'noeviction' policy).
#
# WARNING: If you have slaves attached to an instance with maxmemory on,
# the size of the output buffers needed to feed the slaves are subtracted
# from the used memory count, so that network problems / resyncs will
# not trigger a loop where keys are evicted, and in turn the output
# buffer of slaves is full with DELs of keys evicted triggering the deletion
# of more keys, and so forth until the database is completely emptied.
#
# In short... if you have slaves attached it is suggested that you set a lower
# limit for maxmemory so that there is some free RAM on the system for slave
# output buffers (but this is not needed if the policy is 'noeviction').
#
# maxmemory <bytes>
```

```
# MAXMEMORY POLICY: how Redis will select what to remove when maxmemory
# is reached. You can select among five behaviors:
#
# volatile-lru -> remove the key with an expire set using an LRU algorithm
# allkeys-lru -> remove any key according to the LRU algorithm
# volatile-random -> remove a random key with an expire set
# allkeys-random -> remove a random key, any key
# volatile-ttl -> remove the key with the nearest expire time (minor TTL)
# noeviction -> don't expire at all, just return an error on write operations
#
# Note: with any of the above policies, Redis will return an error on write
#       operations, when there are no suitable keys for eviction.
#
#       At the date of writing these commands are: set setnx setex append
#       incr decr rpush lpush rpushx lpushx linsert lset rpoplpush sadd
#       sinter sinterstore sunion sunionstore sdiff sdiffstore zadd zincrby
#       zunionstore zinterstore hset hsetnx hmset hincrby incrby decrby
#       getset mset msetnx exec sort
#
# The default is:
#
# maxmemory-policy noeviction


# LRU and minimal TTL algorithms are not precise algorithms but approximated
# algorithms (in order to save memory), so you can tune it for speed or
# accuracy. For default Redis will check five keys and pick the one that was
# used less recently, you can change the sample size using the following
# configuration directive.
#
```

```
# The default of 5 produces good enough results. 10 Approximates very closely
# true LRU but costs a bit more CPU. 3 is very fast but not very accurate.
#
# maxmemory-samples 5

############################## APPEND ONLY MODE ###############################

# By default Redis asynchronously dumps the dataset on disk. This mode is
# good enough in many applications, but an issue with the Redis process or
# a power outage may result into a few minutes of writes lost (depending on
# the configured save points).
#
# The Append Only File is an alternative persistence mode that provides
# much better durability. For instance using the default data fsync policy
# (see later in the config file) Redis can lose just one second of writes in a
# dramatic event like a server power outage, or a single write if something
# wrong with the Redis process itself happens, but the operating system is
# still running correctly.
#
# AOF and RDB persistence can be enabled at the same time without problems.
# If the AOF is enabled on startup Redis will load the AOF, that is the file
# with the better durability guarantees.
#
# Please check http://redis.io/topics/persistence for more information.

appendonly no

# The name of the append only file (default: "appendonly.aof")
```

```
appendfilename "appendonly.aof"



# The fsync() call tells the Operating System to actually write data on disk

# instead of waiting for more data in the output buffer. Some OS will really
flush

# data on disk, some other OS will just try to do it ASAP.

#

# Redis supports three different modes:

#

# no: don't fsync, just let the OS flush the data when it wants. Faster.

# always: fsync after every write to the append only log. Slow, Safest.

# everysec: fsync only one time every second. Compromise.

#

# The default is "everysec", as that's usually the right compromise between

# speed and data safety. It's up to you to understand if you can relax this to

# "no" that will let the operating system flush the output buffer when

# it wants, for better performances (but if you can live with the idea of

# some data loss consider the default persistence mode that's snapshotting),

# or on the contrary, use "always" that's very slow but a bit safer than

# everysec.

#

# More details please check the following article:

# http://antirez.com/post/redis-persistence-demystified.html

#

# If unsure, use "everysec".



# appendfsync always

appendfsync everysec

# appendfsync no
```

```
# When the AOF fsync policy is set to always or everysec, and a background
# saving process (a background save or AOF log background rewriting) is
# performing a lot of I/O against the disk, in some Linux configurations
# Redis may block too long on the fsync() call. Note that there is no fix for
# this currently, as even performing fsync in a different thread will block
# our synchronous write(2) call.
#
# In order to mitigate this problem it's possible to use the following option
# that will prevent fsync() from being called in the main process while a
# BGSAVE or BGREWRITEAOF is in progress.
#
# This means that while another child is saving, the durability of Redis is
# the same as "appendfsync none". In practical terms, this means that it is
# possible to lose up to 30 seconds of log in the worst scenario (with the
# default Linux settings).
#
# If you have latency problems turn this to "yes". Otherwise leave it as
# "no" that is the safest pick from the point of view of durability.

no-appendfsync-on-rewrite no

# Automatic rewrite of the append only file.
# Redis is able to automatically rewrite the log file implicitly calling
# BGREWRITEAOF when the AOF log size grows by the specified percentage.
#
# This is how it works: Redis remembers the size of the AOF file after the
# latest rewrite (if no rewrite has happened since the restart, the size of
# the AOF at startup is used).
```

```
#
# This base size is compared to the current size. If the current size is
# bigger than the specified percentage, the rewrite is triggered. Also
# you need to specify a minimal size for the AOF file to be rewritten, this
# is useful to avoid rewriting the AOF file even if the percentage increase
# is reached but it is still pretty small.
#
# Specify a percentage of zero in order to disable the automatic AOF
# rewrite feature.

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

# An AOF file may be found to be truncated at the end during the Redis
# startup process, when the AOF data gets loaded back into memory.
# This may happen when the system where Redis is running
# crashes, especially when an ext4 filesystem is mounted without the
# data=ordered option (however this can't happen when Redis itself
# crashes or aborts but the operating system still works correctly).
#
# Redis can either exit with an error when this happens, or load as much
# data as possible (the default now) and start if the AOF file is found
# to be truncated at the end. The following option controls this behavior.
#
# If aof-load-truncated is set to yes, a truncated AOF file is loaded and
# the Redis server starts emitting a log to inform the user of the event.
# Otherwise if the option is set to no, the server aborts with an error
# and refuses to start. When the option is set to no, the user requires
# to fix the AOF file using the "redis-check-aof" utility before to restart
```

```
# the server.
#
# Note that if the AOF file will be found to be corrupted in the middle
# the server will still exit with an error. This option only applies when
# Redis will try to read more data from the AOF file but not enough bytes
# will be found.
aof-load-truncated yes

################################ LUA SCRIPTING  ###############################

# Max execution time of a Lua script in milliseconds.
#
# If the maximum execution time is reached Redis will log that a script is
# still in execution after the maximum allowed time and will start to
# reply to queries with an error.
#
# When a long running script exceeds the maximum execution time only the
# SCRIPT KILL and SHUTDOWN NOSAVE commands are available. The first can be
# used to stop a script that did not yet called write commands. The second
# is the only way to shut down the server in the case a write command was
# already issued by the script but the user doesn't want to wait for the natural
# termination of the script.
#
# Set it to 0 or a negative value for unlimited execution without warnings.
lua-time-limit 5000

################################ REDIS CLUSTER  ###############################
#
# +------------------------------------------------------------------------+
```

```
# WARNING EXPERIMENTAL: Redis Cluster is considered to be stable code, however
# in order to mark it as "mature" we need to wait for a non trivial percentage
# of users to deploy it in production.
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#
# Normal Redis instances can't be part of a Redis Cluster; only nodes that are
# started as cluster nodes can. In order to start a Redis instance as a
# cluster node enable the cluster support uncommenting the following:
#
# cluster-enabled yes

# Every cluster node has a cluster configuration file. This file is not
# intended to be edited by hand. It is created and updated by Redis nodes.
# Every Redis Cluster node requires a different cluster configuration file.
# Make sure that instances running in the same system do not have
# overlapping cluster configuration file names.
#
# cluster-config-file nodes-6379.conf

# Cluster node timeout is the amount of milliseconds a node must be unreachable
# for it to be considered in failure state.
# Most other internal time limits are multiple of the node timeout.
#
# cluster-node-timeout 15000

# A slave of a failing master will avoid to start a failover if its data
# looks too old.
#
# There is no simple way for a slave to actually have a exact measure of
```

```
# its "data age", so the following two checks are performed:
#
# 1) If there are multiple slaves able to failover, they exchange messages
#    in order to try to give an advantage to the slave with the best
#    replication offset (more data from the master processed).
#    Slaves will try to get their rank by offset, and apply to the start
#    of the failover a delay proportional to their rank.
#
# 2) Every single slave computes the time of the last interaction with
#    its master. This can be the last ping or command received (if the master
#    is still in the "connected" state), or the time that elapsed since the
#    disconnection with the master (if the replication link is currently down).
#    If the last interaction is too old, the slave will not try to failover
#    at all.
#
# The point "2" can be tuned by user. Specifically a slave will not perform
# the failover if, since the last interaction with the master, the time
# elapsed is greater than:
#
#   (node-timeout * slave-validity-factor) + repl-ping-slave-period
#
# So for example if node-timeout is 30 seconds, and the slave-validity-factor
# is 10, and assuming a default repl-ping-slave-period of 10 seconds, the
# slave will not try to failover if it was not able to talk with the master
# for longer than 310 seconds.
#
# A large slave-validity-factor may allow slaves with too old data to failover
# a master, while a too small value may prevent the cluster from being able to
# elect a slave at all.
```

```
#
# For maximum availability, it is possible to set the slave-validity-factor
# to a value of 0, which means, that slaves will always try to failover the
# master regardless of the last time they interacted with the master.
# (However they'll always try to apply a delay proportional to their
# offset rank).
#
# Zero is the only value able to guarantee that when all the partitions heal
# the cluster will always be able to continue.
#
# cluster-slave-validity-factor 10


# Cluster slaves are able to migrate to orphaned masters, that are masters
# that are left without working slaves. This improves the cluster ability
# to resist to failures as otherwise an orphaned master can't be failed over
# in case of failure if it has no working slaves.
#
# Slaves migrate to orphaned masters only if there are still at least a
# given number of other working slaves for their old master. This number
# is the "migration barrier". A migration barrier of 1 means that a slave
# will migrate only if there is at least 1 other working slave for its master
# and so forth. It usually reflects the number of slaves you want for every
# master in your cluster.
#
# Default is 1 (slaves migrate only if their masters remain with at least
# one slave). To disable migration just set it to a very large value.
# A value of 0 can be set but is useful only for debugging and dangerous
# in production.
#
```

```
# cluster-migration-barrier 1

# By default Redis Cluster nodes stop accepting queries if they detect there
# is at least an hash slot uncovered (no available node is serving it).
# This way if the cluster is partially down (for example a range of hash slots
# are no longer covered) all the cluster becomes, eventually, unavailable.
# It automatically returns available as soon as all the slots are covered again.
#
# However sometimes you want the subset of the cluster which is working,
# to continue to accept queries for the part of the key space that is still
# covered. In order to do so, just set the cluster-require-full-coverage
# option to no.
#
# cluster-require-full-coverage yes

# In order to setup your cluster make sure to read the documentation
# available at http://redis.io web site.

################################## SLOW LOG ###################################

# The Redis Slow Log is a system to log queries that exceeded a specified
# execution time. The execution time does not include the I/O operations
# like talking with the client, sending the reply and so forth,
# but just the time needed to actually execute the command (this is the only
# stage of command execution where the thread is blocked and can not serve
# other requests in the meantime).
#
# You can configure the slow log with two parameters: one tells Redis
# what is the execution time, in microseconds, to exceed in order for the
```

```
# command to get logged, and the other parameter is the length of the

# slow log. When a new command is logged the oldest one is removed from the

# queue of logged commands.



# The following time is expressed in microseconds, so 1000000 is equivalent

# to one second. Note that a negative number disables the slow log, while

# a value of zero forces the logging of every command.

slowlog-log-slower-than 10000



# There is no limit to this length. Just be aware that it will consume memory.

# You can reclaim memory used by the slow log with SLOWLOG RESET.

slowlog-max-len 128



################################ LATENCY MONITOR ##############################



# The Redis latency monitoring subsystem samples different operations

# at runtime in order to collect data related to possible sources of

# latency of a Redis instance.

#

# Via the LATENCY command this information is available to the user that can

# print graphs and obtain reports.

#

# The system only logs operations that were performed in a time equal or

# greater than the amount of milliseconds specified via the

# latency-monitor-threshold configuration directive. When its value is set

# to zero, the latency monitor is turned off.

#

# By default latency monitoring is disabled since it is mostly not needed

# if you don't have latency issues, and collecting data has a performance
```

```
# impact, that while very small, can be measured under big load. Latency
# monitoring can easily be enabled at runtime using the command
# "CONFIG SET latency-monitor-threshold <milliseconds>" if needed.
latency-monitor-threshold 0

############################# EVENT NOTIFICATION ##############################

# Redis can notify Pub/Sub clients about events happening in the key space.
# This feature is documented at http://redis.io/topics/notifications
#
# For instance if keyspace events notification is enabled, and a client
# performs a DEL operation on key "foo" stored in the Database 0, two
# messages will be published via Pub/Sub:
#
# PUBLISH __keyspace@0__:foo del
# PUBLISH __keyevent@0__:del foo
#
# It is possible to select the events that Redis will notify among a set
# of classes. Every class is identified by a single character:
#
#  K     Keyspace events, published with __keyspace@<db>__ prefix.
#  E     Keyevent events, published with __keyevent@<db>__ prefix.
#  g     Generic commands (non-type specific) like DEL, EXPIRE, RENAME, ...
#  $     String commands
#  l     List commands
#  s     Set commands
#  h     Hash commands
#  z     Sorted set commands
#  x     Expired events (events generated every time a key expires)
```

```
#  e     Evicted events (events generated when a key is evicted for maxmemory)
#  A     Alias for g$lshzxe, so that the "AKE" string means all the events.
#
#  The "notify-keyspace-events" takes as argument a string that is composed
#  of zero or multiple characters. The empty string means that notifications
#  are disabled.
#
#  Example: to enable list and generic events, from the point of view of the
#           event name, use:
#
#  notify-keyspace-events Elg
#
#  Example 2: to get the stream of the expired keys subscribing to channel
#             name __keyevent@0__:expired use:
#
#  notify-keyspace-events Ex
#
#  By default all notifications are disabled because most users don't need
#  this feature and the feature has some overhead. Note that if you don't
#  specify at least one of K or E, no events will be delivered.
notify-keyspace-events ""

############################### ADVANCED CONFIG ###############################

# Hashes are encoded using a memory efficient data structure when they have a
# small number of entries, and the biggest entry does not exceed a given
# threshold. These thresholds can be configured using the following directives.
hash-max-ziplist-entries 512
hash-max-ziplist-value 64
```

```
# Lists are also encoded in a special way to save a lot of space.
# The number of entries allowed per internal list node can be specified
# as a fixed maximum size or a maximum number of elements.
# For a fixed maximum size, use -5 through -1, meaning:
# -5: max size: 64 Kb  <-- not recommended for normal workloads
# -4: max size: 32 Kb  <-- not recommended
# -3: max size: 16 Kb  <-- probably not recommended
# -2: max size: 8 Kb   <-- good
# -1: max size: 4 Kb   <-- good
# Positive numbers mean store up to _exactly_ that number of elements
# per list node.
# The highest performing option is usually -2 (8 Kb size) or -1 (4 Kb size),
# but if your use case is unique, adjust the settings as necessary.
list-max-ziplist-size -2


# Lists may also be compressed.
# Compress depth is the number of quicklist ziplist nodes from *each* side of
# the list to *exclude* from compression.  The head and tail of the list
# are always uncompressed for fast push/pop operations.  Settings are:
# 0: disable all list compression
# 1: depth 1 means "don't start compressing until after 1 node into the list,
#    going from either the head or tail"
#    So: [head]->node->node->...->node->[tail]
#    [head], [tail] will always be uncompressed; inner nodes will compress.
# 2: [head]->[next]->node->node->...->node->[prev]->[tail]
#    2 here means: don't compress head or head->next or tail->prev or tail,
#    but compress all nodes between them.
# 3: [head]->[next]->[next]->node->node->...->node->[prev]->[prev]->[tail]
```

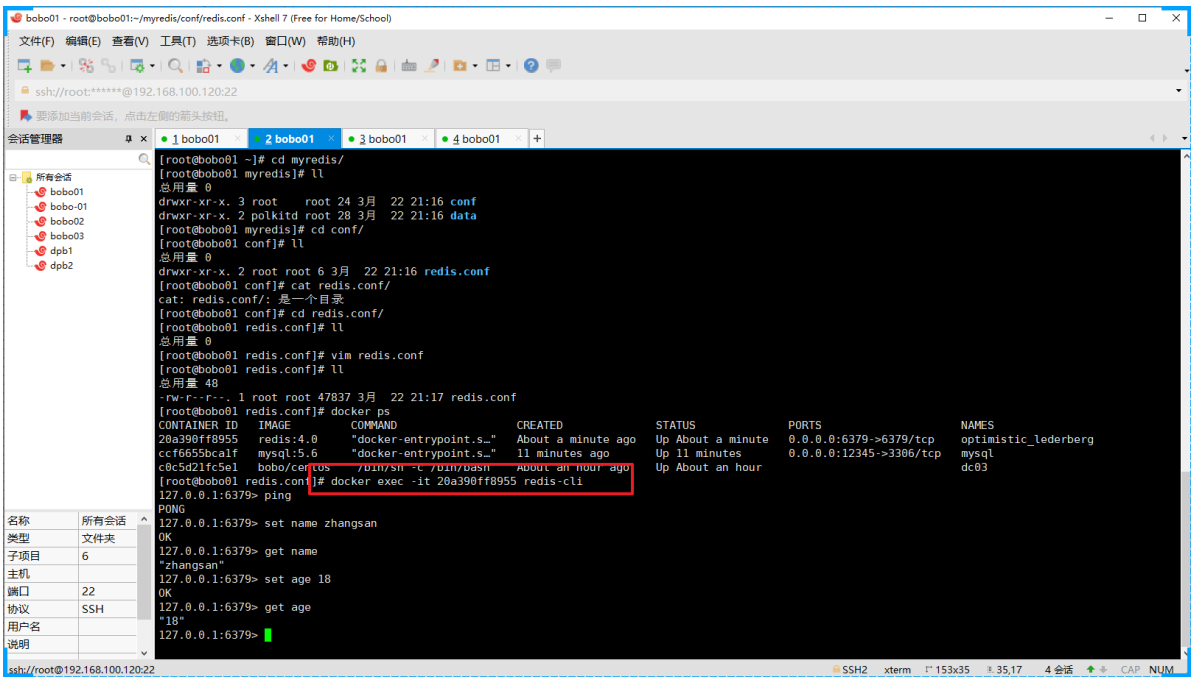
```
# etc.

list-compress-depth 0


# Sets have a special encoding in just one case: when a set is composed

# of just strings that happen to be integers in radix 10 in the range

# of 64 bit signed integers.

# The following configuration setting sets the limit in the size of the

# set in order to use this special memory saving encoding.

set-max-intset-entries 512


# Similarly to hashes and lists, sorted sets are also specially encoded in

# order to save a lot of space. This encoding is only used when the length and

# elements of a sorted set are below the following limits:

zset-max-ziplist-entries 128

zset-max-ziplist-value 64


# HyperLogLog sparse representation bytes limit. The limit includes the

# 16 bytes header. When an HyperLogLog using the sparse representation crosses

# this limit, it is converted into the dense representation.

#

# A value greater than 16000 is totally useless, since at that point the

# dense representation is more memory efficient.

#

# The suggested value is ~ 3000 in order to have the benefits of

# the space efficient encoding without slowing down too much PFADD,

# which is O(N) with the sparse encoding. The value can be raised to

# ~ 10000 when CPU is not a concern, but space is, and the data set is

# composed of many HyperLogLogs with cardinality in the 0 - 15000 range.

hll-sparse-max-bytes 3000
```

```
# Active rehashing uses 1 millisecond every 100 milliseconds of CPU time in
# order to help rehashing the main Redis hash table (the one mapping top-level
# keys to values). The hash table implementation Redis uses (see dict.c)
# performs a lazy rehashing: the more operation you run into a hash table
# that is rehashing, the more rehashing "steps" are performed, so if the
# server is idle the rehashing is never complete and some more memory is used
# by the hash table.
#
# The default is to use this millisecond 10 times every second in order to
# actively rehash the main dictionaries, freeing memory when possible.
#
# If unsure:
# use "activerehashing no" if you have hard latency requirements and it is
# not a good thing in your environment that Redis can reply from time to time
# to queries with 2 milliseconds delay.
#
# use "activerehashing yes" if you don't have such hard requirements but
# want to free memory asap when possible.
activerehashing yes


# The client output buffer limits can be used to force disconnection of clients
# that are not reading data from the server fast enough for some reason (a
# common reason is that a Pub/Sub client can't consume messages as fast as the
# publisher can produce them).
#
# The limit can be set differently for the three different classes of clients:
#
# normal -> normal clients including MONITOR clients
```

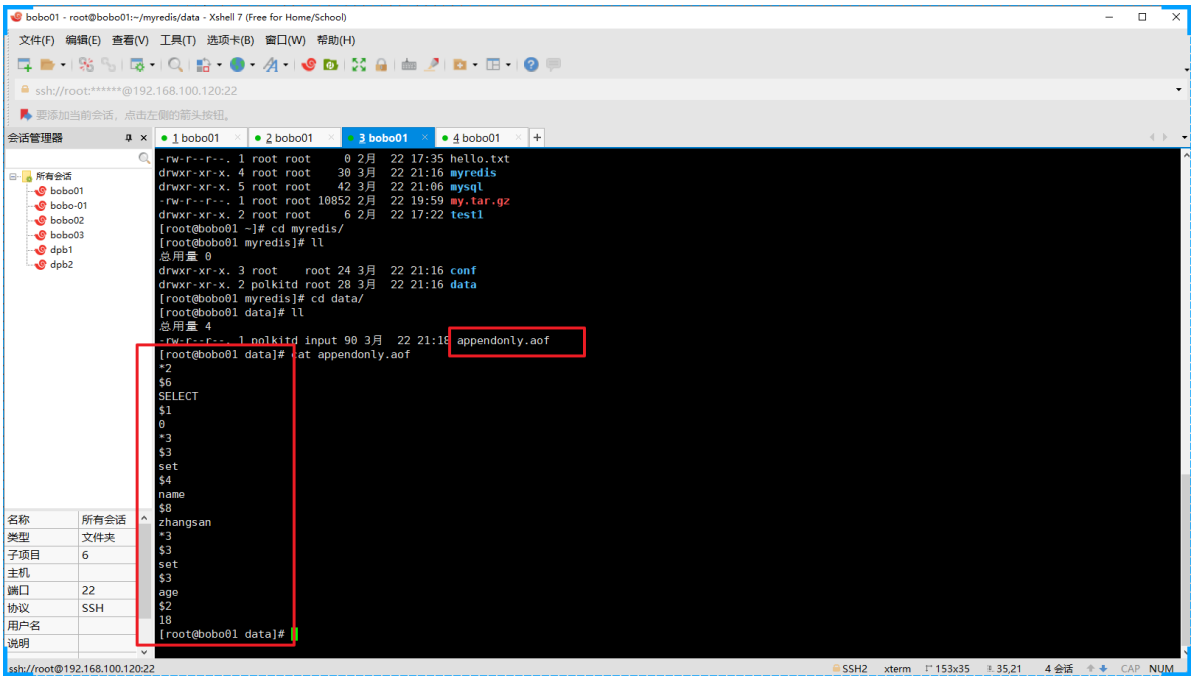
```
# slave  -> slave clients

# pubsub -> clients subscribed to at least one pubsub channel or pattern

#

# The syntax of every client-output-buffer-limit directive is the following:

#

# client-output-buffer-limit <class> <hard limit> <soft limit> <soft seconds>

#

# A client is immediately disconnected once the hard limit is reached, or if

# the soft limit is reached and remains reached for the specified number of

# seconds (continuously).

# So for instance if the hard limit is 32 megabytes and the soft limit is

# 16 megabytes / 10 seconds, the client will get disconnected immediately

# if the size of the output buffers reach 32 megabytes, but will also get

# disconnected if the client reaches 16 megabytes and continuously overcomes

# the limit for 10 seconds.

#

# By default normal clients are not limited because they don't receive data

# without asking (in a push way), but just after a request, so only

# asynchronous clients may create a scenario where data is requested faster

# than it can read.

#

# Instead there is a default limit for pubsub and slave clients, since

# subscribers and slaves receive data in a push fashion.

#

# Both the hard or the soft limit can be disabled by setting them to zero.

client-output-buffer-limit normal 0 0 0

client-output-buffer-limit slave 256mb 64mb 60

client-output-buffer-limit pubsub 32mb 8mb 60
```

```
# Redis calls an internal function to perform many background tasks, like

# closing connections of clients in timeout, purging expired keys that are

# never requested, and so forth.

#

# Not all tasks are performed with the same frequency, but Redis checks for

# tasks to perform according to the specified "hz" value.

#

# By default "hz" is set to 10. Raising the value will use more CPU when

# Redis is idle, but at the same time will make Redis more responsive when

# there are many keys expiring at the same time, and timeouts may be

# handled with more precision.

#

# The range is between 1 and 500, however a value over 100 is usually not

# a good idea. Most users should use the default of 10 and raise this up to

# 100 only in environments where very low latency is required.

hz 10



# When a child rewrites the AOF file, if the following option is enabled

# the file will be fsync-ed every 32 MB of data generated. This is useful

# in order to commit the file to the disk more incrementally and avoid

# big latency spikes.

aof-rewrite-incremental-fsync yes
```

测试连接


bobo01 - root@bobo01:~/myredis/conf/redis.conf - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://root:******@192.168.100.120:22
要添加当前会话，点击左侧的箭头按钮。
会话管理器
1 bobo01
2 bobo01
3 bobo01
4 bobo01
所有会话
bobo01
bobo-01
bobo02
bobo03
dpb1
dpb2
[root@bobo01 ~]# cd myredis/
[root@bobo01 myredis]# ll
总用量 0
drwxr-xr-x. 3 root root 24 3月 22 21:16 conf
drwxr-xr-x. 2 polkitd root 28 3月 22 21:16 data
[root@bobo01 myredis]# cd conf/
[root@bobo01 conf]# ll
总用量 0
drwxr-xr-x. 2 root root 6 3月 22 21:16 redis.conf
[root@bobo01 conf]# cat redis.conf/
cat: redis.conf/: 是一个目录
[root@bobo01 conf]# cd redis.conf/
[root@bobo01 redis.conf]# ll
总用量 0
[root@bobo01 redis.conf]# vim redis.conf
[root@bobo01 redis.conf]# ll
总用量 48
-rw-r--r--. 1 root root 47837 3月 22 21:17 redis.conf
[root@bobo01 redis.conf]# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
20a390ff8955 redis:4.0 "docker-entrypoint.s…" About a minute ago Up About a minute 0.0.0.0:6379->6379/tcp optimistic_lederberg
ccf6655bca1f mysql:5.6 "docker-entrypoint.s…" 11 minutes ago Up 11 minutes 0.0.0.0:12345->3306/tcp mysql
c0c5d21fc5e1 bobo/centos "/bin/sh -c /bin/bash" About an hour ago Up About an hour dc03
[root@bobo01 redis.conf]# docker exec -it 20a390ff8955 redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> set name zhangsan
OK
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> set age 18
OK
127.0.0.1:6379> get age
"18"
127.0.0.1:6379>
名称 所有会话
类型 文件夹
子项目 6
主机
端口 22
协议 SSH
用户名
说明
ssh://root@192.168.100.120:22
SSH2 xterm 153x35 35,17 4 会话 CAP NUM

查看持久化的文件


bobo01 - root@bobo01:~/myredis/data - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
ssh://root:******@192.168.100.120:22
要添加当前会话，点击左侧的箭头按钮。
会话管理器
1 bobo01
2 bobo01
3 bobo01
4 bobo01
所有会话
bobo01
bobo-01
bobo02
bobo03
dpb1
dpb2
-rw-r--r--. 1 root root 0 2月 22 17:35 hello.txt
drwxr-xr-x. 4 root root 30 3月 22 21:16 myredis
drwxr-xr-x. 5 root root 42 3月 22 21:06 mysql
-rw-r--r--. 1 root root 10852 2月 22 19:59 my.tar.gz
drwxr-xr-x. 2 root root 6 2月 22 17:22 test1
[root@bobo01 ~]# cd myredis/
[root@bobo01 myredis]# ll
总用量 0
drwxr-xr-x. 3 root root 24 3月 22 21:16 conf
drwxr-xr-x. 2 polkitd root 28 3月 22 21:16 data
[root@bobo01 myredis]# cd data/
[root@bobo01 data]# ll
总用量 4
-rw-r--r--. 1 polkitd input 90 3月 22 21:18 appendonly.aof
[root@bobo01 data]# cat appendonly.aof
*2
$6
SELECT
$1
0
*3
$3
set
$4
name
$8
zhangsan
*3
$3
set
$3
age
$2
18
[root@bobo01 data]#
名称 所有会话
类型 文件夹
子项目 6
主机
端口 22
协议 SSH
用户名
说明
ssh://root@192.168.100.120:22
SSH2 xterm 153x35 35,21 4 会话 CAP NUM