

第一章 Docker简介

学前储备知识：

必须，熟练掌握Linux命令

建议，Git相关的知识

1.1 Docker 能解决什么问题

难题：

软件开发中最麻烦的事之一，就是环境配置。软件从开发到上线，一般都要经过开发、测试、上线。而每个人的计算机环境配置可能都不相同，谁能保证自己开发的软件，能在每一台机器上跑起来？

每台机器必须保证：操作系统的设置，各种依赖和组件的安装都正确，软件才能运行起来。

比如，开发与部署一个 Java 应用，计算机必须安装 JDK，并配置环境变量，还必须有各种依赖。在windows上进行开发，到Linux上进行部署时这些环境又得重装，这样很就麻烦。而且系统如果需要集群，那每一台机器都得重新配置环境。

开发人员经常说："它在我的机器可以正常运行"，言下之意就是，其他机器很可能跑不了。环境配置如此麻烦，换一台机器，就要重来一次，费力费时。很多人想到，能不能从根本上解决问题，软件可以带环境安装？也就是说，安装的时候，把原始环境一模一样地复制过来。

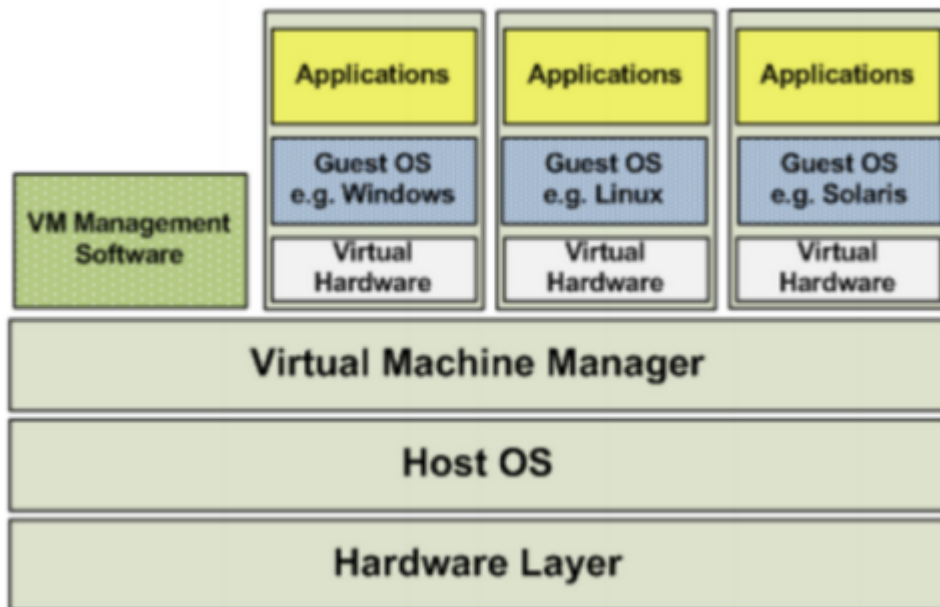
解决：

开发人员可以使用 Docker 来解决 "它在我的机器可以正常运行" 的问题，它会将运行程序的相关配置打包（打包成一个镜像），然后直接搬移到新的机器上运行。

1.2 什么是虚拟机技术

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 VMware workstation 虚拟化产品提供了虚拟的硬件，在 Windows 系统里面运行 Linux 系统。安装在虚拟机（如Linux）上的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(如Windows)来说，虚拟机(如Linux)就是一个普通文件，不需要了就删掉，对其他部分毫无影响。



- 虽然可以通过虚拟机还原软件需要的配置环境。但是虚拟机有几个缺点：

- **资源占用多**

每个虚拟机会独占一部分内存和硬盘空间。哪怕虚拟机里面的应用程序，真正使用的内存只有 1MB，虚拟机依然需要几百 MB 的内存才能运行。

- **冗余步骤多**

虚拟机是完整的操作系统，一些系统级别的操作步骤，往往无法跳过，比如用户登录。

启动慢

启动硬件上的操作系统需要多久，启动虚拟机就需要多久。可能要等几分钟，虚拟机才能真正运行。

1.3 什么是容器



1.3.1 容器技术

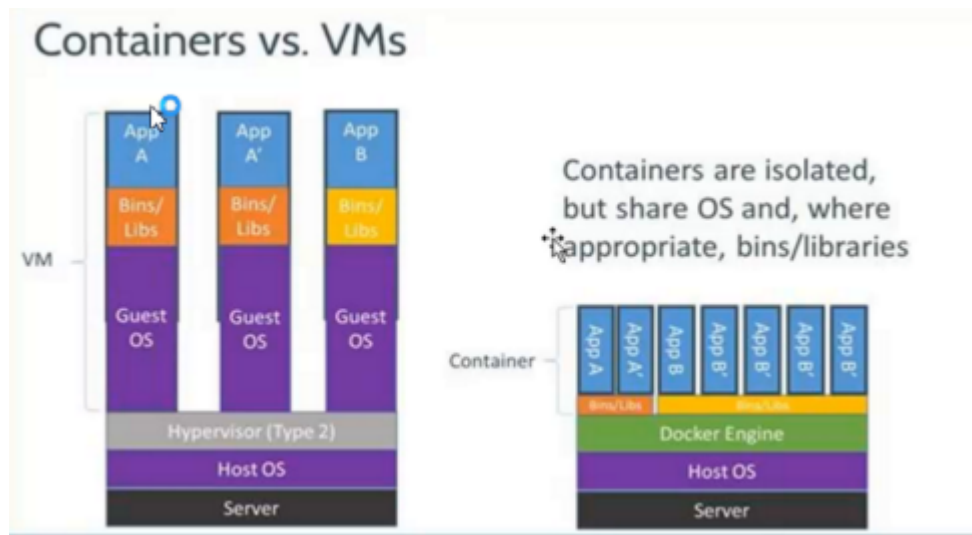
由于虚拟机存在这些缺点，Linux 发展出了另一种虚拟化技术：Linux 容器（Linux Containers，缩写为 LXC）。

容器与虚拟机有所不同，虚拟机通过虚拟软件中间层将一台或者多台独立的虚拟机器运行在物理硬件之上。而容器则是直接运行在操作系统内核之上，是进程级别的，并对进程进行了隔离，**而不是模拟一个完整的操作系统**。因此，容器虚拟化也被称为“操作系统级虚拟化”，容器技术可以将软件需要的环境配置都打包到一个隔离的容器中。让多个独立的容器高效且轻量的运行在同一台宿主机上。而Docker就是为了实现这一切而生的。

1.3.2 容器与虚拟机比较

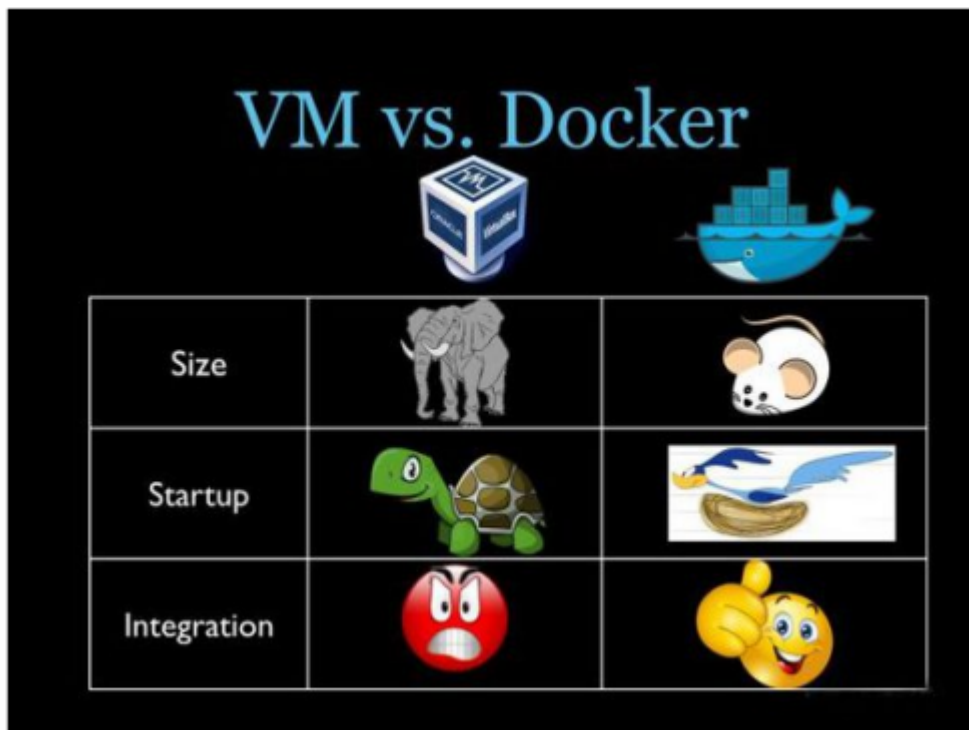
由于容器是进程级别的，相比虚拟机有很多优势。

(1) 本质上的区别：



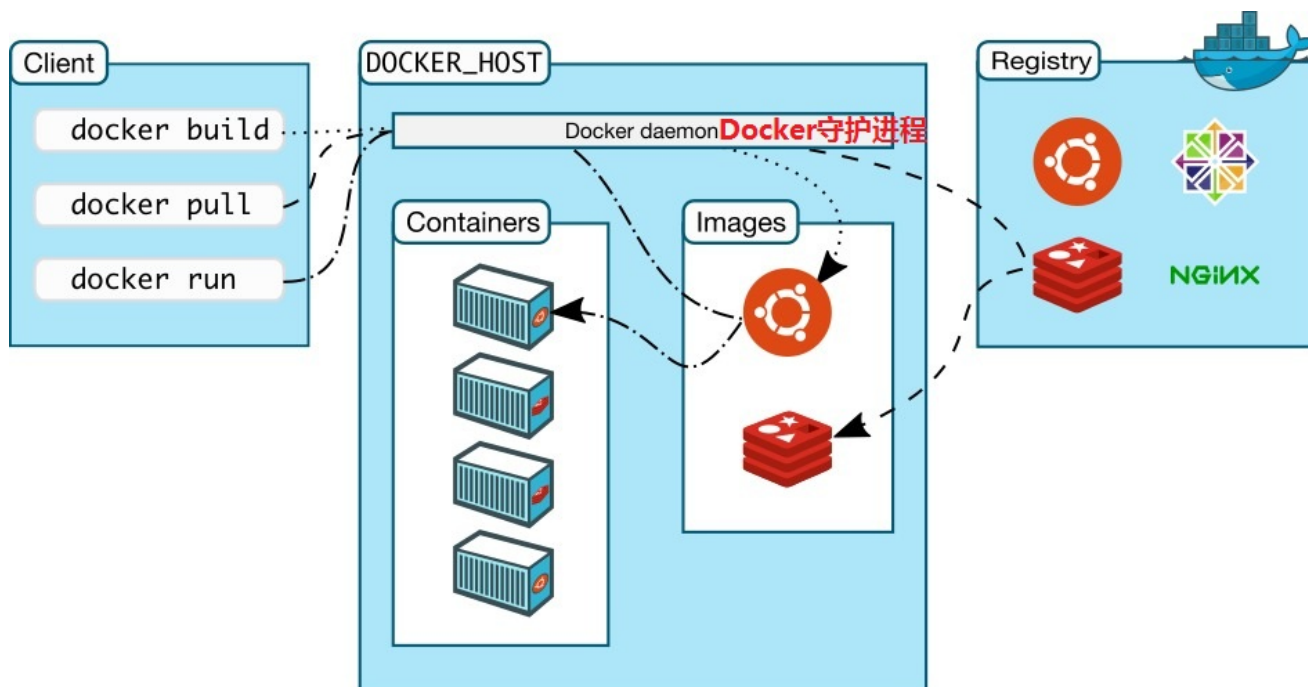
(2) 使用上的区别：

容器：体积小、启动快（秒级）、资源占用少（只占用需要的资源）、好评如潮



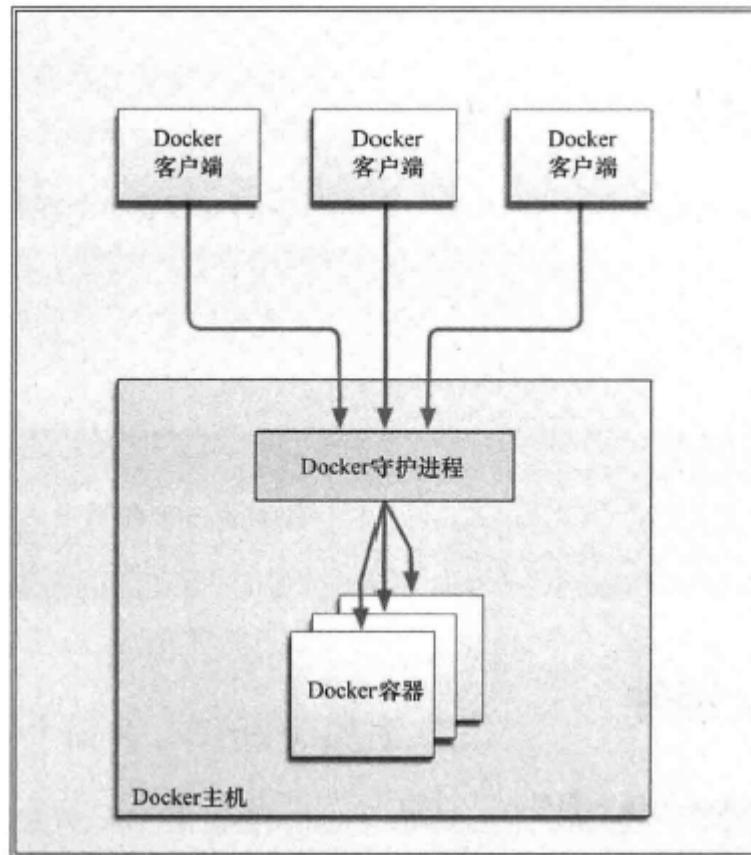
虚拟机已死，容器才是未来。

1.4 分析 Docker 容器架构



1.4.1 Docker 客户端和服务端

Docker是一个客户端-服务器（C/S）架构程序。Docker客户端只需要向Docker服务器或者守护进程发出请求，服务器或者守护进程将完成所有工作并返回结果。Docker提供了一个命令行工具和一整套RESTful API。你可以在同一台宿主机上运行Docker守护进程和客户端，也可以从本地的Docker客户端连接到运行在另一台宿主机上的远程Docker守护进程。



1.4.2 Docker 镜像(Image)

镜像 (Image) 是Docker中的一个模板。通过 Docker镜像 来创建 Docker容器，一个镜像可以创建出多个容器。镜像是由一系列指令一步一步构建出来。例如：

- 1 添加一个文件；
- 2
- 3 执行一个命令；
- 4
- 5 打开一个窗口。

镜像与容器的关系类似于Java中类与对象的关系。镜像体积很小，非常“便携”，易于分享、存储和更新。

Docker	Java
镜像	类
容器	对象

```
class Emp{} //镜像：
```

```
Emp e1 = new Emp(); //容器1
```

```
Emp e2 = new Emp(); //容器2
```

1.4.3 Docker 容器(Container)

容器 (Container) 是基于镜像创建的运行实例，一个容器中可以运行一个或多个应用程序 (jdk+开发的java应用程序)。

Docker 可以帮助你构建和部署容器，你只需要把自己的应用程序或者服务打包放进容器即可。

我们可以认为，镜像是Docker生命周期中的构建或者打包阶段，而容器则是启动或者执行阶段。

可以理解容器中有包含：一个精简版的Linux环境 + 要运行的应用程序

1.4.4 Docker 仓库(repository)

- 仓库 (Repository) 是集中存放镜像文件的场所。
- 有时候会把仓库 (Repository) 和仓库注册服务器 (Registry) 混为一谈，但并不严格区分。实际上，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)。
- 仓库分为公有仓库 (Public) 和私有仓库 (Private) 两种。
 - Docker公司运营的公共仓库叫做 Docker Hub (<https://hub.docker.com/>)，存放了数量庞大的镜像供用户下载。用户可以在Docker Hub注册账号，分享并保存自己的镜像。(说明：在Docker Hub下载镜像巨慢)
 - 国内的公有仓库包括阿里云、网易云等，可以提供大陆用户更稳定快速的访问。
- 当用户创建了自己的镜像之后就可以使用 push 命令将它上传到公有或者私有仓库，这样下次在另外一台机器上使用这个镜像时候，只需要从仓库上 pull 下来就可以了。

Docker 仓库的概念跟 Git 类似，注册服务器可以理解为 GitHub 这样的托管服务。



第二章 VMware和CentOS7安装

2.1 Docker 安装环境说明

- Docker官方建议在Ubuntu中安装，因为 Docker 是基于Ubuntu发布的，而且一般 Docker 出现的问题 Ubuntu是最先更新或者打补丁的。在很多版本的 CentOS 中是不支持更新最新的一些补丁包的。
- 由于很多公司的环境都使用的是 CentOS，因此这里我们将 Docker 安装到 CentOS 上。

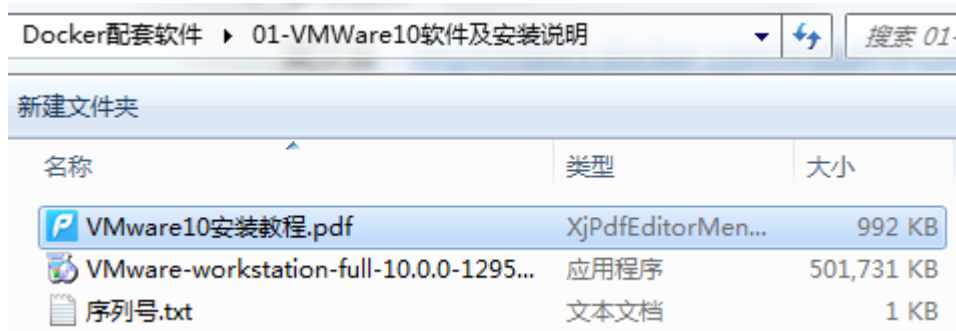
- **注意：**
 - Docker 要求 CentOS 系统的内核版本高于 3.10，查看此前提条件来验证你的 CentOS 版本是否支持 Docker。
 - 需要安装 64位 CentOS7.x 以上的版本，在 CentOS6.x 的版本中，安装前需要安装其他很多的环境而且 Docker 很多补丁不支持更新。
- 参考资料：
 - 英文版：<https://docs.docker.com/install/linux/docker-ce/centos/>
 - 中文版：<https://docs.docker-cn.com/engine/installation/linux/docker-ce/centos/>

2.2 在VMware Workstation10中安装CentOS7

我们采用的是CentOS7进行讲解，所以我们要在本机安装CentOS7

建议电脑内存8G+，8G也会有点吃力噢

- 安装 VMware Workstation10虚拟软件
 - 安装资源位于：Docker配套软件/ 01-VMWare10软件及安装说明/



- 安装 CentOS7
 - 安装资源位于：Docker配套软件/02-CentOS7安装教程/

第三章 Docker 安装卸载与启停

3.1 查看当前系统的内核版本

- 查看当前系统的内核版本是否高于 3.10

```
1 [root@mengxuegu /]# uname -r
2 3.10.0-693.el7.x86_64
```

3.2 安装 Docker 服务

- 使用镜像仓库进行安装，采用 yum 命令在线安装(即电脑需要联网)

root 用户运行以下命令：

1. 卸载旧版本：（如果安装过旧版本的话）

Docker 的早期版本称为 `docker` 或 `docker-engine`。如果安装了这些版本，请卸载它们及关联的依赖资源。

```
1 yum remove docker docker-common docker-selinux docker-engine
```

```
[root@192 ~]# yum remove docker \
> docker-common \
> docker-selinux \
> docker-engine
已加载插件：fastestmirror, langpacks
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
```

2. 安装所需的软件包

`yum-utils` 提供了 `yum-config-manager` 实用程序，并且 `devicemapper` 存储驱动需要 `device-mapper-persistent-data` 和 `lvm2`。

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

```
[root@192 ~]# yum install -y yum-utils device-mapper-persistent-data lvm2
已加载插件：fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirrors.aliyun.com
 * extras: ap.stykers.moe
 * updates: mirrors.163.com
正在解决依赖关系
--> 正在检查事务
---> 软件包 device-mapper-persistent-data.x86_64.0.0.7.0-0.1.rc6.el7 将被 升级
---> 软件包 device-mapper-persistent-data.x86_64.0.0.7.3-3.el7 将被 更新
---> 软件包 lvm2.x86_64.7.2.02.171-8.el7 将被 升级
---> 软件包 lvm2.x86_64.7.2.02.180-10.el7_6.3 将被 更新
--> 正在处理依赖关系 lvm2-libs = 7:2.02.180-10.el7_6.3，它被软件包 7:lvm2-2.02.180-10.el7_6.3.x86_64 需要
--> 正在处理依赖关系 libdevmapper.so.1.02(DM_1_02_141) (64bit)，它被软件包 7:lvm2-2.02.180-10.el7_6.3.x86_64 需要
---> 软件包 yum-utils.noarch.0.1.1.31-42.el7 将被 升级
```

3. 设置Docker的镜像仓库

```
1 yum-config-manager \
2 --add-repo \
3 https://download.docker.com/linux/centos/docker-ce.repo
```

上面处可能会报错（原因是国内访问不到docker官方镜像的缘故）


```
Could not fetch/save url https://download.docker.com/linux/centos/docker-ce.repo to file  
/etc/yum.repos.d/docker-ce.repo:
```

解决：使用以下方式：阿里源访问

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

```
device-mapper-libs.x86_64 7:1.02.149-10.el7_6.3  
lvm2-libs.x86_64 7:2.02.180-10.el7_6.3
```

完毕！

```
[root@192 ~]# yum-config-manager \  
>     --add-repo \  
>     https://download.docker.com/linux/centos/docker-ce.repo
```

已加载插件：fastestmirror, langpacks

adding repo from: https://download.docker.com/linux/centos/docker-ce.repo

grabbing file https://download.docker.com/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo

repo saved to /etc/yum.repos.d/docker-ce.repo

4. 安装最新版本的 Docker CE

```
1 yum install docker-ce
```

```
[root@192 ~]# yum install docker-ce
```

已加载插件：fastestmirror, langpacks

docker-ce-stable

	3.5 kB	00:00
--	--------	-------

(1/2): docker-ce-stable/x86_64/updateinfo

	55 B	00:01
--	------	-------

(2/2): docker-ce-stable/x86_64/primary_db

	25 kB	00:01
--	-------	-------

Loading mirror speeds from cached hostfile

* base: mirrors.aliyun.com

* extras: ap.stykers.moe

* updates: mirrors.163.com

正在解决依赖关系

--> 正在检查事务

---> 软件包 docker-ce.x86_64.3.18.09.3-3.el7 将被 安装

--> 正在处理依赖关系 container-selinux >= 2.9, 它被软件包 3:docker-ce-18.09.3-3.

- 安装中出现下面提示, 输入 `y` 然后回车

```
selinux-policy          noarch 3.13.1-229.el7_6.9 updates 483 k
selinux-policy-targeted noarch 3.13.1-229.el7_6.9 updates 6.9 M
setools-libs            x86_64 3.3.8-4.el7 base 620 k
```

事务概要

安装 1 软件包 (+ 3 依赖软件包)
升级 (11 依赖软件包)

总计: 65 M

总下载量: 65 M

Is this ok [y/d/N]:

输入 y

- 如用的Docker国外网下载比较慢，当出现下面提示，输入 然后回车

```
(12/14): setools-libs-3.3.8-4.el7.x86_64.rpm | 620 KB 00:00
(13/14): containerd.io-1.2.4-3.1.el7.x86_64.rpm | 22 MB 16:59
(14/14): docker-ce-cli-18.09.3-3.el7.x86_64.rpm | 14 MB 09:00
```

总计 44 kB/s | 65 MB 25:00

从 https://download.docker.com/linux/centos/gpg 检索密钥

导入 GPG key 0x621E9F35:

用户ID : "Docker Release (CE rpm) <docker@docker.com>"

指纹 : 060a 61c5 1b55 8a7f 742b 77aa c52f eb6b 621e 9f35

来自 : https://download.docker.com/linux/centos/gpg

是否继续? [y/N]:

5. 安装完成

作为依赖被升级:

```
libselinux.x86_64 0:2.5-14.1.el7
libselinux-python.x86_64 0:2.5-14.1.el7
libselinux-utils.x86_64 0:2.5-14.1.el7
libsemanage.x86_64 0:2.5-14.el7
libsemanage-python.x86_64 0:2.5-14.el7
libsepol.x86_64 0:2.5-10.el7
policycoreutils.x86_64 0:2.5-29.el7_6.1
policycoreutils-python.x86_64 0:2.5-29.el7_6.1
selinux-policy.noarch 0:3.13.1-229.el7_6.9
selinux-policy-targeted.noarch 0:3.13.1-229.el7_6.9
setools-libs.x86_64 0:3.3.8-4.el7
```

完毕!

[root@192 ~]#

如上面安装不成功，则按第3.3节卸载再按上面重新安装

3.3 卸载 Docker 服务

1. 卸载 Docker 软件包

```
1 [root@mengxuegu /]# yum remove docker-ce
```

2. 删除镜像/容器等

```
1 [root@mengxuegu /]# rm -rf /var/lib/docker
```

3.4 启动与停止Docker服务

上面安装只是安装好,但是没有启动Docker服务。

- `systemctl` 命令是系统服务管理器指令,它是 `service` 和 `chkconfig` 两个命令组合。
 - 启动docker: `systemctl start docker`
 - 停止docker: `systemctl stop docker`
 - 重启docker: `systemctl restart docker`
 - 查看docker状态: `systemctl status docker`
 - 开机自动启动docker: `systemctl enable docker`

```
[root@192 ~]# systemctl status docker
```

● docker.service - Docker Application Container Engine

Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)

Active: **active (running)** since 四 2019-03-14 21:30:53 CST; 24s ago

Docs: <https://docs.docker.com>

说明已启动

Main PID: 44172 (dockerd)

Memory: 32.7M

CGroup: /system.slice/docker.service

└─44172 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

```
3月 14 21:30:52 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:52.689273030+08:00" level=
3月 14 21:30:52 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:52.689348673+08:00" level=
3月 14 21:30:52 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:52.763615884+08:00" level=
3月 14 21:30:52 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:52.765951104+08:00" level=
3月 14 21:30:53 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:53.216679150+08:00" level=
3月 14 21:30:53 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:53.509845805+08:00" level=
3月 14 21:30:53 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:53.578386555+08:00" level=
3月 14 21:30:53 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:53.579575439+08:00" level=
3月 14 21:30:53 192.168.10.12 systemd[1]: Started Docker Application Container Engine.
3月 14 21:30:53 192.168.10.12 dockerd[44172]: time="2019-03-14T21:30:53.604336684+08:00" level=
Hint: Some lines were ellipsized, use -l to show in full.
```

```
[root@192 ~]#
```

3.5 Docker 版本查看

- 查看当前安装的 Docker 版本

```
1 [root@mengxuegu /]# docker version
```

```
[root@192 ~]# docker version
Client:
Version:      18.09.3
API version:  1.39
Go version:   go1.10.8
Git commit:   774a1f4
Built:        Thu Feb 28 06:33:21 2019
OS/Arch:      linux/amd64
Experimental: false
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
[root@192 ~]#
```

3.6 Docker 帮助命令

- 查看 docker 帮助命令: `docker --help`

```
[root@192 ~]# docker --help
```

选项, 可省略

Usage: docker [OPTIONS] COMMAND → docker常用命令

A self-sufficient runtime for containers

Options:

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

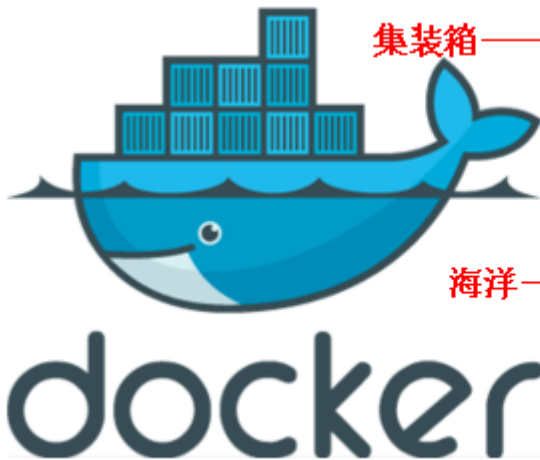
Management Commands:

<code>builder</code>	Manage builds
<code>config</code>	Manage Docker configs
<code>container</code>	Manage containers
<code>engine</code>	Manage the docker engine
<code>image</code>	Manage images
<code>network</code>	Manage networks
<code>node</code>	Manage Swarm nodes

- 查看 docker 概要信息: `docker info`

```
[root@192 ~]# docker info
Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
Images: 0
Server Version: 18.09.3
Storage Driver: overlay2 存储驱动
  Backing Filesystem: xfs
  Supports d_type: true
  Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
  Volume: local
  Network: bridge host macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splu
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: e6b3f5632f50dbc4e9cb6288d911bf4f5e95b18e
runc version: 6635b4f0c6af3810594d2770f662f34ddc15b40d
init version: fec3683
Security Options:
  seccomp
  Profile: default
Kernel Version: 3.10.0-693.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
CPUs: 4
Total Memory: 1.789GiB
Name: 192.168.10.12
ID: FZXA:LCAA:EBAI:XPGM:TXYS:U5CN:YYEW:XR42:RJAX:B2RU:2EON:URLL
Docker Root Dir: /var/lib/docker 镜像与容器存储位置
Debug Mode (client): false
```

第四章 Docker 镜像操作



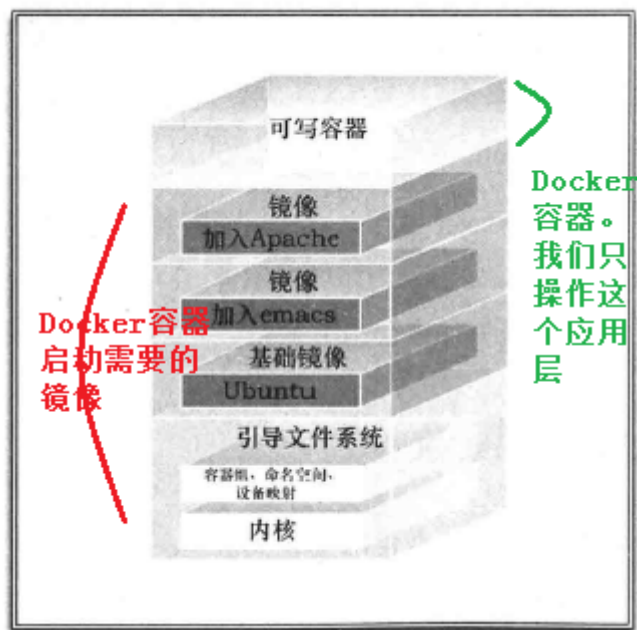
集装箱——》容器实例，通过镜像模板生成的

鲸鱼——》Docker

海洋——》宿主机，我们采用的是CentOS7

4.1 什么是Docker镜像

- Docker镜像是由文件系统叠加而成（是一种文件的存储形式）。最底端是一个文件引导系统，即bootfs，这很像典型的Linux/Unix的引导文件系统。Docker用户几乎永远不会和引导系统有什么交互。实际上，当一个容器启动后，它将会被移动到内存中，而引导文件系统则会被卸载，以留出更多的内存供磁盘镜像使用。Docker容器启动是需要的一些文件，而这些文件就可以称为Docker镜像。



4.2 列出镜像

- docker官网镜像搜索：<https://hub.docker.com/>
- 列出docker下的已安装所有镜像：

```
1 [root@mengxuegu /]# docker images
```



```
[root@192 ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mysql	5.6	69056560eb44	13 days ago	256MB
mysql	latest	91dadee7afee	13 days ago	477MB

- REPOSITORY：镜像所在仓库的名称
- TAG：镜像标签（一般是版本号）
- IMAGE ID：镜像ID
- CREATED：镜像的创建日期
- SIZE：镜像大小

- 只显示镜像ID

```
1 [root@mengxuegu ~]# docker images -q
```

```
[root@192 ~]# docker images -q
```

```
69056560eb44
```

```
91dadee7afee
```

```
[root@192 ~]#
```

- 这些镜像都是存储在Docker宿主机的 `/var/lib/docker` 目录下

```
[root@192 etc]# ll /var/lib/docker
```

```
总用量 0
```

```
drwx-----. 2 root root 24 3月 14 21:30 builder
drwx-----. 4 root root 92 3月 14 21:30 buildkit
drwx-----. 2 root root 6 3月 14 21:30 containers
drwx-----. 3 root root 22 3月 14 21:30 image
drwxr-xr-x. 3 root root 19 3月 14 21:30 network
drwx-----. 3 root root 40 3月 18 11:14 overlay2
drwx-----. 4 root root 32 3月 14 21:30 plugins
drwx-----. 2 root root 6 3月 18 11:14 runtimes
drwx-----. 2 root root 6 3月 14 21:30 swarm
drwx-----. 2 root root 6 3月 18 11:14 tmp
drwx-----. 2 root root 6 3月 14 21:30 trust
drwx-----. 2 root root 25 3月 14 21:30 volumes
```

- 为了区分同一个仓库下的不同镜像，Docker提供了一种称为标签（TAG）的功能。每个镜像都带有一个标签（TAG），例如10.2.1、latest等等。这种机制使得同一个仓库中可以存储多个镜像。--- 版本号
- 我们可以使用仓库名后面加上一个冒号和标签名（REPOSITORY:TAG）来指定该仓库中的某一具体的镜像，如果未指定镜像的标签，将下载latest最新版本，例如：只写了centos，docker将默认使用centos:latest镜像。

4.3 搜索镜像

- 如果你需要从网络中查找需要的镜像，可以通过以下命令搜索

```
1 docker search [OPTIONS] 镜像名称
```

如: `docker search centos`

```
[root@192 etc]# docker search centos
NAME 镜像名称 DESCRIPTION 描述 STARS 关注度 OFFICIAL 官方 AUTOMATED
centos The official build of CentOS. 5256 [OK]
ansible/centos7-ansible Ansible on Centos7 121 [OK]
jdeath/centos-ssh CentOS-6 6.10 x86_64 / CentOS-7 7.5.1804 x86_64 108 [OK]
consol/centos-xfce-vnc Centos container with "headless" VNC session... 83 [OK]
```

- NAME：仓库名称
 - DESCRIPTION：镜像描述
 - STARS：关注度，反应一个镜像的受欢迎程度
 - OFFICIAL：是否官方
 - AUTOMATED：自动构建，表示该镜像由Docker Hub自动构建流程创建的
- OPTIONS 选项说明：
 - `-s` 列出关注数大于指定值的镜像

```
1 [root@mengxuegu /]# docker search -s 100 centos
```

- `--no-trunc` 显示完整的镜像描述DESCRIPTION

```
1 [root@mengxuegu /]# docker search --no-trunc centos
```

4.4拉取镜像

4.4.1从Docker Hub拉取镜像

- 命令：

```
1 [root@mengxuegu /]# docker pull 镜像名:标签名
```

例如：我们拉取 mysql 5.6版本的镜像

```
1 [root@mengxuegu /]# docker pull mysql:5.6
```

- 注意：如果出现下载失败情况：网络连接超时

```
1 [root@mengxuegu /]# docker pull mysql:5.6
2 Trying to pull repository docker.io/library/mysql ...
3 ## 出现下面情况，说明下载连接超时
4 Get https://registry-1.docker.io/v2/: net/http: request canceled
   (Client.Timeout exceeded while awaiting headers)
```

- 成功下载情况：

```
1 [root@mengxuegu /]# docker pull mysql:5.6
2 5.6: Pulling from library/mysql
3 f7e2b70d04ae: Pull complete
4 df7f6307ff0a: Pull complete
```

```
5 e29ed02b1013: Pull complete
6 9cb929db392c: Pull complete
7 42cc77b24286: Pull complete
8 b7493809599f: Pull complete
9 9e72fa203c2b: Pull complete
10 e4a5e4487a94: Pull complete
11 165ca9a539aa: Pull complete
12 81140eaaa67e: Pull complete
13 19021337b46f: Pull complete
14 Digest:
   sha256:36cad5daaae69fbcc15dd33b9f25f35c41bbe7e06cb7df5e14d8b966fb26c1b4
15 Status: Downloaded newer image for mysql:5.6
```

4.4.2 配置国内镜像加速器

国情的原因，目前国内访问 Docker HUB 官方的相关镜像下载比较慢，可以使用国内的一些镜像加速器，镜像保持和官方一致，关键是速度快，推荐使用。镜像加速器其实是把官方的库文件整个拖到自己的服务器上做镜像，并定时与官方做同步。

- 第一种：ustc

ustc是老牌的linux镜像服务提供者了，还在遥远的ubuntu 5.04版本的时候就在用。ustc的docker镜像加速器速度很快。ustc docker mirror的优势之一就是不需要注册，是真正的公共服务。<https://lug.ustc.edu.cn/wiki/mirrors/help/docker>

- 步骤：

1. 通过修改daemon配置文件/etc/docker/daemon.json来使用加速器,如果不存在则手动创建

```
1 vim /etc/docker/daemon.json
```

2. 打开文件后，按 **i** 字母后插入状态，在该文件中输入如下内容：

```
1 {
2   "registry-mirrors": ["https://docker.mirrors.ustc.edu.cn"]
3 }
```

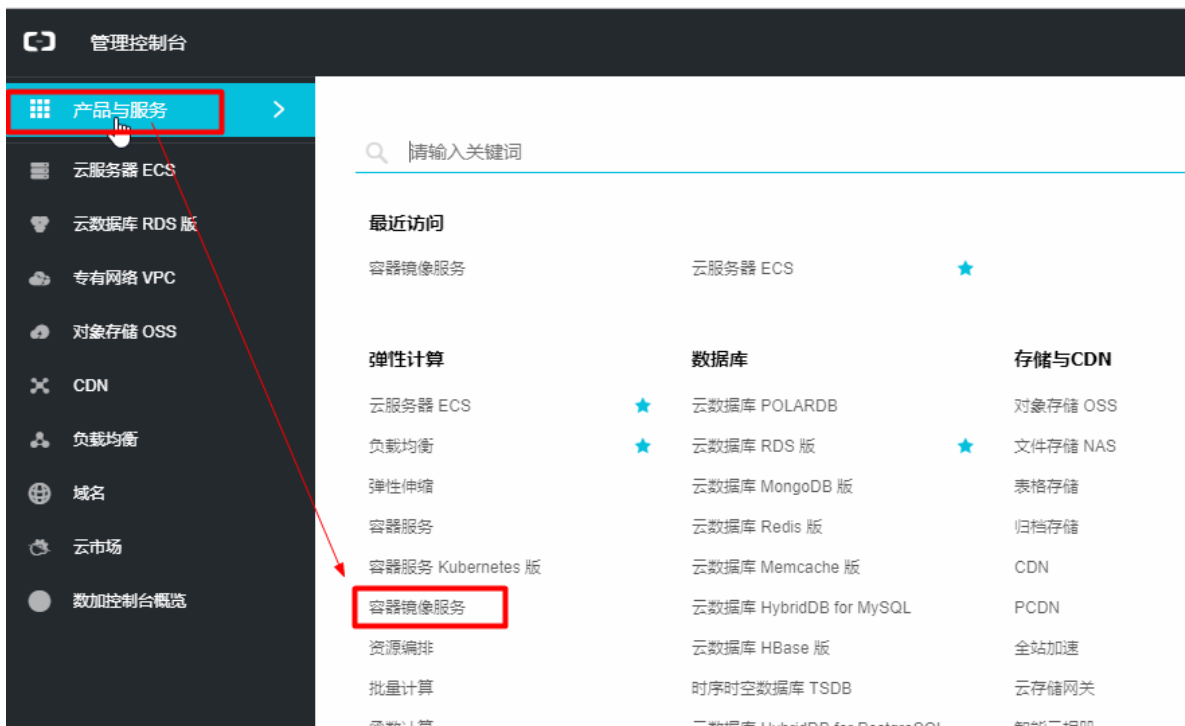
3. 注意：一定要重启docker服务，如果重启docker后无法加速，那就重新启动CentOS

```
1 # 重载此配置文件
2 systemctl daemon-reload
3 # 重启 docker
4 systemctl restart docker
```

- 第二种：阿里云镜像, 需要注册帐号

1. 注册并登陆阿里云<https://www.aliyun.com/>，进入“管理控制台”后，在如下图找到 容器镜像服务

应用 MySQL Vue.js ky 虚拟机中的服务 本地应用 系统 Docker 淘宝



2. 通过修改daemon配置文件/etc/docker/daemon.json来使用加速器,如果不存在则手动创建

```
1 vim /etc/docker/daemon.json
```

3. 文件加入以下内容

```
1 {  
2   "registry-mirrors": ["https://w57n2hu2.mirror.aliyuncs.com"]  
3 }
```

4. 一定要重启docker服务, 如果重启docker后无法加速, 可以重新启动CentOS

```
1 # 重载此配置文件
2 systemctl daemon-reload
3 # 重启 docker
4 systemctl restart docker
```

- 以上两种方式 二选其一
- 再通过 `docker pull` 命令下载镜像：速度杠杠的

```
1 docker pull mysql
```

4.5 删除镜像

- 删除某一个镜像

```
1 docker rmi 镜像ID
```

```
[root@192 /]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mysql                5.6                69056560eb44       13 days ago        256MB
mysql                latest             91dadee7afee       13 days ago        477MB
[root@192 /]# docker rmi 91dadee7afee
Untagged: mysql:latest
Untagged: mysql@sha256:4589ba2850b93d103e60011fe528fc56230516c1efb4d3494c33ff499505356f
Deleted: sha256:91dadee7afeebe274c51104d572ab6a2dc0ae97473f71afc57fbfd48c0ceb8aa
Deleted: sha256:82eee4082d8cbb961ff33564cc655db9b4c73d118e9859bfec26a2c5e0714c
Deleted: sha256:cd9489e5896812af2e519e18470b6efe6ad19e556cfe38df1d9bab7706bdc94
Deleted: sha256:a7205b2f8740073d175dd4ee3c69d1fe44f2d31529fb947e6ded8899f0f2b52c
Deleted: sha256:f890db1a51f76670d9752256c6a897af077e608dc83711e756ddba7f22e9150b
Deleted: sha256:af3357c7a40759832f8e1188d7e73e5e1c2dfb503397ee2b8f61133bfce13b5b
Deleted: sha256:6780729a47b08db93437f74d430a865d4d0728707d134f540217a9ba773c4376
Deleted: sha256:1248c21c6b437a9b7528c6a61acc40589b9b373094a609552c91e719cb40df21
[root@192 /]#
```

- 删除所有镜像 (是 `` 反单引号)

```
1 docker rmi `docker images -q`
```

- 其中 `docker images -q` 获取所有镜像id

第五章 Docker 容器操作

5.1 查看容器

- 查看正在运行容器：

```
1 docker ps
```

```
[root@192 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

列说明：

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
容器 ID	镜像	初始命令	创建日期	容器状态	端口号	容器名字

- 查看所有的容器（启动与未启动的容器）：

```
1 docker ps -a
```

```
[root@localhost ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f41cd9046524	tomcat:8	"catalina.sh run"	5 hours ago	Exited (130) 5 hours ago		mytomcat

- 查看最后一次运行的容器：

```
1 docker ps -l
```

```
[root@localhost ~]# docker run -di --name=itmak_mysql -p 3306:3306 mysql:5.6
```

最近一次启动的容器

```
[root@localhost ~]# docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
2182e67a5f31	mysql:5.6	"docker-entrypoint..."	28 seconds ago	Exited (1) 12 seconds ago

28秒前启动

- 查看停止的容器

```
1 docker ps -f status=exited
```

```
[root@localhost ~]# docker ps -f status=exited
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
2182e67a5f31	mysql:5.6	"docker-entrypoint..."	5 minutes ago	Exited (1) 5 minutes ago
f41cd9046524	tomcat:8	"catalina.sh run"	5 hours ago	Exited (130) 5 hours ago

5.2 创建与启动容器

注意：必须先有镜像，再有容器，下面以CentOS镜像演示

```
1 docker pull centos:7
```

5.2.1 创建容器命令

```
1 docker run [OPTIONS] 镜像名:标签名
```

- 创建容器 [OPTIONS] 常用的参数说明：

- i 表示交互式运行容器（就是创建容器后，马上会启动容器，并进入容器），通常与 -t 同时使用。

- `-t` 启动后会进入其容器命令行, 通常与 `-i` 同时使用; 加入 `-it` 两个参数后, 容器创建就能登录进去。即分配一个伪终端。
- `--name` 为创建的容器指定一个名称。
- `-d` 创建一个守护式容器在后台运行, 并返回容器ID;
这样创建容器后不会自动登录容器, 如果加 `-i` 参数, 创建后就会运行容器。
- `-v` 表示目录映射, 格式为: `-p 宿主机目录:容器目录`
注意: 最好做目录映射, 在宿主机上做修改, 然后共享到容器上。
- `-p` 表示端口映射, 格式为: `-p 宿主机端口:容器端口`

5.2.2 交互式容器

说明: 就是创建容器后, 马上会启动容器, 并进入容器

1. 创建一个交互式容器并取名为 `mycentos`, (`/bin/bash` 是linux中的命令解析器, 会进入到容器里面命令行)

```
1 docker run -it --name mycentos centos:7 /bin/bash
```

```
[root@192 ~]# docker run -it --name=mycentos centos:7 /bin/bash
[root@12b974bb4ac2 ~]#
```

表示已经进入容器中的命令行窗口

上图显示, 已经进入容器中 (上面主机名变成了容器实例编号)

2. 新开一个 shell 窗口, 这时我们通过 `docker ps` 命令查看, 发现可以看到启动的容器, 状态为 Up 启动状态

```
[root@192 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
12b974bb4ac2	centos:7	"/bin/bash"	2 minutes ago	Up 2 minutes

不能在容器里面执行 已经启动

3. 新开一个 shell 窗口, 再创建一个 `mycentos1` 容器 (注意: `docker` 命令是不能在容器里面使用)

```
1 docker run -it --name mycentos1 centos:7 /bin/bash
```

```
[root@192 ~]# docker run -it --name=mycentos1 centos:7 /bin/bash
[root@b5eeda5e8e2a ~]#
```

4. 再通过 `docker ps` 命令查看, 发现有2个已经启动的容器

```
[root@192 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
b5eeda5e8e2a	centos:7	"/bin/bash"	About a minute ago	Up About a minute
12b974bb4ac2	centos:7	"/bin/bash"	5 minutes ago	Up 5 minutes

总结: 同一个镜像, 可运行多个容器

5.3 退出容器

1. 退出并停止当前容器，**注意在容器内部的命令行执行**

```
1 exit
```

```
[root@192 ~]# docker run -it --name=mycentos centos:7 /bin/bash
[root@12b974bb4ac2 ~]# exit
exit
[root@192 ~]#
```

- 用 `docker ps -a` 命令，容器的状态变成 `Exited` 退出状态

```
[root@192 ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
b5eeda5e8e2a	centos:7	"/bin/bash"	6 minutes ago	Up 6 minutes
12b974bb4ac2	centos:7	"/bin/bash"	10 minutes ago	已退出 Exited (0) 6 seconds ago

2. 退出不停止当前容器

按键盘：`Ctrl + p + q`，按一次一行，则多按几次，一般是连两次即可。

```
[root@192 ~]# docker run -it --name=mycentos1 centos:7 /bin/bash
[root@b5eeda5e8e2a ~]#
[root@192 ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
b5eeda5e8e2a	centos:7	"/bin/bash"	17 minutes ago	Up 17 minutes
12b974bb4ac2	centos:7	"/bin/bash"	22 minutes ago	Exited (0) 11 minutes ago

按Ctrl+q+p退出后，仍然还是运行中

5.3 启动与停止容器

- 启动已运行过的容器

```
1 docker start 容器名称|容器id
```

如：`docker start mycentos`

```
[root@192 ~]# docker start mycentos
mycentos
[root@192 ~]#
```

- 启动所有运行过的容器（注意：反单引号`），`docker ps -a -q` 是查询所有运行过的容器ID

```
1 docker start `docker ps -a -q`
```

```
[root@192 ~]# docker ps -a -q  
0c20b3037f46  
b5eeda5e8e2a
```

```
[root@192 ~]# docker start `docker ps -a -q`  
0c20b3037f46  
b5eeda5e8e2a
```

- 停止正在运行的容器（正常停止）

```
1 docker stop 容器名称|容器id
```

如：`docker stop mycentos`

```
[root@192 ~]# docker stop mycentos  
mycentos  
[root@192 ~]#
```

- 强制停止正在运行的容器（一般不用此，除非卡了）

```
1 docker kill 容器名称|容器id
```

如：`docker kill mycentos1`

```
[root@192 ~]# docker kill mycentos1  
mycentos1  
[root@192 ~]#
```

- 停止所有在运行的容器

```
1 docker stop `docker ps -a -q`
```

```
[root@192 ~]# docker stop `docker ps -a -q`  
0c20b3037f46  
b5eeda5e8e2a
```

5.4 创建守护式容器

如果对于一个需要长期运行的容器来说，我们可以创建一个守护式容器（后台运行的容器）。

- 创建（-d）并运行（-i）守护式容器命令如下（容器名称不能重复）：

```
1 docker run -id --name mycentos2 centos:7
```

```
[root@192 ~]# docker run -id --name=mycentos2 centos:7  
616c4a69e950b07a1370f22db481c2da0bbd87f0a9e49a296d6a09e2e434520  
[root@192 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
616c4a69e950	centos:7	"/bin/bash"	13 seconds ago	Up 12 seconds		mycentos2

5.5 登录容器

登录已经启动的容器方式：

- 使用 `docker exec` 进入容器中打开命令行终端

```
1 docker exec -it 容器名称|容器id /bin/bash
```

如：`docker exec -it mycentos2 /bin/bash`

```
[root@192 ~]# docker exec -it mycentos2 /bin/bash
[root@0c20b3037f46 /]#
```

- `exit` 针对通过 `docker exec` 进入的容器，只退出但不停止容器

```
1 [root@0c20b3037f46 /]# exit
```

```
[root@192 ~]# docker exec -it mycentos2 /bin/bash
[root@0c20b3037f46 /]# exit
exit
[root@192 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0c20b3037f46	centos:7	"/bin/bash"	About an hour ago	Up 2 minutes		mycentos2

exit 仅退出，并没有停止

注意：交互式容器通过 `docker exec` 进入容器后，使用 `exit` 也一样的只退出但不停止容器

5.6 拷贝宿主机与容器中的文件

如果我们需要将宿主机文件拷贝到容器内可以使用 `docker cp` 命令，也可以将文件从容器内拷贝到宿主机

- 将宿主机文件拷贝到容器内

```
1 docker cp 要拷贝的宿主机文件或目录 容器名称:容器文件或目录
```

1. 在宿主机上创建一个mengxuegu文件并写入内容

```
1 cd / #切换到根目录/下
2 echo 123456 > mengxuegu #创建并写入内容 mengxuegu 文件中
3 ls #查看是否已创建
4 cat mengxuegu #查看文件内容
```

```
[root@192 ~]# cd /
[root@192 /]# echo 123456 > mengxuegu
[root@192 /]# ls
bin boot dev etc home lib lib64 media mengxuegu
[root@192 /]# cat mengxuegu
123456
[root@192 /]#
```

2. 将mengxuegu文件拷贝进 mycentos2 容器中的 `/opt` 目录下(mycentos2要是UP启动状态)

```
1 docker cp mengxuegu mycentos2:/opt
```

3. 登录 mycentos2 容器，查看/opt目录下是否有 mengxuegu 文件

```
1 [root@192 /]# docker exec -it mycentos2 /bin/bash
2 [root@0c20b3037f46 /]# ls /opt/
3 mengxuegu
4 [root@0c20b3037f46 /]# cat /opt/mengxuegu
5 123456
```

```
[root@192 ~]# cd /
[root@192 /]# echo 123456 > mengxuegu
[root@192 /]# ls
bin boot dev etc home lib lib64 media mengxuegu mnt opt proc root run sbin srv sys tmp usr var
[root@192 /]# cat mengxuegu
123456
[root@192 /]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
0c20b3037f46       centos:7            "/bin/bash"         About an hour ago   Up 35 minutes      0.0.0.0:22->22      mycentos2
b5eeda5e8e2a       centos:7            "/bin/bash"         2 hours ago        Up 18 minutes      0.0.0.0:22->22      mycentos1
```

要是启动状态

拷贝mengxuegu文件进容器的/opt目录

登录 mycentos2 容器

查看容器中的文件内容一致

• 从容器内文件拷贝到宿主机

```
1 docker cp 容器名称:要拷贝的容器文件或目录 宿主机文件或目录
```

1. 向 mycentos2 容器中的 mengxuegu 文件进行追加内容abc (要登录进容器中)

```
1 cd / #切换到根目录/下
2 echo 123456 > mengxuegu #创建并写入内容 mengxuegu 文件中
3 ls #查看是否已创建
4 cat mengxuegu #查看文件内容
```

2. 需要使用 exit 先退出 docker 容器命令行，回到宿主机

```
[root@0c20b3037f46 /]# exit
exit
[root@192 /]#
```

3. 从容器拷贝mengxuegu文件到宿主机的 /opt 目录下 (在宿主机中操作)

```
1 docker cp mycentos2:/opt/mengxuegu /opt/mengxuegu-copy
```

```
[root@0c20b3037f46 /]# cat /opt/mengxuegu
123456
[root@0c20b3037f46 /]# echo abc >> /opt/mengxuegu
[root@0c20b3037f46 /]# cat /opt/mengxuegu
123456
abc
[root@0c20b3037f46 /]# exit
exit
[root@192 /]# docker cp mycentos2:/opt/mengxuegu /opt/mengxuegucopy
[root@192 /]# cat /opt/mengxuegucopy
123456
abc
[root@192 /]#
```

- 注意：文件拷贝 `docker cp` 命令 均在宿主机中操作

5.7 数据目录挂载

我们可以在创建容器的时候，将宿主机的目录与容器内的目录进行映射，这样我们就可以通过修改宿主机某个目录的文件从而去影响容器。使用 `-v` 选项

```
1 docker run -id -v /宿主机绝对路径目录:/容器内目录 --name 容器名 镜像名
```

5.7.1 需求

将宿主机的/opt目录与容器内的/opt目录进行映射，当在宿主机 /opt 目录下创建一个文件 `test.txt`，这个 `test.txt` 会 **自动同步** 到容器映射目录 /opt

5.7.2 实现

1. 创建容器并挂载映射目录（使用 `-v 宿主机目录:容器目录`）

创建容器时，将 宿主机目录/opt 挂载 容器目录/opt

```
1 docker run -id -v /opt:/opt --name=mycentos3 centos:7
```

2. 在宿主机/opt 目录下创建一个文件 `test.txt`，这个 `test.txt` 会 **自动同步** 到容器映射目录 /opt 目录下

```
1 echo 1 > /opt/test.txt
2 docker exec -it mycentos3 /bin/bash
3 cat /opt/test.txt
4 1111
```



```
[root@192 ~]# docker run -id -v /opt:/opt --name=mycentos3 centos:7
30d46774ac5cf3f3b4f5092e847abd2697937fb056b1692bedb03a7a360964c 创建容器并映射路径
[root@192 ~]# echo 1111 > /opt/test.txt 主机/opt目录添加文件及内容
[root@192 ~]# docker exec -it mycentos3 /bin/bash 进入容器
[root@30d46774ac5c /]# cat /opt/test.txt 容器已经自动同步了test.txt文件
1111
[root@30d46774ac5c /]#
```

5.7.3 目录挂载只读 (Read-only) 权限

- 实现挂载的目录只有 只读 (Read-only) 权限，命令如下：

```
1 docker run -id -v /宿主主机绝对路径目录:/容器内目录:ro --name=容器名 镜像名
```

- 实现：

```
1 docker run -id -v /dataHost:/dataContainer:ro --name=mycentos4 centos:7
```

```
[root@192 ~]# docker run -id -v /dataHost:/dataContainer:ro --name=mycentos4 centos:7
5690083d3d73e627f8e304c5f20b7945d0e46ddeda4c4a17b9097eac9b4b920a 只读read-only
[root@192 ~]# cd /dataHost/
[root@192 dataHost]# ls
[root@192 dataHost]# echo 123 > hello.txt
[root@192 dataHost]# docker exec -it mycentos4 /bin/bash 进入容器中
[root@5690083d3d73 /]# cat /dataContainer/hello.txt
123
[root@5690083d3d73 /]# echo aaa >> /dataContainer/hello.txt
bash: /dataContainer/hello.txt: Read-only file system
[root@5690083d3d73 /]# 只读，不允许修改
```

5.8 看容器内部细节

- 查看容器运行内部细节，比如可看容器的IP

```
1 docker inspect mycentos2
```

```
[root@192 ~]# docker inspect mycentos2
[
  {
    "Id": "0c20b3037f467700ccd7f0094a6e02447e9579c8d51cb71a62772a9e50cac11b",
    "Created": "2019-03-18T08:28:03.644616748Z",
    "Path": "/bin/bash",
    "Args": [],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 11904,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2019-03-18T09:15:39.746379066Z",
      "FinishedAt": "2019-03-18T09:07:09.541428183Z"
    },
    "Image": "sha256:9f38484d220fa527b1fb19747638497179500a1bed8bf0498eb788229229e6e1",
    "ResolvConfPath": "/var/lib/docker/containers/0c20b3037f467700ccd7f0094a6e02447e9579c8d51cb71a62772a9e50cac11b/resolv.conf",
    "HostPath": "/var/lib/docker/containers/0c20b3037f467700ccd7f0094a6e02447e9579c8d51cb71a62772a9e50cac11b"
  }
]
```

5.9 查看容器IP地址

- 直接显示IP地址

```
1 docker inspect --format '{{.NetworkSettings.IPAddress}}' mycentos2
```

```
[root@192 ~]# docker inspect --format '{{.NetworkSettings.IPAddress}}' mycentos2
172.17.0.2
[root@192 ~]#
```

5.10 删除容器

- 删除指定的容器：

```
1 docker rm 容器名称 | 容器ID
```

注意，只能删除停止的容器

```
[root@192 ~]# docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS          NAMES
30d46774ac5c   centos:7   "/bin/bash"             26 minutes ago Up 26 minutes   mycentos3
0c20b3037f46   centos:7   "/bin/bash"             4 hours ago    Up 3 hours     mycentos2
b5eeda5e8e2a   centos:7   "/bin/bash"             4 hours ago    Up 2 hours     mycentos1
12b974bb4ac2   centos:7   "/bin/bash"             4 hours ago    Exited(137) 3 hours ago mycentos
```

运行中无法被删除PORTS
只能删除停止的容器

```
[root@192 ~]# docker rm mycentos3 删除运行的报错
Error response from daemon: You cannot remove a running container 30d46774ac5cfd3f3b4f5092e847abd2697937fb056b1692bedb03a7a360964c. Stop the container before attempting removal or force remove
[root@192 ~]# docker rm mycentos 删除已停止的 mycentos 容器
mycentos
[root@192 ~]#
```

- 删除所有容器（其中运行中的容器无法删除，所以先停再删）：

```
1 docker rm `docker ps -a -q`
```

```
[root@192 ~]# docker kill `docker ps -a -q` 先停止所有容器
30d46774ac5c
0c20b3037f46
b5eeda5e8e2a
[root@192 ~]# docker rm `docker ps -a -q` 再删除所有容器
30d46774ac5c
0c20b3037f46
b5eeda5e8e2a
[root@192 ~]# docker ps -a 查看，容器已经被全部删除
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS          NAMES
[root@192 ~]#
```

第六章 实战部署应用

6.1 MySQL 部署

6.1.1 拉取MySQL镜像

- 拉取命令

```
1 docker pull mysql:5.7
```

```
[root@192 ~]# docker pull mysql:5.7
5.7: Pulling from library/mysql
f7e2b70d04ae: Pull complete
df7f6307ff0a: Pull complete
e29ed02b1013: Pull complete
9cb929db392c: Pull complete
42cc77b24286: Pull complete
a6d57750cc73: Pull complete
79510826e343: Pull complete
07e462ad61e2: Pull complete
fa594cb5b94d: Pull complete
1b44278270ad: Pull complete
3edb3c323f55: Pull complete
Digest: sha256:de482b2b0fdb5bb142462c07c5650a74e0daa31e501bc52448a2be10f384e6d
Status: Downloaded newer image for mysql:5.7
[root@192 ~]#
```

- 查看镜像 `docker images`

```
[root@192 ~]# docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
mysql                5.7          ee7cbd482336     2 weeks ago     372MB
```

6.1.2 创建 MySQL 容器

```
1 docker run -id --name mxg_mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 mysql:5.7
```

`-p` 代表端口映射, 格式为 宿主机映射端口:容器运行端口

`-e` 代表添加环境变量, `MYSQL_ROOT_PASSWORD` 是 root 用户的登陆密码

```
[root@192 ~]# docker run -id --name=mxg_mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 mysql:5.7
af98f77a000ce78277adcbd3618ba24409d48cf608abd6bc2768e54173bf26d4
[root@192 ~]#
```

6.1.3 进入MySQL容器, 登陆MySQL

- 进入mysql容器

```
1 docker exec -it mxg_mysql /bin/bash
```

- 登陆mysql, 密码是上面设的 123456

```
1 mysql -u root -p
```

```
[root@192 ~]# docker exec -it mxg_mysql /bin/bash 进入mysql容器
root@aaf98f77a000c:/# mysql -u root -p 容器中登录Mysql
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.25 MySQL Community Server (GPL)
```

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

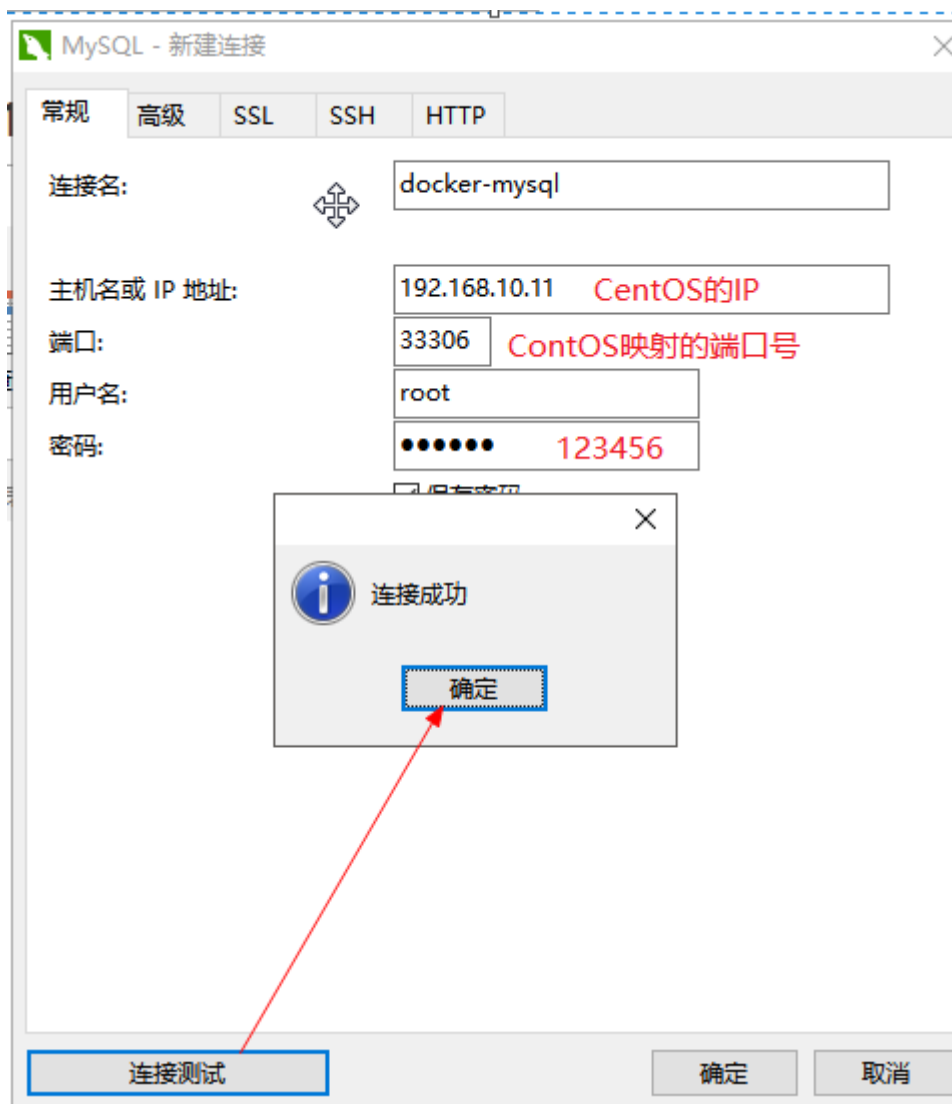
Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
mysql> 登录成功
```

6.1.4 远程登陆MySQL

1. 在我们本机的电脑上去连接虚拟机 Centos 中的Docker容器，这里 192.168.10.11 是虚拟机操作系统的IP，端口号是映射端口：33306



2. 在本地客户端执行创建库和表，增删改查等等操作

如连接不上，则查看宿主机防火墙有没关闭或者是上面暴露端口号配置是否正确

```
1 查看状态：systemctl status firewalld
2 关闭：systemctl stop firewalld
3 开机禁用：systemctl disable firewalld
```

6.2 Redis 部署

6.2.1 拉取Redis镜像

```
1 docker pull redis
```

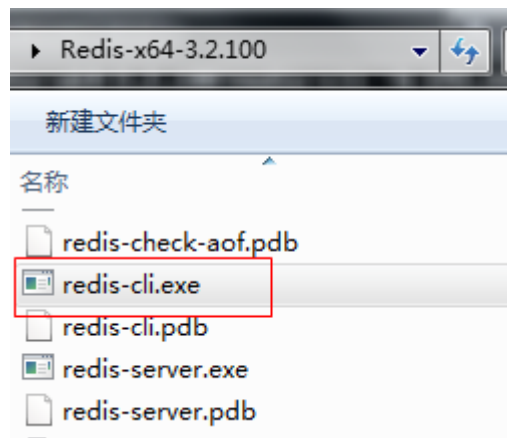
6.2.2 创建Redis容器

```
1 docker run -id --name mxg_redis -p 6379:6379 redis
```

6.2.3 客户端测试

- 在你的本地电脑命令提示符下，用window版本redis 进行测试

工具位于：Docker配套软件\03-Redis工具\Redis-x64-3.2.100.zip



- 打开windows的 cmd, 切换到客户端所在目录，然后输入以下命令，进行连接：

```
1 redis-cli -h 192.168.10.11
```

```
E:\Redis-x64-3.2.100>redis-cli -h 192.168.10.11
192.168.10.11:6379> _
```

6.3 Tomcat 部署

6.3.1 拉取tomcat镜像

```
1 docker pull tomcat:8
```

```
[root@192 ~]# docker pull tomcat:8
8: Pulling from library/tomcat
22dbe790f715: Pull complete
0250231711a0: Pull complete
6fba9447437b: Pull complete
4afad9c4aba6: Pull complete
12fce9923c9d: Pull complete
0dd81083d77e: Pull complete
4c2afc6ff72d: Pull complete
a62332a260f5: Pull complete
96dfb5ad080f: Pull complete
782f63228ce1: Pull complete
d610fffe30ad: Pull complete
Digest: sha256:aacce4e5ca37a3b8241c544deffd65f4cafbfb1a2fb2066f97621253c34f7dc4
Status: Downloaded newer image for tomcat:8
[root@192 ~]#
```

6.3.2 创建tomcat容器

创建tomcat容器用于 Web应用，并且进行目录映射

```
1 docker run -id --name=mxg_tomcat -p 8888:8080 -v /usr/local/project:/usr/local/tomcat/webapps --privileged=true tomcat:8
```

-p 表示地址映射, 宿主机端口号:容器运行端口号

-v 表示地址映射, 宿主机目录:容器映射目录

--privileged=true 如果映射的是多级目录，防止有可能出现没有权限的问题，所以加上此参数

```
[root@192 ~]# docker run -id --name=mxg_tomcat -p 8888:8080 -v /usr/local/project:/usr/local/tomcat/webapps --privileged=true tomcat:8
e4b5ce7e262b762c2c39a50d10737d61b92d70ed7b8f2ec2790491e49789b382
[root@192 ~]#
```

6.3.3 进入Tomcat容器

```
1 [root@192 ~]# docker exec -it mxg_tomcat /bin/bash # 进入Tomcat容器
2
3 root@e4b5ce7e262b:/usr/local/tomcat# cd webapps/
4 root@e4b5ce7e262b:/usr/local/tomcat/webapps# ls # 查看webapps是空目录，没有任何文件
5 # 空的
6 root@e4b5ce7e262b:/usr/local/tomcat/webapps# exit # 退出容器
7 exit
8 [root@192 ~]#
```

6.3.4 部署web应用

1. 将 Web应用系统 的发布源码，放到宿主机的 /usr/local/project 目录下，它会自动同步到tomcat容器中的 webapp目录
 - 例如：在宿主机的 /usr/local/project 目录创建mengxuegu目录，往里增加一个 hello.html 文件，文件内容如下：

```
1 <html>
2     <body>hello docker tomcat</body>
3 </html>
```

执行命令：

```
1 [root@192 ~]# cd /usr/local/project/
2 [root@192 project]# mkdir mengxuegu
3 [root@192 project]# cd mengxuegu/
4 [root@192 mengxuegu]# vim hello.html
5 # 添加上面内容
6
7 [root@192 mengxuegu]# cat hello.html
8 <html>
9     <body>hello docker tomcat</body>
10 </html>
```

2. 再进入tomcat容器中查看是否已经同步

```
1 [root@192 ~]# docker exec -it mxg_tomcat /bin/bash # 进入容器
2 root@e4b5ce7e262b:/usr/local/tomcat# ls webapps/mengxuegu
3 hello.html
4 root@e4b5ce7e262b:/usr/local/tomcat# cat webapps/mengxuegu/hello.html
5 <html>
6     <body>hello docker tomcat</body>
7 </html>
```

3. 测试，地址栏输入：

<http://192.168.10.11:8888/mengxuegu/hello.html>



6.4 RabbitMQ 部署

6.4.1 拉取 RabbitMQ 镜像

```
1 docker pull rabbitmq:management
```

注意：如果docker pull rabbitmq 后面不带management，启动rabbitmq后是无法打开管理界面的，所以我们要下载带management插件的rabbitmq。

```
[root@192 ~]# docker pull rabbitmq:management
management: Pulling from library/rabbitmq
898c46f3b1a1: Pull complete
63366dfa0a50: Pull complete
041d4cd74a92: Pull complete
6e1bee0f8701: Pull complete
d258c5276992: Pull complete
07aff2c19d3f: Pull complete
66d54af144e9: Pull complete
98899efc6397: Pull complete
6c78d30096fa: Pull complete
59785d0cfd4d: Pull complete
f5cec8db8c8f: Pull complete
8311b7cb5912: Pull complete
Digest: sha256:0f8b43dd82c76e5312aea1de6966e9f3de84eded9889a587c8e1485d84e861c8
Status: Downloaded newer image for rabbitmq:management
```

6.4.2 创建 RabbitMQ 容器

方式一：创建镜像（默认用户名密码），远程连接端口5672，管理系统访问端口15672，默认用户名：guest，密码也是 guest

```
1 docker run -id --name mxg_rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:management
```

方式二：启动镜像（设置用户名密码）

```
1 docker run -id --name mxg_rabbitmq2 -e RABBITMQ_DEFAULT_USER=username -e
  RABBITMQ_DEFAULT_PASS=password -p 5672:5672 -p 15672:15672 rabbitmq:management
```

上面二选一

```
[root@192 ~]# docker run -id --name mxg_rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:management
f9e892950c2ae5afb2e10e7b037d152637d1bb84d318864786e5af4b035b309e0
[root@192 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
f9e892950c2a        rabbitmq:management  "docker-entrypoint.s..."  13 seconds ago     Up 10 seconds      4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, 15671/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp
af98f77a000c        mysql:5.7           "docker-entrypoint.s..."  22 hours ago       Up 17 minutes      33060/tcp, 0.0.0.0:3306->3306/tcp
```

6.4.3 访问Rabbit管理界面

- 访问管理界面的地址是 [http://\[宿主机IP\]:15672](http://[宿主机IP]:15672)，如：<http://192.168.10.11:15672>，默认 guest 用户，密码也是 guest。

192.168.10.11:15672

视频相关 阿里云 梦学谷 Java资料 硬件设备



Username: guest

Password:

Login

第七章 备份与迁移

7.1 容器保存为镜像

- 通过以下命令将容器保存为镜像:

```
1 docker commit [-m "提交的描述信息"] [-a "创建者"] 容器名称|容器ID 生成的镜像名[:标签名]
```

7.1.1 无目录挂载-容器保存为镜像

- 查看是否有挂载目录

```
1 docker inspect --format '{{.Mounts}}' 容器名
```

```
[root@192 ~]# docker inspect --format='{{.Mounts}}' mycentos2
[]
[root@192 ~]# docker inspect --format='{{.Mounts}}' mxg_tomcat
[{{bind /usr/local/project /usr/local/tomcat/webapps true rprivate}}]
[root@192 ~]#
```

- mycentos2 容器无数据目录挂载，保存为镜像方式如下：

```
1 docker commit mycentos2 mycentos_new:1.1
```

mycentos2 是容器名称

mycentos_new 是新的镜像名称

此镜像的内容就是你当前容器的内容，接下来你可以用此镜像再次运行新的容器

```
[root@192 ~]# docker commit mycentos2 mycentos_new:1.1
sha256:7066c2910e70eedfa07ba04c43723617d3b6f82a53cebb6672265c1d20c07e6e
[root@192 ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mycentos_new	1.1	7066c2910e70	6 seconds ago	202MB
rabbitmq	management	7cb42b93c14d	4 days ago	213MB
redis	latest	4161e91dcc29	4 days ago	95MB
centos	7	9f38484d220f	9 days ago	202MB
tomcat	8	dd6ff929584a	2 weeks ago	463MB
tomcat	latest	dd6ff929584a	2 weeks ago	463MB
mysql	5.6	69056560eb44	2 weeks ago	256MB
mysql	5.7	ee7cbd482336	2 weeks ago	372MB

- 创建并登录容器

```
1 docker run -it --name mxg_mycentos_new mycentos_new:1.1 /bin/bash
```

- 在第5.6章节向 mycentos2 容器中添加 mengxuegu 文件，所以新的容器中也有此文件，如下：

```
[root@192 ~]# docker run -it --name=mxg_mycentos_new mycentos_new:1.1 /bin/bash
[root@b3578fd83b8f /]# cd /opt/
[root@b3578fd83b8f opt]# ls
mengxuegu
[root@b3578fd83b8f opt]# cat mengxuegu 新容器的/opt下有mengxuegu文件
123456
[root@b3578fd83b8f opt]#
```

7.1.2 有目录挂载情况(难点)

- **问题：**如果 Docker 对容器挂载了数据目录，在将容器保存为镜像时，数据不会被保存到镜像中。
- **原因：**因为宿主机与容器做了路径映射，再 commit 一个新的镜像时，该路径下的所有数据都会被抛弃，不会被保存到新镜像中。可通过 `docker inspect --format='{{.Mounts}}'` 镜像名 查看是否有目录挂载。
- **解决：**
 - **目录挂载方法。**先把在宿主机的数据备份在某个目录下，在 `docker run` 的时候使用 `-v` 参数将宿主主机上的目录映射到容器里的目标路径中（tomcat 是 `/usr/local/tomcat/webapps`，mysql 是 `var/lib/mysql`）
 - **拷贝方法。**先把在宿主机的数据备份在某个目录下，通过拷贝的方法 `docker cp` 将备份的数据复制进容器里的目标路径中（tomcat 是 `/usr/local/tomcat/webapps`，mysql 是 `var/lib/mysql`）。

7.1.2.1 Tomcat 保存镜像实战操作(目录挂载方法)

- 查看数据保存的位置

```
1 docker inspect --format='{{.Mounts}}' mxg_tomcat
```

```
[root@192 /]# docker inspect --format='{{.Mounts}}' mxg_tomcat
[{{bind /usr/local/project /usr/local/tomcat/webapps true rprivate}}]
[root@192 /]# 宿主机目录 容器中目录
```

- 宿主机数据保存在 /usr/local/project，将此路径数据备份在 baseproject (如果后面镜像是提供给别人，则此备份的数据也同时提供)

```
1 cp -rf /usr/local/project/ /usr/local/baseproject
```

- mxg_tomcat 容器保存为镜像

```
1 docker commit mxg_tomcat tomcat_new:1
```

- 采用 **目录挂载方法** 创建容器，目录挂载时，宿主机的路径指定为备份数据目录可还原数据

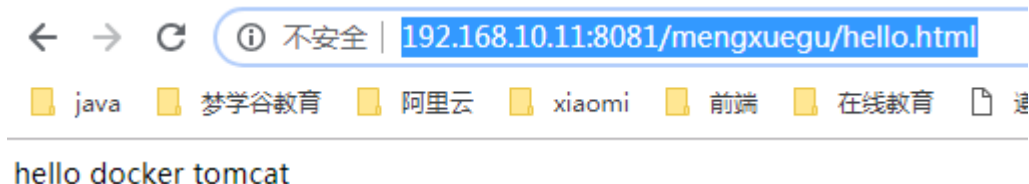
```
1 docker run -id --name mxg_tomcat_new -p 8081:8080 -v \  
2 /usr/local/baseproject:/usr/local/tomcat/webapps --privileged=true tomcat_new:1
```

- 登录查看webapps下有mengxuegu目录

```
1 docker exec -it mxg_tomcat_new /bin/bash
```

```
[root@192 mengxuegu]# docker exec -it mxg tomcat new /bin/bash  
root@1c09d1759788:/usr/local/tomcat# cd webapps/  
root@1c09d1759788:/usr/local/tomcat/webapps# ls  
mengxuegu
```

- 访问：<http://192.168.10.11:8081/mengxuegu/hello.html>



7.1.2.2 MySQL 保存镜像实战操作(拷贝方法)

- 查看数据保存的位置

```
1 docker inspect --format='{{.Mounts}}' mxg_mysql
```

```
[root@192 local]# docker inspect --format='{{.Mounts}}' mxg_mysql  
[{"volume": "c4ecc0d59c96bfea03428a799a7a63d6f5c0102a8c17a393f53ef11d3a0acc5",  
"/var/lib/docker/volumes/c4ecc0d59c96bfea03428a799a7a63d6f5c0102a8c17a393f53ef11d3a0acc5/_data",  
"/var/lib/mysql": "local", "true"}]  
[root@192 local]#
```

宿主机路径 容器路径

容器路径为：`/var/lib/mysql`，宿主机数据保存在：

```
1 /var/lib/docker/volumes/c4ecc0d59c96bfea03428a799a7a63d6f5c0102a8c17a393f53ef11d3a0acc5/_data
```

- 将此路径数据备份在 /base_data (如果后面镜像是提供给别人, 则此备份的数据也同时提供)

```
1 cp -rf /var/lib/docker/volumes/c4ecc0d59c96bfea03428a799a7a63d6f5c0102a8c17a393f53ef11d3a0acc5/_data /mysql
```

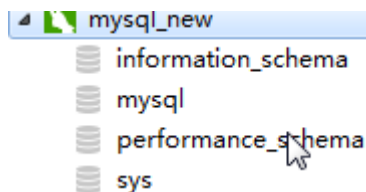
- mxg_mysql 容器保存为镜像

```
1 docker commit mxg_mysql mysql_new:1
```

- 通过上面保存的镜像创建容器

```
1 docker run -id --name mxg_mysql_new -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456 mysql_new:1
```

- 连接上 (192.168.10.11:3306), 查看当前没有 test 库, 即没有还原数据



- 采用 拷贝方法 进行还原数据, 容器中的数据目录为: /var/lib/mysql

```
1 docker cp /mysql/ mxg_mysql_new:/var/lib/
```

```
[root@192 ~]# docker cp /mysql/ mxg_mysql_new:/var/lib/
[root@192 ~]#
```

- 重启MySQL容器, 才可生效, 不然报错

```
1 docker restart mxg_mysql_new
```

必须重启, 不然刷新出现以下错误



注意: 如果发现上面刷新后, 发现没有test数据库, 说明docker cp的路径有问题, 一定要与上面步骤一样。

如果其他容器有数据目录挂载，解决方式同上面一致。

7.2 备份镜像

```
1 docker save -o mycentos.tar mycentos_new:1.1
```

-o 指定输出到的文件

- 执行后，运行 `ls` 命令即可看到打成的tar包，因为有463M所以打包要一会

```
[root@192 ~]# docker save -o mycentos.tar mycentos_new:1.1
[root@192 ~]# ls
anaconda-ks.cfg  initial-setup-ks.cfg  mycentos.tar
[root@192 ~]#
```

7.3 镜像恢复与迁移

- 首先我们先删除掉 `mycentos_new:1.1` 镜像(注意先停止并删除所有引用了的容器)

```
1 docker rmi mycentos_new:1.1
```

```
[root@192 ~]# docker ps -a
CONTAINER ID   IMAGE               COMMAND                  CREATED        STATUS              PORTS
888b396b2257   mycentos_new:1.1   "/bin/bash"            2 minutes ago Exited (127) 2 minutes ago
mycentos4
```

```
[root@192 ~]# docker rm 888b396b2257
888b396b2257
[root@192 ~]# docker rmi mycentos_new:1.1
Untagged: mycentos_new:1.1
Deleted: sha256:cbf68d07074e5f4a6e1639c403a45a96d7f9959d60a4a30963cad1bf21bebd67
Deleted: sha256:1e9bef6d7f379523a9c2a634112e7c61a86e52c6ddd6d1d54fa6928cc56dde81
```

- 然后执行此命令进行恢复 `mycentos_new:1.1` 镜像

```
1 docker load -i mycentos.tar
```

- `-i` 输入的镜像文件

```
[root@192 ~]# docker load -i mycentos.tar
8d003c7fdb2b: Loading layer [=====>] 6.144kB/6.144kB
Loaded image: mycentos_new:1.1
```

- 执行后再次查看镜像，可以看到镜像已经恢复

```
1 docker images
```

```
[root@192 ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mycentos new	1.1	cbf68d07074e	8 minutes ago	202MB
rabbitmq	management	7cb42b93c14d	4 days ago	213MB
redis	latest	4161e91dcc29	4 days ago	95MB
centos	7	9f38484d220f	9 days ago	202MB
tomcat	8	dd6ff929584a	2 weeks ago	463MB
tomcat	latest	dd6ff929584a	2 weeks ago	463MB
mysql	5.6	69056560eb44	2 weeks ago	256MB
mysql	5.7	ee7cbd482336	2 weeks ago	372MB

第八章 Dockerfile语法与实战

8.1 什么是Dockerfile

Dockerfile 用于构建一个新的Docker镜像的脚本文件，是由一系列命令和参数构成的脚本。

- 构建新的镜像步骤：
 - 编写 Dockerfile 文件
 - 通过 docker build 命令生成新的镜像
 - 通过 docker run 命令运行
- 查看 DockerFile 文件示例，以CentOS为例：

参考：https://hub.docker.com/_/centos

The screenshot shows the Docker Hub page for the CentOS image. On the left, under 'Supported tags and respective Dockerfile links', the 'latest' tag is highlighted, with a red box around the text 'latest, centos7, 7 (docker/Dockerfile)'. A red arrow points from this text to the 'docker/Dockerfile' link in the breadcrumb navigation of the Dockerfile view on the right. The Dockerfile content is displayed in a code block, showing the following instructions:

```
1 FROM scratch
2 ADD centos-7-x86_64-docker.tar.xz /
3
4 LABEL org.label-schema.schema-version="1.0" \
5       org.label-schema.name="CentOS Base Image" \
6       org.label-schema.vendor="CentOS" \
7       org.label-schema.license="GPLv2" \
8       org.label-schema.build-date="20190305"
9
10 CMD ["/bin/bash"]
```



```
1 FROM scratch # 基础镜像，scratch相当于java中的Object
2 ADD centos-7-x86_64-docker.tar.xz / # centos
3
4 LABEL org.label-schema.schema-version="1.0" \ # 标签说明
5     org.label-schema.name="CentOS Base Image" \
6     org.label-schema.vendor="CentOS" \
7     org.label-schema.license="GPLv2" \
8     org.label-schema.build-date="20190305"
9
10 CMD ["/bin/bash"] # 默认执行的命令，创建运行容器时最后会加上 /bin/bash，
11                  # 所以创建容器时，可不加 /bin/bash，即如下：
12                  # docker run -it --name=mycentos0 centos:7
13                  # 如果加了，则在后面采用我们自己加的命令执行/bin/bash
```

8.2 Dockerfile 语法规则

1. 每条指令的保留字都必须为大写字母且后面至少要有个参数
2. 执行顺序按从上往下执行。
3. # 用于注释
4. 每条指令都会创建一个新的镜像层，并对镜像进行提交

8.3 Dockerfile 执行流程

1. Docker 从基础镜像运行一个容器
2. 执行每一条指定并对容器作出修改
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容器
5. 执行 Dockerfile 中的下一条指令直到所有指令都执行完成

8.4 Dockerfile 常用指令

指令（大写的是保留字）	作用	参考
FROM image_name:tag	基础镜像，基于哪个基础镜像启动构建流程	
MAINTAINER user_name	镜像的创建者的姓名和邮箱地址等	contos6.8
COPY source_dir/file dest_dir/file	和ADD相似，但是如果有压缩文件并不能解压	contos6.8
ADD source_dir/file dest_dir/file	将宿主机的文件复制到容器内，如果是一个压缩文件，将会在复制后自动解压	contos6.8
ENV key value	设置环境变量(可以写多条)	tomcat7
RUN command	是Dockerfile的核心部分(可以写多条)，运行到当前行要执行的其他命令(可想象成sout("aaa"))	tomcat7
WORKDIR path_dir	设置工作目录，当创建容器后，命令终端默认登录进来后所在的目录。未指定则为根目录 /	tomcat7
EXPOSE port	当前对外暴露的端口号，使容器内的应用可以通过端口和外界交互	tomcat7
CMD argument	Dockerfile中可以有多条CMD，但是只有最后一个会生效。在构建容器时，会被 docker run 后面指定的参数覆盖。	tomcat7
ENTRYPOINT argument	和CMD相似，但是并不会被docker run指定的参数覆盖，而是追加参数	
VOLUME	将宿主机文件夹挂载到容器中	

8.5 Dockerfile 构建镜像实战

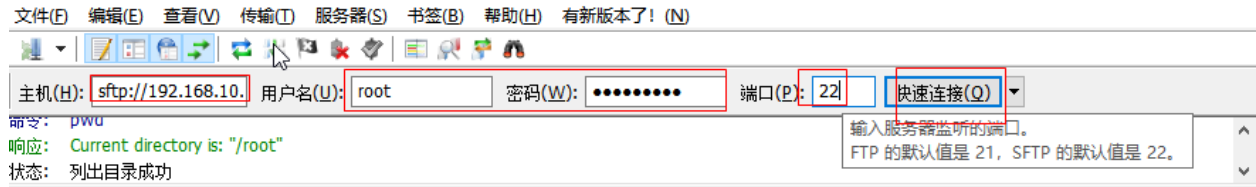
需求：构建一个jdk镜像

步骤：

1. 在宿主机上创建目录,并切换到新建的目录中

```
1 mkdir -p /usr/local/mydocker
2 cd /usr/local/mydocker
```

2. 下载 `jdk-8u111-linux-x64.tar.gz` 并上传到服务器（虚拟机）中的 `/usr/local/mydocker` 目录



```
[root@192 mydocker]# pwd
/usr/local/mydocker
[root@192 mydocker]# ll
??? 177192
-rw-r--r--. 1 root root 181442359 3? 19 16:45 jdk-8u111-linux-x64.tar.gz
[root@192 mydocker]#
```

3. 创建文件 Dockerfile：`vim Dockerfile`，然后按字母 `i` 插入状态，粘贴以下内容

```
1 #来自基础镜像
2 FROM centos:7
3 #指定镜像创建者信息
4 MAINTAINER mengxuegu
5 #切换工作目录 /usr/local
6 WORKDIR /usr/local
7 #创建一个存放jdk的路径
8 RUN mkdir /usr/local/java
9 #将jdk压缩包复制并解压到容器中/usr/local/java
10 ADD jdk-8u171-linux-x64.tar.gz /usr/local/java
11
12 #配置java环境变量
13 ENV JAVA_HOME /usr/local/java/jdk1.8.0_171
14 ENV JRE_HOME $JAVA_HOME/jre
15 ENV CLASSPATH
16     $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib:$CLASSPATH
17
18 CMD ["/bin/bash"]
```

4. 构建镜像语法：

```
1 docker build [-f 指定Dockerfile所在路径与文件名] -t 生成的镜像名:标签名 .
```

注意后边的 空格 和点 `.` 不要省略，`.` 表示当前目录

`-f` 指定Dockerfile文件所在路径与文件名。如果未指定 `-f` 值，则找当前目录下名为 `Dockerfile` 的构建文件

示例：生成镜像名为jdk，标签为1.8

```
1 docker build -t jdk:1.8 .
```

```
[root@192 mydocker]# docker build -t jdk:1.8 .
Sending build context to Docker daemon 181.4MB
Step 1/9 : FROM centos:7
7: Pulling from library/centos
3ba884070f61: Pull complete
Digest: sha256:8d487d68857f5bc9595793279b33d082b03713341ddec91054382641d14db861
Status: Downloaded newer image for centos:7
--> 9f38484d220f
Step 2/9 : MAINTAINER mengxuegu
--> Running in fe29a8432917
Removing intermediate container fe29a8432917
--> 696d88422658
Step 3/9 : WORKDIR /usr/local
--> Running in d59df4990c68
Removing intermediate container d59df4990c68
--> 8854975cc7b0
Step 4/9 : RUN mkdir /usr/local/java
--> Running in dc387c1bf2f6
Removing intermediate container dc387c1bf2f6
--> e64a83dfdd09
Step 5/9 : ADD jdk-8u111-linux-x64.tar.gz /usr/local/java/
--> ee50e249c4a2
Step 6/9 : ENV JAVA_HOME /usr/local/java/jdk1.8.0_111
--> Running in 4416930dc248
Removing intermediate container 4416930dc248
--> 19318f573d50
Step 7/9 : ENV JRE_HOME $JAVA_HOME/jre
--> Running in 3279f0f422e8
Removing intermediate container 3279f0f422e8
--> 0e5e73ae29c4
Step 8/9 : ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib:$CLASSPATH
--> Running in 4632f8f4f095
Removing intermediate container 4632f8f4f095
--> 4319a467ad29
Step 9/9 : ENV PATH $JAVA_HOME/bin:$PATH
--> Running in dbc3b56477b7
Removing intermediate container dbc3b56477b7
--> 9d83aaad867f
Successfully built 9d83aaad867f
Successfully tagged jdk:1.8
```

5. 查看镜像是否构建完成

```
1 docker images
```

```
[root@192 mydocker]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
jdk                  1.8                9d83aaad867f       6 minutes ago      567MB
```

6. 创建并运行容器

```
1 docker run -it --name=myjdk8 jdk:1.8 /bin/bash
```

```
[root@192 mydocker]# docker run -it --name=myjdk8 jdk:1.8 /bin/bash
[root@745dc3b715f8 local]# pwd
/usr/local
[root@745dc3b715f8 local]# java -version
java version "1.8.0_111"
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.111-b14, mixed mode)
```

大功告成，工作目录是 /usr/local

第九章 本地镜像发布到阿里云仓库

9.1 阿里云镜像仓库配置

1. 登录阿里云：<https://account.aliyun.com/login/login.htm>，然后进入管理控制台



2. 第一次进入可能会出现如下提示，则设置一下仓库密码。



容器镜像服务

容器镜像服务（Container Registry）提供多地域镜像托管能力，稳定的国内外镜像构建服务，便捷的镜像授权功能，方便用户进行镜像全生命周期管理。在开通流程中，您需要设置独立于账号密码的Registry登录密码，便于镜像的上传、下载。

如果您是子账号开通服务，请确认主账号已经设置过Registry登录密码。

设置Registry登录密码

设置Registry登录密码

Docker客户端登录时使用的用户名为阿里云账户全名，密码为当前设置的密码

用户名 阿里云账户全名

* 密码

8-32位，必须包含字母、符号或数字中的至少两项

* 确认密码

3. 创建一个公开镜像仓库

创建镜像仓库

1 仓库信息 2 代码源

地域 华东1 (杭州)

* 命名空间 mengxuegu

* 仓库名称 jdk

长度为2-64个字符，可使用小写英文字母、数字，可使用分隔符“-”、“.”（分隔符不能在首位或末位）

* 摘要 jdk

长度最长100个字符

描述信息 jdk1.8

支持Markdown格式

仓库类型 ☒ 公开 ☐ 私有

1

仓库信息

2

代码源

代码源

您可以通过命令行推送镜像到镜像仓库。

上一步 创建镜像仓库 取消

4. 点击仓库右边的 **管理** 进行查看推送镜像到Registry操作指南

镜像仓库

设置Registry登录密码 创建镜像仓库

全部命名空间

仓库名称

容器镜像服务已停止控制台授权功能，请迁移至RAM控制台设置子账号权限，具体参考《RAM权限管理文档》。

仓库名称	命名空间	仓库状态	仓库类型	权限	仓库地址	创建时间	操作
jdk	mengxuegu	● 正常	公开	管理	↓	2019-03-19 18:14:25	管理 删除

5. 推送镜像到 Registry 指南

1. 登录阿里云Docker Registry

```
$ sudo docker login --username=mengxuegu88 registry.cn-hangzhou.aliyuncs.com
```

用于登录的用户名为阿里云账号全名，密码为开通服务时设置的密码。

您可以在产品控制台首页修改登录密码。

2. 从Registry中拉取镜像

```
$ sudo docker pull registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:[镜像版本号]
```

3. 将镜像推送到Registry

```
$ sudo docker login --username=mengxuegu88 registry.cn-hangzhou.aliyuncs.com  
$ sudo docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:[镜像版本号]  
$ sudo docker push registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:[镜像版本号]
```

请根据实际镜像信息替换示例中的[ImageId]和[镜像版本号]参数。


9.2 本地镜像发布到阿里云镜像仓库

需求：将本地镜像 jdk:1.8 推送到 阿里云镜像仓库（每个人的registry地址不一样）

```
1 docker login --username=mengxuegu88 registry.cn-hangzhou.aliyuncs.com  
2 docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:[镜像版本号]  
3 docker push registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:[镜像版本号]
```

1. 登录阿里云Docker Registry, 密码为开通服务时设置的密码

```
1 docker login --username=mengxuegu88 registry.cn-hangzhou.aliyuncs.com
```

```
[root@192 mydocker]# docker login --username=mengxuegu88 registry.cn-hangzhou.aliyuncs.com  
Password:  此处输入的是Registry密码  
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 
```

2. 标记此镜像为阿里云仓库的镜像

```
1 docker tag 9d83aad867f registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:1.8
```



```
[root@192 mydocker]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jdk	1.8	9d83aaad867f	2 hours ago	567MB
mytomcat	1.1	c3541d3c33a7	4 hours ago	463MB
centos	7	9f38484d220f	4 days ago	202MB
tomcat	8	dd6ff929584a	13 days ago	463MB
redis	latest	0f88f9be5839	2 weeks ago	95MB
mysql	5.7	ee7cbd482336	2 weeks ago	372MB

```
[root@192 mydocker]# docker tag 9d83aaad867f registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:1.8
```

```
[root@192 mydocker]#
```

3. 推送到镜像到阿里云仓库

```
1 docker push registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:1.8
```

```
[root@192 mydocker]# docker push registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:1.8
```

The push refers to repository [registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk]

417baf7c1403: Pushed

dd7e785ff449: Pushed

d69483a6face: Pushed

1.8: digest: sha256:421d522401c7d3fa4e9a3211150d6d68d186c250ecaa6958c8320733371670be size: 949

```
[root@192 mydocker]#
```

4. 阿里云镜像库查看推送成功的jdk镜像

<https://cr.console.aliyun.com/repository/cn-hangzhou/mengxuegu/jdk/images>

版本	镜像ID	状态	Digest	镜像大小
1.8	217774d883e9...	● 正常	eda92a5218559f42f49 b424171d352f67659de 323b5fe874d492241b7 8d21505	254.972 MB

- 注意：如果你回头查看在【镜像仓库】中没有你创建的仓库时，则看下地区是不是和创建仓库时不一致。



9.3 拉取与运行阿里云的镜像

1. 复制库中的镜像地址，用于拉取Registry中的jdk:1.8镜像



- 1 `s# 拉取 Registry 中的jdk:1.8镜像`
- 2 `docker pull registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:1.8`

2. 创建jdk容器

- 1 `docker run -it --name=mxg_jdk8 9d83aaad867f /bin/bash`

```
[root@192 mydocker]# docker pull registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:1.8
1.8: Pulling from mengxuegu/jdk
8ba884070f61: Already exists
6f35d72a2148: Pull complete
9e05dcc9e02c: Pull complete
Digest: sha256:421d522401c7d3fa4e9a3211150d6d68d186c250ecaa6958c8320733371670be
Status: Downloaded newer image for registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk:1.8
[root@192 mydocker]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
registry.cn-hangzhou.aliyuncs.com/mengxuegu/jdk	1.8	9d83aaad867f	3 hours ago	567MB
mytomcat	1.1	c3541d3c33a7	5 hours ago	463MB
centos	7	9f38484d220f	4 days ago	202MB
tomcat	8	dd6ff929584a	13 days ago	463MB
redis	latest	0f88f9be5839	2 weeks ago	95MB
mysql	5.7	ee7cbd482336	2 weeks ago	372MB

```
[root@192 mydocker]# docker run -it --name=mxg_jdk8 9d83aaad867f /bin/bash
[root@5a3c363e5abc local]# java version
Error: Could not find or load main class version
[root@5a3c363e5abc local]# java -version
java version "1.8.0_111"
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.111-b14, mixed mode)
[root@5a3c363e5abc local]#
```

查看已经拉取

创建登录容器

第十章 微服务Docker容器化自动部署

10.1 微服务部署方法

1. 传统手动部署：

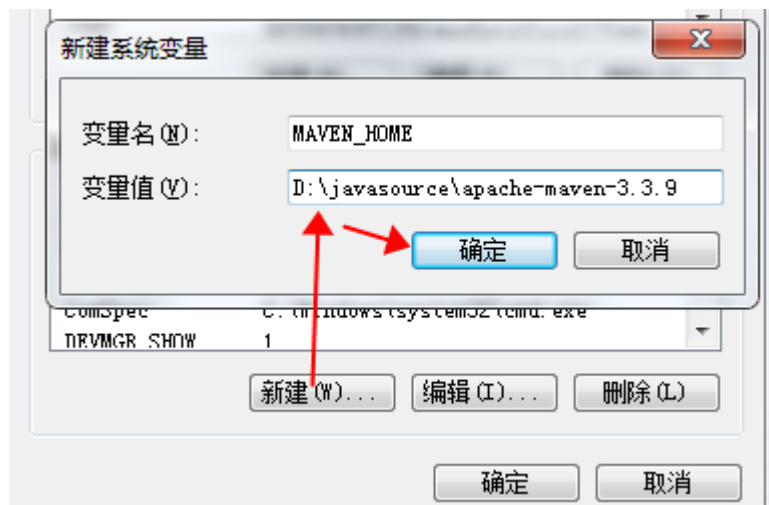
首先基于源码打包生成jar包（或war包），将jar包（或war包）上传至虚拟机并拷贝至JDK容器。

2. 通过Maven插件自动部署：对于数量众多的微服务，手动部署无疑是非常麻烦的做法，并且容易出错。所以我们这里学习如何自动部署，这也是企业实际开发中经常使用的方法。

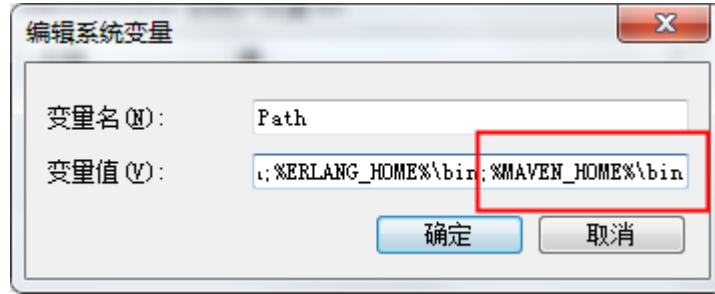
以下采用 Maven插件自动部署

10.2 配置Maven环境变量

1. 因为我们后面要在cmd命令行窗口使用 mvn 指令，所以要在本地配置Maven环境变量（如已配置请忽略）：



然后在Path中追加 ;%MAVEN_HOME%\bin



2. cmd命令窗口测试（后面也会在IDEA上操作，所以需要重启计算机才可以使用）

```
C:\Users\Administrator>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41
7+08:00)
Maven home: D:\javasource\apache-maven-3.3.9\bin\..
Java version: 1.8.0_151, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_151\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"
```

10.3 修改Docker配置

1. 修改宿主机的docker配置，让其docker服务可以远程访问，暴露的docker服务端口号 2375

```
1 vim /lib/systemd/system/docker.service
```

在 ExecStart= 后添加配置

```
1 -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock
```

0.0.0.0 代表所有 ip，也可指定 ip。修改后如下：

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

2. 刷新配置，重启服务

```
1 systemctl daemon-reload
2 systemctl restart docker
```

3. 验证是否生效，访问：<http://192.168.10.11:2375/version>，响应如下内容则成功：

```
192.168.10.11:2375/version
应用 MySQL Vue.js ky 虚拟机中的服务 本地应用 系统 Docker 淘宝 其他书签

{"Platform":{"Name":"Docker Engine - Community"},"Components":[{"Name":"Engine","Version":"18.09.3","Details":{"ApiVersion":"1.39","Arch":"amd64","BuildTime":"2019-02-28T06:02:24.000000000+00:00","Experimental":"false","GitCommit":"774alf4","GoVersion":"go1.10.8","KernelVersion":"3.10.0-693.el7.x86_64","MinAPIVersion":"1.12","Os":"linux"}],"Version":"18.09.3","ApiVersion":"1.39","MinAPIVersion":"1.12","GitCommit":"774alf4","GoVersion":"go1.10.8","Os":"linux","Arch":"amd64","KernelVersion":"3.10.0-693.el7.x86_64","BuildTime":"2019-02-28T06:02:24.000000000+00:00"}
```

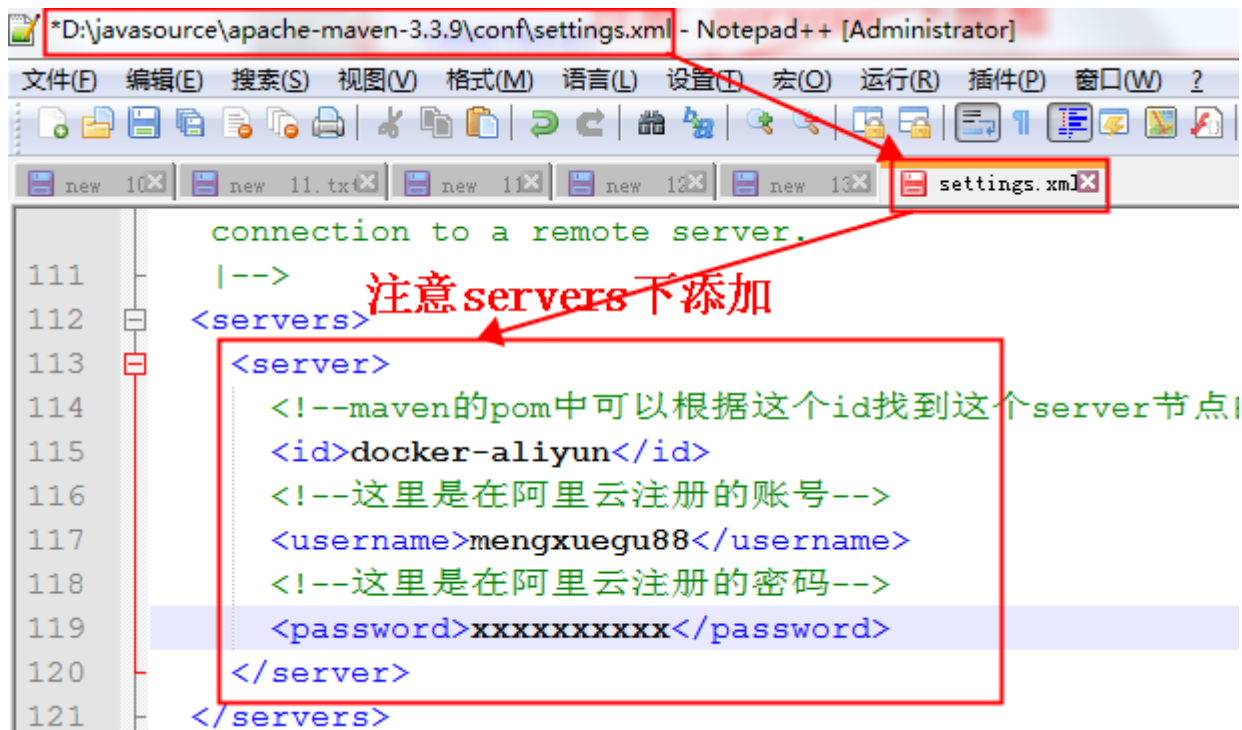
如果访问不到，则查看防火墙是否关闭

- 1 查看状态：systemctl status firewalld
- 2 关闭：systemctl stop firewalld
- 3 开机禁用：systemctl disable firewalld

10.4 自动部署 Config 服务端

使用Maven插件自动部署 Config 配置中心：microservice-cloud-11-config-server-5001

1. 在 Maven 的安装目录下的 settings.xml 文件中添加阿里云用户名和密码



- 1 <server>
- 2 <!--maven的pom中可以根据这个id找到这个server节点的配置-->
- 3 <id>docker-aliyun</id>
- 4 <!--这里是在阿里云注册的账号-->
- 5 <username>改阿里云注册的账号</username>
- 6 <!--这里是在阿里云注册的密码-->
- 7 <password>改阿里云注册的密码</password>
- 8 </server>

2. 在 microservice-cloud-11-config-server-5001 工程 pom.xml 增加插件配置

位于：Docker\ Docker配套资源文件\ docker-maven自动部署pom.xml

```
1 <build>
2   <finalName>app</finalName>
3   <plugins>
4     <!-- 插件一定要在其他构建插件之上，否则打包文件会有问题。 -->
5     <plugin>
6       <groupId>org.springframework.boot</groupId>
7       <artifactId>spring-boot-maven-plugin</artifactId>
8     </plugin>
9     <!-- docker的maven插件，官网：
10        https://github.com/spotify/docker-maven-plugin -->
11     <plugin>
12       <groupId>com.spotify</groupId>
13       <artifactId>docker-maven-plugin</artifactId>
14       <version>1.0.0</version>
15       <!--生成镜像相关配置-->
16       <configuration>
17         <!-- 将forceTags设为true，这样就会覆盖构建相同标签的镜像 -->
18         <forceTags>true</forceTags>
19         <!-- 远程 docker 宿主机地址，端口号
是/lib/systemd/system/docker.service所暴露的端口号，生成镜像到docker中 -->
20         <dockerHost>http://192.168.10.11:2375</dockerHost>
21         <!--内容是之前修改的maven的settings.xml配置文件中，server节点的id-->
22         <serverId>docker-aliyun</serverId>
23         <!-- 镜像名：阿里云镜像仓库地址
24             ${project.artifactId}引用当前工程名，
25             ${project.version}引用当前工程版本号
26             registry.cn-
hangzhou.aliyuncs.com/mengxuegu/demo:0.0.1-SNAPSHOT -->
27         <imageName>registry.cn-
hangzhou.aliyuncs.com/mengxuegu/${project.artifactId}:${project.version}
</imageName>
28         <!--基础镜像-->
29         <!--<baseImage>jdk1.8</baseImage>-->
30         <baseImage>java</baseImage>
31         <!--类似于Dockerfile的ENTRYPOINT指令-->
32         <entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]
</entryPoint>
33         <resources>
34           <resource> <!-- 指定打包的资源文件 -->
35             <targetPath></targetPath> <!-- 指定要复制的目录路径，这里是
当前目录 -->
36             <directory>${project.build.directory}</directory> <!--
指定要复制的根目录，这里是target目录 -->
37             <include>${project.build.finalName}.jar</include> <!--
这里指定最后生成的jar包 -->
38           </resource>
39         </resources>
40       </configuration>
41     </plugin>
```

```
42     </plugins>
43 </build>
```

3. 在阿里云创建一个 `microservice-cloud-11-config-server-5001` 仓库



容器镜像服务 | 镜像仓库 | 创建镜像仓库

命名空间: mengxuegu

* 仓库名称: microservice-cloud-11-config-server-5001
长度为2-64个字符，可使用小写英文字母、数字，可使用分隔符“-”、“.”（分隔符不能在首位或未位）

仓库类型: ☒ 公开 ☐ 私有

* 摘要: config-server
长度最长100个字符

描述信息: config-server配置服务端

支持Markdown格式

下一步

4. 在CMD的命令提示符下，进入microservice-cloud-11-config-server-5001工程所在的目录，输入以下 命令，进行打包和上传镜像（mvn clean package 打包成jar，然后docker:build构建成镜像，-DpushImage推送到仓库）：

如果报mvn不是内部命令，则maven配置环境变量，然后重启电脑

```
1 mvn clean package docker:build -DpushImage
```




```
Step 1/3 : FROM java
```

```
---> d23bdf5b1b1b
```

构建镜像中

```
Step 2/3 : ADD /app.jar //
```

```
---> 09d66b222e04
```

```
Step 3/3 : ENTRYPOINT ["java", "-jar", "/app.jar"]
```

```
---> Running in d4b601713a8b
```

```
Removing intermediate container d4b601713a8b
```

```
---> d555ca0ba449
```

```
ProgressMessage{id=null, status=null, stream=null, error=null}
```

```
Successfully built d555ca0ba449
```

```
Successfully tagged registry.cn-hangzhou.aliyuncs.com/mengxuegu/mengxuegu
```

```
[INFO] Built registry.cn-hangzhou.aliyuncs.com/mengxuegu/mengxuegu
```

```
[INFO] Pushing registry.cn-hangzhou.aliyuncs.com/mengxuegu/mengxuegu
```

```
The push refers to repository [registry.cn-hangzhou.aliyuncs.com/mengxuegu/mengxuegu]
```

```
355819b587e6: Preparing
```

```
35c20f26d188: Preparing
```

开始推送到阿里仓库中

```
355819b587e6: Pushed
```

```
?[8B1.0-SNAPSHOT: digest: sha256:1d1da146643e13bd
```

```
null: null
```

```
[INFO] -----
```

```
[INFO] BUILD SUCCESS
```

构建镜像并推送成功

```
[INFO] -----
```

```
[INFO] Total time: 01:16 min
```

```
[INFO] Finished at: 2019-03-25T21:05:40+08:00
```

```
[INFO] Final Memory: 50M/399M
```

```
[INFO] -----
```

如果报连接超时Timeout:

```
1 Exception caught: Timeout: GET http://192.168.10.11:2375/version:
com.spotify.docker.client.shaded.javax.ws.rs.ProcessingException:
org.apache.http.conn.ConnectTimeoutException: Connect to 192.168.10.11:2375
[/192.168.10.12] failed: connect timed out -> [Help 1]
```

```
com.spotify.docker.client.shaded.javax.ws.rs.ProcessingException: org.apache.http.conn.ConnectTimeoutException: Connect to 192.168.10.11:2375
```


5. 宿主机查看 docker 上就有 `microservice-cloud-11-config-server-5001` 镜像

REPOSITORY	TAG	IMAGE ID
registry.cn-hangzhou.aliyuncs.com/mengxuegu/microservice-cloud-11-config-server-5001.0-SNAPSHOT		d555ca0ba449

6. 去阿里云仓库查看，镜像ID为 d555ca0ba449



可看到容器正在启动

上面查看日志启动正常，你可以再采用守护容器方式创建容器在后台进行运行。

```
1 # 如果存在 config_server 容器，则把它先停止再删除
2 docker stop config_server
3 docker rm config_server
4
5 # 创建守护容器
6 docker run -id --name config_server -p 5001:5001 d555ca0ba449
```

8. 测试，项目启动需要一定时间，所以过一会访问配置内容：

<http://192.168.10.11:5001/microservice-config-product-dev.yml>



```
eureka:
  client:
    fetchRegistry: true
    registerWithEureka: true
    serviceUrl:
      defaultZone: http://eureka6001.com:6001/eureka
  instance:
    instanceId: microservice-product-config:8001
    prefer-ip-address: true
mybatis:
  config-location: classpath:mybatis/mybatis.cfg.xml
  mapper-locations: classpath:mybatis/mapper/**/*.xml
  type-aliases-package: com.mengxuegu.springcloud.entities
server:
```

10.5 自动部署 Eureka 注册中心

1. 保证 Config 服务端容器是启动状态，因为 Eureka 需要连接它

```
[root@192 ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS	PORTS
64be495777	d555ca0ba449	"java -jar /app.jar"	config_server	About an hour ago	Up About an hour	0.0.0.0:5001->5001/tcp

2. 在阿里云创建一个 `microservice-cloud-13-eureka-config-6001` 仓库

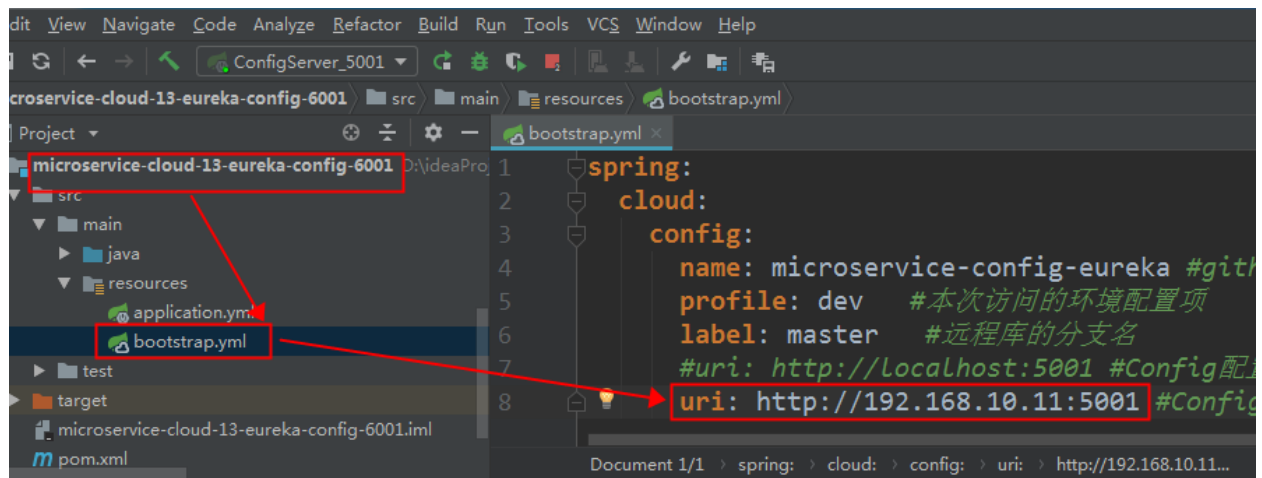
3. 修改 GitHub 上面的 microservice-config-eureka.yml 文件中的
eureka.instance.hostname 和 eureka.client.serviceUrl.defaultZone

完整配置位于：Docker配套资源文件\microservice-config-eureka.yml

```
1 eureka:
2   instance:
3     hostname: 192.168.10.11 #修改此处
4   client:
5     registerWithEureka: false
6     fetchRegistry: false
7     serviceUrl:
8       defaultZone: http://${eureka.instance.hostname}:6001/eureka/ #修改此处
9   server:
10    enable-self-preservation: true # 开启自我保护机制*****
```

4. 修改 microservice-cloud-13-eureka-config-6001 工程中 bootstrap.yml 的 Config 服务端地址：

```
1 uri: http://192.168.10.11:5001
```



注意一定要修改，不然启动会报错

5. 在 microservice-cloud-13-eureka-config-6001 工程 pom.xml 增加插件配置

```
1 <build>
2   <finalName>app</finalName>
3   <plugins>
4     <!-- 插件一定要在其他构建插件之上，否则打包文件会有问题。 -->
5     <plugin>
6       <groupId>org.springframework.boot</groupId>
7       <artifactId>spring-boot-maven-plugin</artifactId>
8     </plugin>
9     <!-- docker的maven插件，官网：
10        https://github.com/spotify/docker-maven-plugin -->
11     <plugin>
12       <groupId>com.spotify</groupId>
13       <artifactId>docker-maven-plugin</artifactId>
14       <version>1.0.0</version>
15       <!--生成镜像相关配置-->
16       <configuration>
17         <!-- 将forceTags设为true，这样就会覆盖构建相同标签的镜像 -->
18         <forceTags>true</forceTags>
19         <!-- 远程 docker 宿主地址，端口号
20            是/lib/systemd/system/docker.service所暴露的端口号，生成镜像到docker中 -->
21         <dockerHost>http://192.168.10.11:2375</dockerHost>
22         <!--内容是之前修改的maven的settings.xml配置文件中，server节点的id-->
23         <serverId>docker-aliyun</serverId>
24         <!-- 镜像名：阿里云镜像仓库地址
25            ${project.artifactId}引用当前工程名，
26            ${project.version}引用当前工程版本号
27            registry.cn-
28            hangzhou.aliyuncs.com/mengxuegu/demo:0.0.1-SNAPSHOT -->
29         <imageName>registry.cn-
30            hangzhou.aliyuncs.com/mengxuegu/${project.artifactId}:${project.version}
31        </imageName>
32
33         <!--基础镜像-->
34         <!--<baseImage>jdk1.8</baseImage>-->
35         <baseImage>java</baseImage>
36         <!--类似于Dockerfile的ENTRYPOINT指令-->
```

```

32         <entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]
    </entryPoint>
33         <resources>
34             <resource> <!-- 指定打包的资源文件 -->
35                 <targetPath>/</targetPath> <!-- 指定要复制的目录路径，这里是
    当前目录 -->
36                 <directory>${project.build.directory}</directory> <!--
    指定要复制的根目录，这里是target目录 -->
37                 <include>${project.build.finalName}.jar</include> <!--
    这里指定最后生成的jar包 -->
38             </resource>
39         </resources>
40     </configuration>
41 </plugin>
42 </plugins>
43 </build>
    
```

6. 在CMD的命令提示符下，进入microservice-cloud-13-eureka-config-6001工程所在的目录，输入以下命令进行打包和上传镜像（mvn clean package 打包成jar，然后docker:build构建镜像，-DpushImage推送到仓库）：

```
1 mvn clean package docker:build -DpushImage
```

```

Terminal
+ 版权所有 (c) 2009 Microsoft Corporation。保留所有权利。
x
D:\ideaProject\springCloud\microservice-cloud-13-eureka-config-6001>mvn clean package docker:build -DpushImage
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building microservice-cloud-13-eureka-config-6001 1.0-SNAPSHOT
    
```

效果同 config 服务端一样

7. 宿主机查看 docker 镜像，镜像ID为 3b6efaf22549

```

[root@192 ~]# docker images
REPOSITORY                                TAG                                IMAGE ID
registry.cn-hangzhou.aliyuncs.com/mengxuegu/microservice-cloud-13-eureka-config-6001 1.0-SNAPSHOT                      3b6efaf22549
    
```

8. 去阿里云仓库查看，镜像ID为 3b6efaf22549

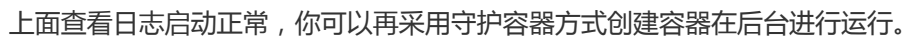
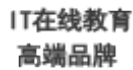


The screenshot shows the Alibaba Cloud Container Registry console. The left sidebar has a menu with '镜像版本' (Image Version) selected. The main content area shows the details for the image 'microservice-cloud-13-eureka-config-6001'. Under the '镜像版本' (Image Version) tab, there is a table with the following data:

版本	镜像ID	状态	Digest
1.0-SNAPSHOT	3b6efaf22549...	● 正常	4e6747a36cac8fd8fde46b408d29a7ab5ecad897df20ec54b61fcde9243d6e90

9. 启动容器，其中 3b6efaf22549 是镜像ID

```
1 docker run -it --name eureka_server -p 6001:6001 3b6efaf22549 /bin/bash
```



如果报错 Cannot execute request on any known server，说明 bootstrap.yml 的IP没有更改

```
com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server
    at com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClient.execute(Retryable
```

仅供购买者学习，禁止盗版、转卖、传播课程



System Status

Environment	test
Data center	default

THE SELF PRESERVATION MODE IS TURNED OFF.THIS MAY NOT PROTECT INSTANCES Replicas

Instances currently registered with Eureka

10.6 自动部署 Product 商品提供者

1. 在阿里云创建一个 `microservice-cloud-14-product-config-8001` 仓库



2. 保证 config_server 和 eureka_server 都是Up启动状态

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8cc73136c2ff	ecddb2329846	"java -jar /app.jar ..."	20 minutes ago	Up 50 seconds	0.0.0.0:6001->6001/tcp	eureka_server
9d64be495777	d555ca0ba449	"java -jar /app.jar"	About an hour ago	Up About an hour	0.0.0.0:5001->5001/tcp	config_server

3. 启动 MySQL5.7 和 RabbitMQ 容器，因为需要使用

```
1 docker start mxg_mysql
2 docker start mxg_rabbitmq
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
8cc73136c2ff	ecddb2329846	"java -jar /app.jar ..."	41 minutes ago	Up 21 min
9d64be495777	d555ca0ba449	"java -jar /app.jar"	About an hour ago	Up About an
fa2b968f6024	rabbitmq:management	"docker-entrypoint.s..."	2 days ago	Up 18 min
p, 15671/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp	mxg_rabbitmq	"docker-entrypoint.s..."	3 days ago	Up 10 secc
3f3a6e1877cc	mysql:5.7	"docker-entrypoint.s..."		
	mxg_mysql			

4. 导入商品提供者数据到端口号是 33306 的 MySQL 容器中，直接将以下脚本导入即可

脚本位于: Docker配套资源文件\springcloud_db01.sql 和 springcloud_db02.sql

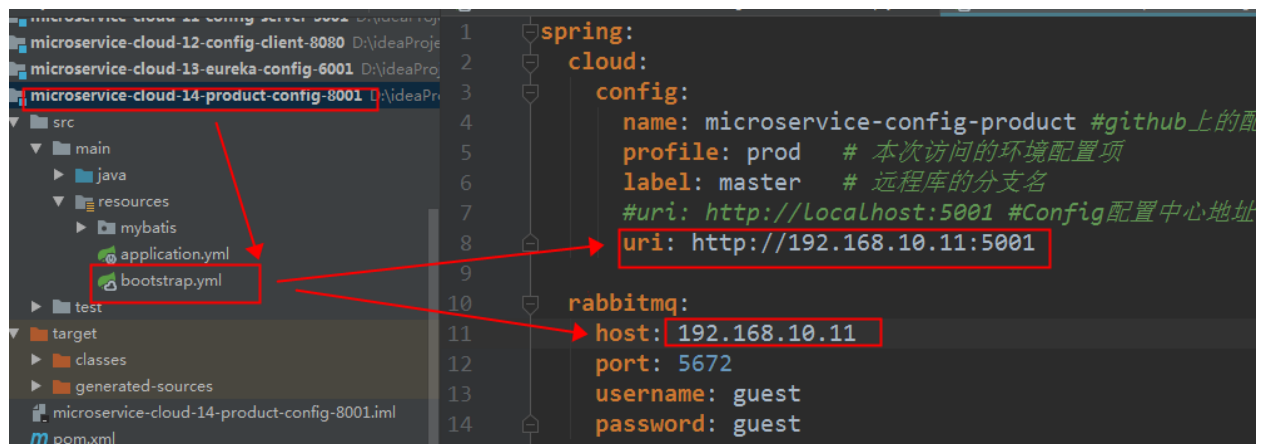
5. 修改 GitHub 上面的 microservice-config-product.yml 文件，修改数据库地址和密码、eureka 地址 spring.datasource.url、spring.datasource.password 和 eureka.client.serviceUrl.defaultZone 注意数据库端口：33306

完整配置位于: Docker配套资源文件\microservice-config-product.yml


```
1 spring:
2   profiles: dev # 开发环境
3   application:
4     name: microservice-product-config # *****服务名*****
5   datasource:
6     type: com.alibaba.druid.pool.DruidDataSource
7     driver-class-name: com.mysql.cj.jdbc.Driver
8     url: jdbc:mysql://192.168.10.11:33306/springcloud_db01?
        serverTimezone=GMT%2B8
9     username: root
10    password: 123456
11 eureka:
12   client:
13     registerWithEureka: true
14     fetchRegistry: true
15     serviceUrl:
16       defaultZone: http://192.168.10.11:6001/eureka
```

6. 修改 microservice-cloud-14-product-config-8001 工程中的 bootstrap.yml 配置:

config服务端与rabbitmq地址



```
1 spring:
2   cloud:
3     config:
4       name: microservice-config-product #github上的配置名称, 注意没有yml后缀名
5       profile: prod # 本次访问的环境配置项
6       label: master # 远程库的分支名
7       #uri: http://localhost:5001 #Config配置中心地址, 通过它获取microservice-
        config-product.yml配置信息
8       uri: http://192.168.10.11:5001
9
10    rabbitmq:
11      host: 192.168.10.11
12      port: 5672
13      username: guest
14      password: guest
15
16    # 暴露触发消息总线的地址
17    management:
```

```
18 endpoints:
19     web:
20         exposure:
21             include: bus-refresh
```

7. 在 `microservice-cloud-14-product-config-8001` 工程 pom.xml 增加插件配置

```
1 <build>
2     <finalName>app</finalName>
3     <plugins>
4         <!-- 插件一定要在其他构建插件之上，否则打包文件会有问题。 -->
5         <plugin>
6             <groupId>org.springframework.boot</groupId>
7             <artifactId>spring-boot-maven-plugin</artifactId>
8         </plugin>
9         <!-- docker的maven插件，官网：
10             https://github.com/spotify/docker-maven-plugin -->
11         <plugin>
12             <groupId>com.spotify</groupId>
13             <artifactId>docker-maven-plugin</artifactId>
14             <version>1.0.0</version>
15             <!--生成镜像相关配置-->
16             <configuration>
17                 <!-- 将forceTags设为true，这样就会覆盖构建相同标签的镜像 -->
18                 <forceTags>true</forceTags>
19                 <!-- 远程 docker 宿主机地址，端口号
是/lib/systemd/system/docker.service所暴露的端口号，生成镜像到docker中 -->
20                 <dockerHost>http://192.168.10.11:2375</dockerHost>
21                 <!--内容是之前修改的maven的settings.xml配置文件中，server节点的id-->
22                 <serverId>docker-aliyun</serverId>
23                 <!-- 镜像名：阿里云镜像仓库地址
24                     ${project.artifactId}引用当前工程名，
25                     ${project.version}引用当前工程版本号
26                     registry.cn-
hangzhou.aliyuncs.com/mengxuegu/demo:0.0.1-SNAPSHOT -->
27                 <imageName>registry.cn-
hangzhou.aliyuncs.com/mengxuegu/${project.artifactId}:${project.version}
</imageName>
28                 <!--基础镜像-->
29                 <!--<baseImage>jdk1.8</baseImage>-->
30                 <baseImage>java</baseImage>
31                 <!--类似于Dockerfile的ENTRYPOINT指令-->
32                 <entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]
</entryPoint>
33             <resources>
34                 <resource> <!-- 指定打包的资源文件 -->
35                     <targetPath></targetPath> <!-- 指定要复制的目录路径，这里是
当前目录 -->
36                     <directory>${project.build.directory}</directory> <!--
指定要复制的根目录，这里是target目录 -->
37                     <include>${project.build.finalName}.jar</include> <!--
这里指定最后生成的jar包 -->
38             </resources>
```

```
39         </resources>
40     </configuration>
41 </plugin>
42 </plugins>
43 </build>
```

8. 在CMD的命令提示符下，进入microservice-cloud-14-product-config-8001工程所在的目录，输入以下命令进行打包和上传镜像（mvn clean package 打包成jar，然后docker:build构建成镜像，-DpushImage推送到仓库）：

```
1 mvn clean package docker:build -DpushImage
```

```
Local Local (1)
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

D:\ideaProject\springCloud\microservice-cloud-14-product-config-8001>mvn clean package docker:build -DpushImage
[INFO] Scanning for projects...
[INFO]
```

效果同 Config 一样

9. 宿主机查看 docker 镜像，镜像ID为 628722467a73

```
[root@192 ~]# docker images
REPOSITORY                                TAG                IMAGE ID
SIZE
registry.cn-hangzhou.aliyuncs.com/mengxuegu/microservice-cloud-14-product-config-8001  1.0-SNAPSHOT      628722467a73
```

10. 去阿里云仓库查看，镜像ID为 628722467a73

管理控制台

microservice-cloud-14-product-config-8001

华东1 (杭州) | 公开 | 本地仓库 | ● 正常

基本信息

仓库授权

触发器

镜像版本

镜像同步

版本	镜像ID	状态	Digest	镜像大小
1.0-SNAPSHOT	628722467a73...	● 正常	09ea1758cd2bd230f1f b58439d4107660ff8b9 668236c374d1e88072 8aaca30d	294.668 MB

11. 启动容器，其中 628722467a73是镜像ID

```
1 docker run -it --name product_config -p 8001:8001 628722467a73 /bin/bash
```

```
[root@192 ~]# docker run -it --name=product_config -p 8001:8001 628722467a73 /bin/bash
2019-03-25 14:46:54.807 INFO 1 --- [ main] s.c.a.AnnotationConfigApplicationConte
notationConfigApplicationContext@e580929: startup date [Mon Mar 25 14:46:54 UTC 2019]; root
2019-03-25 14:46:55.838 INFO 1 --- [ main] f.a.AutowiredAnnotationBeanPostProcess
upported for autowiring
2019-03-25 14:46:55.969 INFO 1 --- [ main] trationDelegate$BeanPostProcessorCheck
on' of type [org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConf
le for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxy

:: Spring Boot :: (v2.0.7.RELEASE)
```

上面查看日志启动正常，你可以再采用守护容器方式创建容器在后台进行运行。

```
1 # 如果存在 product_config 容器，则把它先停止再删除
2 docker stop product_config
3 docker rm product_config
4
5 # 创建守护容器
6 docker run -id --name=product_config -p 8001:8001 3b6efaf22549
```

如果报错 Cannot execute request on any known server，说明 bootstrap.yml 的IP没有更改

```
1 com.netflix.discovery.shared.transport.TransportException: Cannot execute request
on any known server
```

```
com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server
at com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClient.execute(Retryable
```

如果报错 java.sql.SQLException: Access denied for user 'root'@'172.17.0.1' (using password: YES)

可能 数据库密码是错的。

12. 测试

- 再次访问Eureka注册中心，发现Product 成功注册Eureka中

192.168.10.11:6001

java 梦学谷教育 阿里云 xiaomi 前端 在线教育 遵循 semver 标准 Android CODING | 代码托... martin fowler webjars依赖

Renews (last min) 0

RENEWALS ARE LESSER THAN THE THRESHOLD. THE SELF PRESERVATION MODE IS TURNED OFF.THIS MAY NOT PROTECT INSTANCE NETWORK/OTHER PROBLEMS.

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MICROSERVICE-PRODUCT-CONFIG	n/a (1)	(1)	UP (1) - microservice-product-config:8001

- 再请求端口提供者查询数据：<http://192.168.10.11:8001/product/get/1>

← → ↻ ⓘ 不安全 | 192.168.10.11:8001/product/get/1

java 梦学谷教育 阿里云 xiaomi 前端 在线教育

```
{"pid":1,"productName":"格力空调","dbSource":"springcloud_db02"}
```