# DOCKER入门

## docker概述

### 什么是docker

Docker是一个用来开发、运输和运行应用程序的开放平台。使用Docker可以将应用程序与基础结构分离，以便快速交付软件。使用Docker，您可以以管理应用程序的方式管理基础架构。通过利用Docker的方法快速传送、测试和部署代码，可以显著减少编写代码和在生产中运行代码之间的延迟。

### 为什么使用docker（需求）

在传统企业中，有很多这样的情况(例如达内)：有在线的TMOOC系统，有TTS系统。有财务软件系统，有OA，有CRM等。这些系统都部署在一个服务器上是肯定不行的。如果服务器宕机了，会导致这些系统不能使用。所以为了安全性的考虑，有几个系统必须有几台服务器。如图-1所示：



图-1 应用部署结构

但是这样的结构有几个弊端：

1、有些系统运行不需要一个服务器的资源，例如crm。这样对服务器的资源就造成了浪费的情况。

2、服务器的资源利用率特别低。

3、企业为这些系统购买了服务器，花费了很大的成本。

4、必须有一些人去维护这些服务器。

所以就有了docker和容器的概念。如图-2所示：

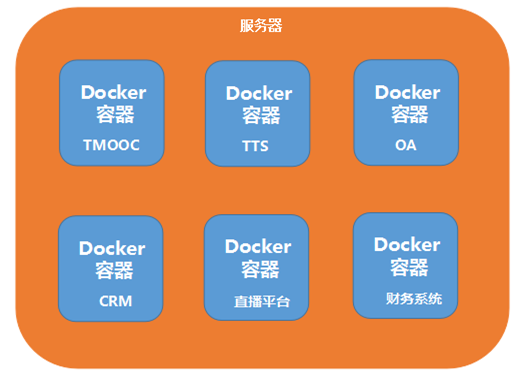


图-2 服务器部署docker结构

一个Docker容器可以理解为一个进程。同一个服务器可以启动多个docker容器进程。

一个Docker容器还可以理解为一个微缩的操作系统。你可以在这个容器内部实现你在一个操作系统中完成的所需功能。

可以在很短时间内增加上万个Docker容器。

可以在很短时间内删除掉上万个Docker容器。

### docker有什么优点

* 更快速的应用交付和部署：

传统的应用开发完成后，需要提供一堆安装程序和配置说明文档，安装部署后需根据配置文档进行繁杂的配置才能正常运行。Docker化之后只需要交付少量容器镜像文件，在正式生产环境加载镜像并运行即可，应用安装配置在镜像里已经内置好，大大节省部署配置和测试验证时间。

* 更便捷的升级和扩缩容：

随着微服务架构和Docker的发展，大量的应用会通过微服务方式架构，应用的开发构建将变成搭乐高积木一样，每个Docker容器将变成一块“积木”，应用的升级将变得非常容易。当现有的容器不足以支撑业务处理时，可通过镜像运行新的容器进行快速扩容，使应用系统的扩容从原先的天级变成分钟级甚至秒级。

* 更简单的系统运维：

应用容器化运行后，生产环境运行的应用可与开发、测试环境的应用高度一致，容器会将应用程序相关的环境和状态完全封装起来，不会因为底层基础架构和操作系统的不一致性给应用带来影响，产生新的BUG。当出现程序异常时，也可以通过测试环境的相同容器进行快速定位和修复。

* 更高效的计算资源利用：

Docker是内核级虚拟化，其不像传统的虚拟化技术一样需要额外的Hypervisor [管理程序] 支持，所以在一台物理机上可以运行很多个容器实例，可大大提升物理服务器的CPU和内存的利用率。

### docker相关资源

Docker官网：<http://www.docker.com>

Docker Hub官网（镜像仓库）：<https://hub.docker.com>

作为自学拓展，深入研究docker，这些是必备资源。

## docker安装

所有步骤，均参考官方文档：<https://docs.docker.com/engine/install/centos/>

本文档只对应在校学习期间对应的服务器环境，centos。

### 卸载旧版本docker（非必须）

为了防止环境中docker冲突，旧docker残留影响安装，需要先执行卸载。如果确定当前环境没有docker，可以不执行。

yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

运行命令候，如果有就删除了，如果没有docker，则如图-3所示：



图-3 没有docker删除记录

### 设置docker仓库

官方文档提供了相对的说明，安装docker有3种途径。如图-4所示：

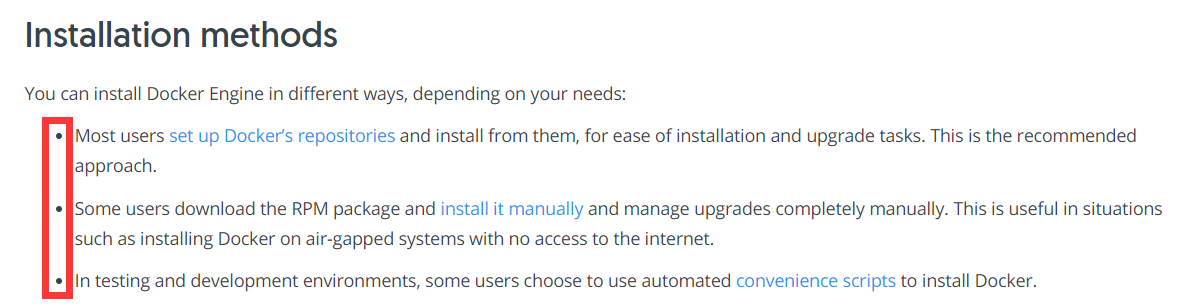


图-4 安装docker的三中方式

这三种方式分别是：

* 设置docker仓库，并且从仓库安装所需内容。
* 下载RPM安装包，手动更新安装。
* 为测试和开发环境使用自定义脚本安装。

我们使用最简单，也是官方最推荐的第一种方式来安装docker即可。

安装yum-utils包（提供用yum-config-manager实用程序）并设置稳定的存储库。

yum install -y yum-utils

yum-config-manager \

--add-repo \

<https://download.docker.com/linux/centos/docker-ce.repo>

如果有其他的仓库管理地址，也可以自行更改链接地址。两个yum命令执行完毕后，仓库设置完成，如图-5所示：



图-5 配置docker仓库结果

### 安装docker引擎

在官方文档中，配置完仓库地址，可选操作是配置测试仓库。如图-6所示：



图-6 可选开启测试仓库

这个nightly和test仓库本身是包含在上一步我们配置的docker.repo中的，但是默认没有开启，可选操作就是如果使用的话，可以enable开启。

注意：可选步骤后续不在解释，自行到官方文档查看。仅以本步骤为例解释。

接下来我们可以直接执行安装docker引擎的操作。执行以下命令，可以安装最新版本docker。

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

在安装过程中，由于操作系统环境中的yum依赖不一定满足安装的全部依赖需求，可能出现问题，如图-7所示：



图-7 docker安装缺少yum依赖

解决办法就是根据提示，使用yum安装依赖，可以从第一个开始尝试，有可能需要全部安装一遍，有可以只需要安装部分，那么其他依赖就自然传递过来。

根据图中我们缺少 container-selinux，fuse-overlayfs，slirp4netns这三个包。我们可以依次执行命令。

yum -y install container-selinux

yum -y install fuse-overlayfs

yum -y install slirp4netns

注意：有可能在这个阶段，需要更换或者更新yum源。比如这里的container-selinux就是需要更换yum源添加包的。

wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo

在执行上述命令前，对原有CentOS-Base.repo先进行备份。

将所需依赖包安装完成后，我们可以重新安装docker相关包。

### 启动验证docker

安装完毕后，我们可以启动docker。

systemctl start docker

然后运行docker的第一个镜像容器。

如果运行成功，可以看到提示信息，表示docker安装成功，如图-8所示：

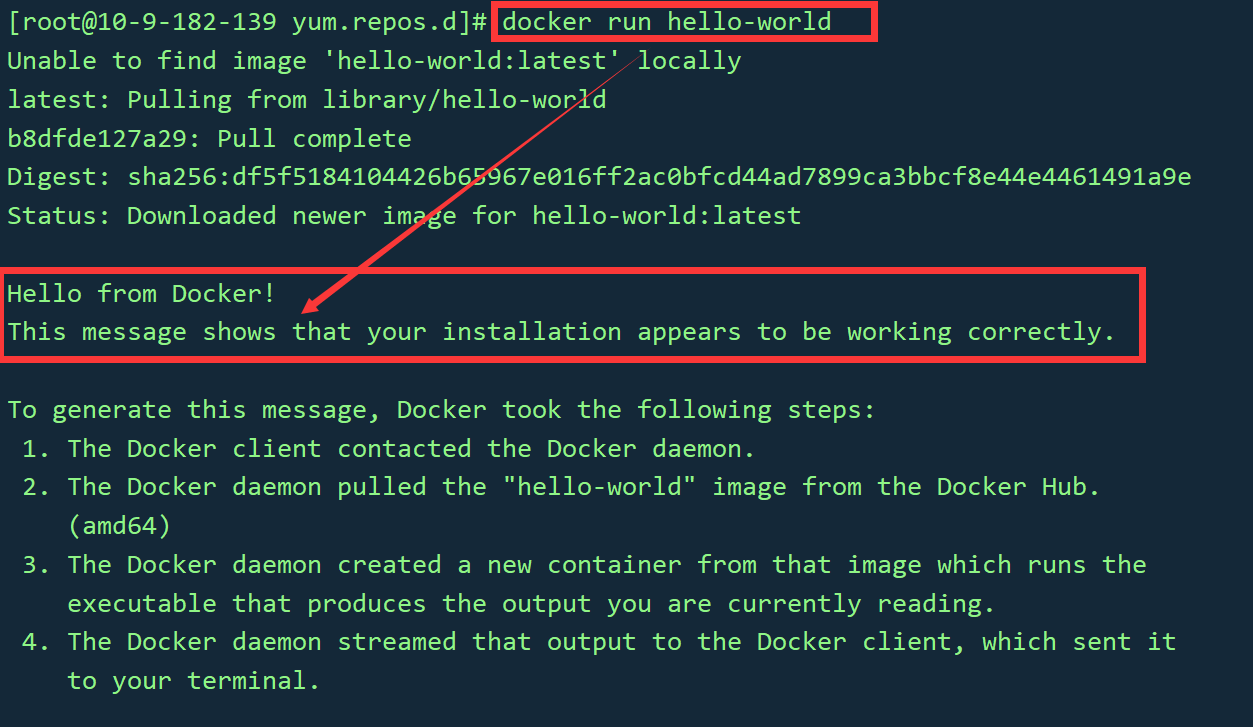


图-8 docker测试镜像hello-world

### 卸载docker

如果想要卸载已经安装好的docker也是非常简单的。

第一步先卸载相关安装包。

yum remove docker-ce docker-ce-cli containerd.io

然后将docker存储的资料数据删除。

rm -rf /var/lib/docker

rm -rf /var/lib/containerd

## docker底层原理

### docker的结构和基本概念

docker的主要结构是客户端，宿主机，容器，镜像和镜像服务器。如图-9所示：

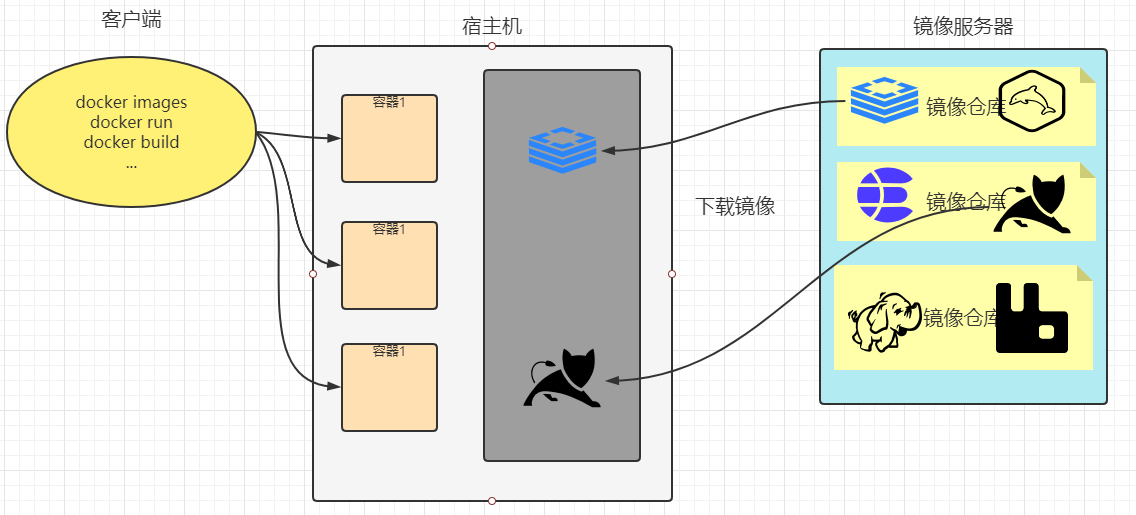


图-9 docker基本结构

* 容器（container）

首先需要了解什么是容器，容器就是一个进程，内部是独立运行的一个或者是一组应用。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。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。

* 镜像（image）

镜像（Image）就是一个只读的模板文件。镜像可以用来创建 Docker 容器，一个镜像可以创建很多容器。 就好似 Java 中的 类和对象，类就是镜像，容器就是对象！也可以把镜像看成是模具，而镜像创建出来的容器就是通过这个模具创建的一个一个的实际产品。

* 宿主机（host）

宿主机就是我们调用命令使用镜像创建容器的服务器。

* 镜像仓库（repository）

一个用来容纳多个镜像的仓库，可以链接仓库获取你想要的内部镜像，一般一个镜像仓库中包含多个不同tag的镜像。

* 镜像服务器（registry）

镜像仓库占用的服务器，这里注意一个镜像服务器未必只有一个仓库，可以有很多仓库，每个仓库又保管的是不同镜像。

* 客户端（docker-client）

调用docker命令，操作镜像，容器的进程。只要能链接宿主机，操作docker的进程都是docker-client。

### docker为什么比虚拟机快

虚拟化技术开启了云计算时代，是的大量硬件服务器可以隔离，可以独立。而docker开启了下一代虚拟化技术，但是docker并不是虚拟机。首先我们来了解一下物理机是怎么工作的。如图-10所示：

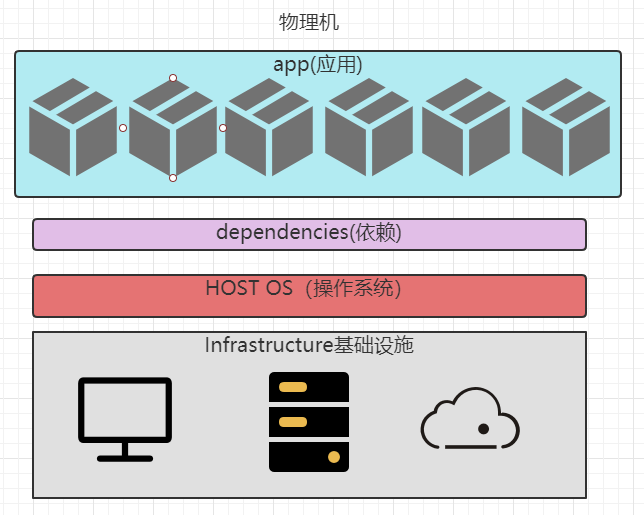


图-10 物理机运行架构

从下到上依次为：

* infrastructure基础设施：这个指的就是你的电脑硬件，可以是个人pc，可以是服务器，可以使云主机。
* HOST OS操作系统：安装使用的主机操作系统，可以是windows，MacOS，linux。
* dependencies依赖：运行程序需要用到的依赖，可以是插件，基础服务，语言环境等。
* app应用：用户使用物理机的应用程序，可以使qq，微信，游戏等。

当虚拟机出现之后，我们可以在一个物理机上安装启动多个虚拟机，这种结构开启了云时代，如图11所示：

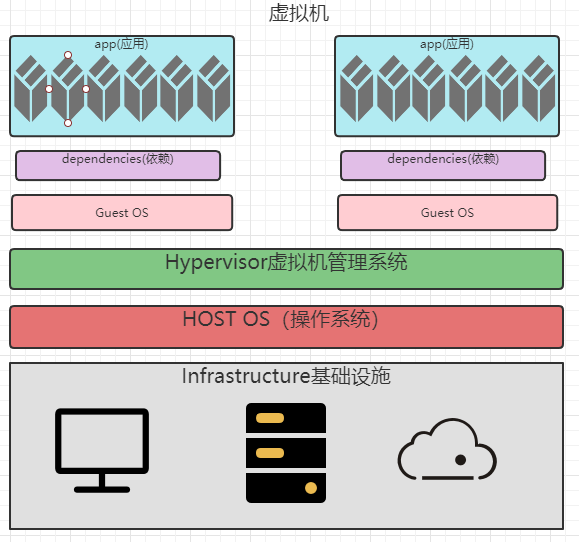


图-11 虚拟机架构

从下到上依次为：

* infrastructure基础设施：这个指的就是你的电脑硬件，可以是个人pc，可以是服务器，可以使云主机。同上，没变化
* HOST OS操作系统：安装使用的主机操作系统，可以是windows，MacOS，linux。同上，没变化。
* Hypervisor虚拟机管理系统：利用Hypervisor，可以在主操作系统之上运行多个不同的从操作系统。也是虚拟机技术关键。
* Guest OS 客户机操作系统：这个表示每一个虚拟机运行时需要启动的操作系统，相当于在宿主机操作系统之上，每个虚拟机又另外配置了自己的操作系统。这个过程是想到消耗资源的，无论是cpu，内存，还是磁盘空间。
* dependencies依赖：运行程序需要用到的依赖，可以是插件，基础服务，语言环境等。这里需要注意的是，每一个虚拟机中的依赖相互是独立的，也就是说即使2个虚拟机都是用同一个jdk环境，那么也是各自配置各自，不能共享。
* app应用：用户使用物理机的应用程序，可以使qq，微信，游戏等。同上，没变化。

最后我们再来看一看docker容器的架构，如图-12所示：

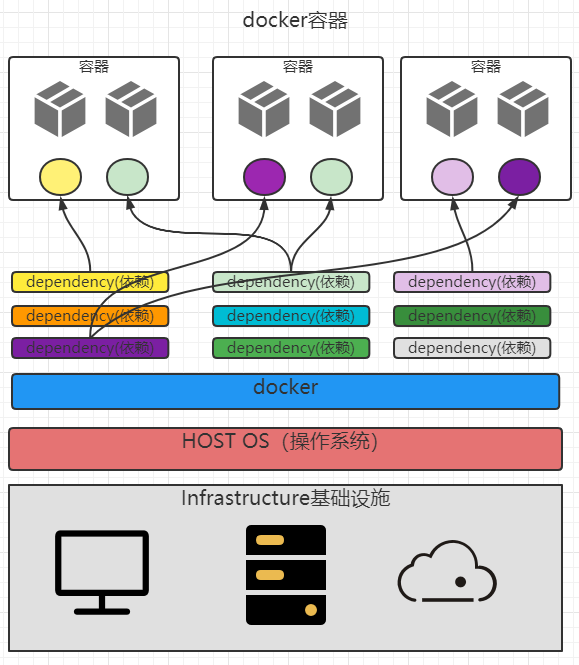


图-12 docker容器架构

这里的结构基本和前面类似，我们只强调和虚拟机区别的地方：

* 没有Hypervisor，而是使用docker管理器，这样一来，不需要为每一个虚拟的容器单独构建客户机操作系统，极大的节省了硬件资源。
* 依赖不在相互独立，而是共享的。这样一来在宿主机上只需要下载安装一次依赖，只要容器使用这个依赖，即使有几万个容器，也都是共享这一个依赖。

综上所述，在漫长的虚拟机话技术发展之后的今天，容器已经成为一个轻便的，不可替代的虚拟化产品，是开发者不得不掌握的技术之一。

## docker的基础命令

docker的命令非常繁琐，选项多，常用选项也多，所以需要在学习docker的精髓知识点之前，必要的步骤就是熟练掌握docker的基础命令。

docker的命令基本语法结构：

docker 子命令 [选项]

### 命令帮助

如果对docker的命令有所疑惑，可以通过一下手段，获取帮助相关信息。

* 子命令查询

如果不了解docker之后可以衔接哪种子命令，可以使用help查看：

docker --help

将会在执行之后看到当前版本docker可以运行的所有子命令，如图-13所示：

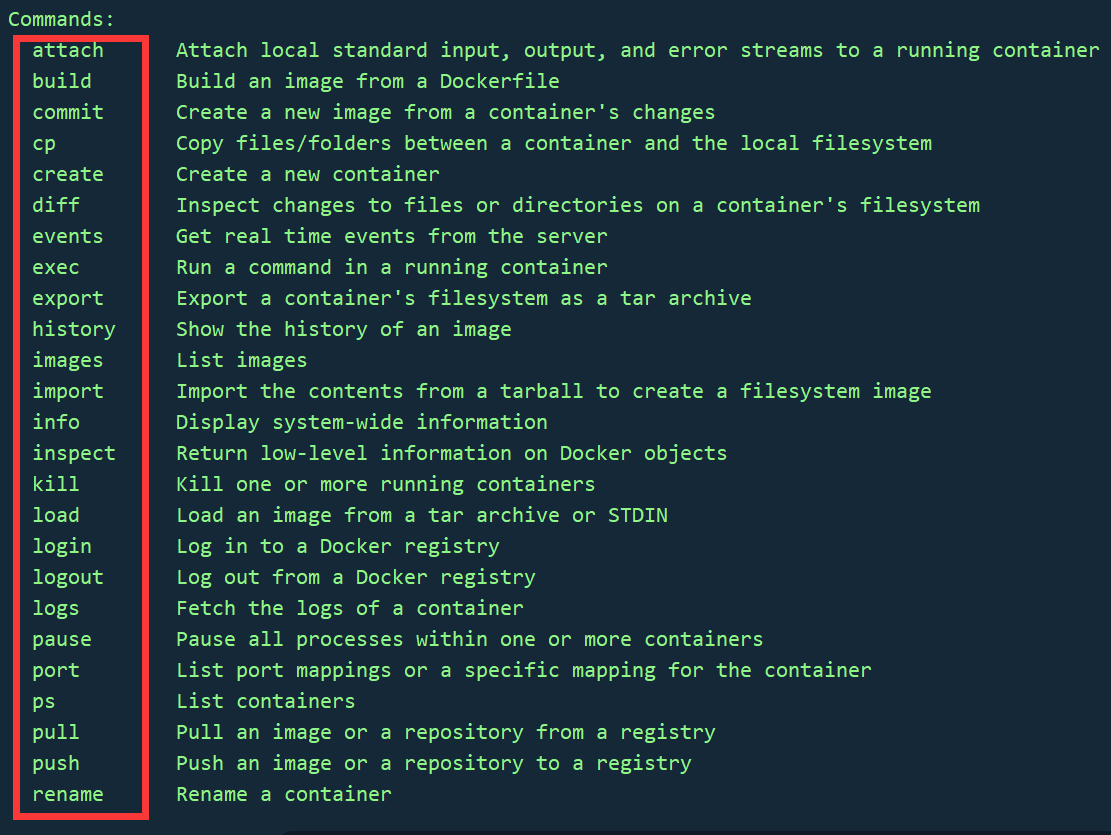


图-13 docker --help

* 子命令帮助

如果想查看具体某一个子命令的相关信息，可以运行help查看：

docker 子命令名称 --help

例如，我们想查看run这个命令的详细信息：

docker run --help

返回结果如图-14所示：

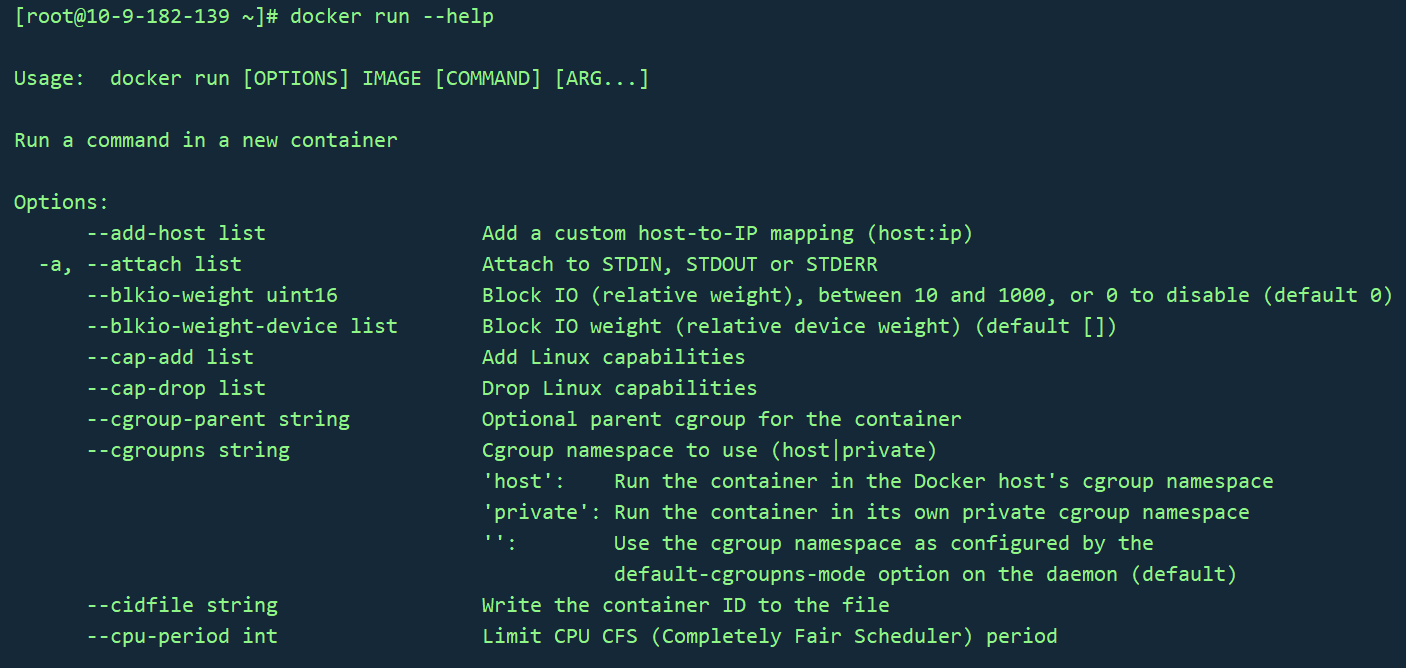


图-14 docker run –help

还有一种方式，也可以让我们更详细的了解docker相关命令内容，就是到官方网站查看。地址为：

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

进入网址之后，可以在左侧看到相关command标签选项，进入选项，点击你想看的命令即可，如图-15所示：

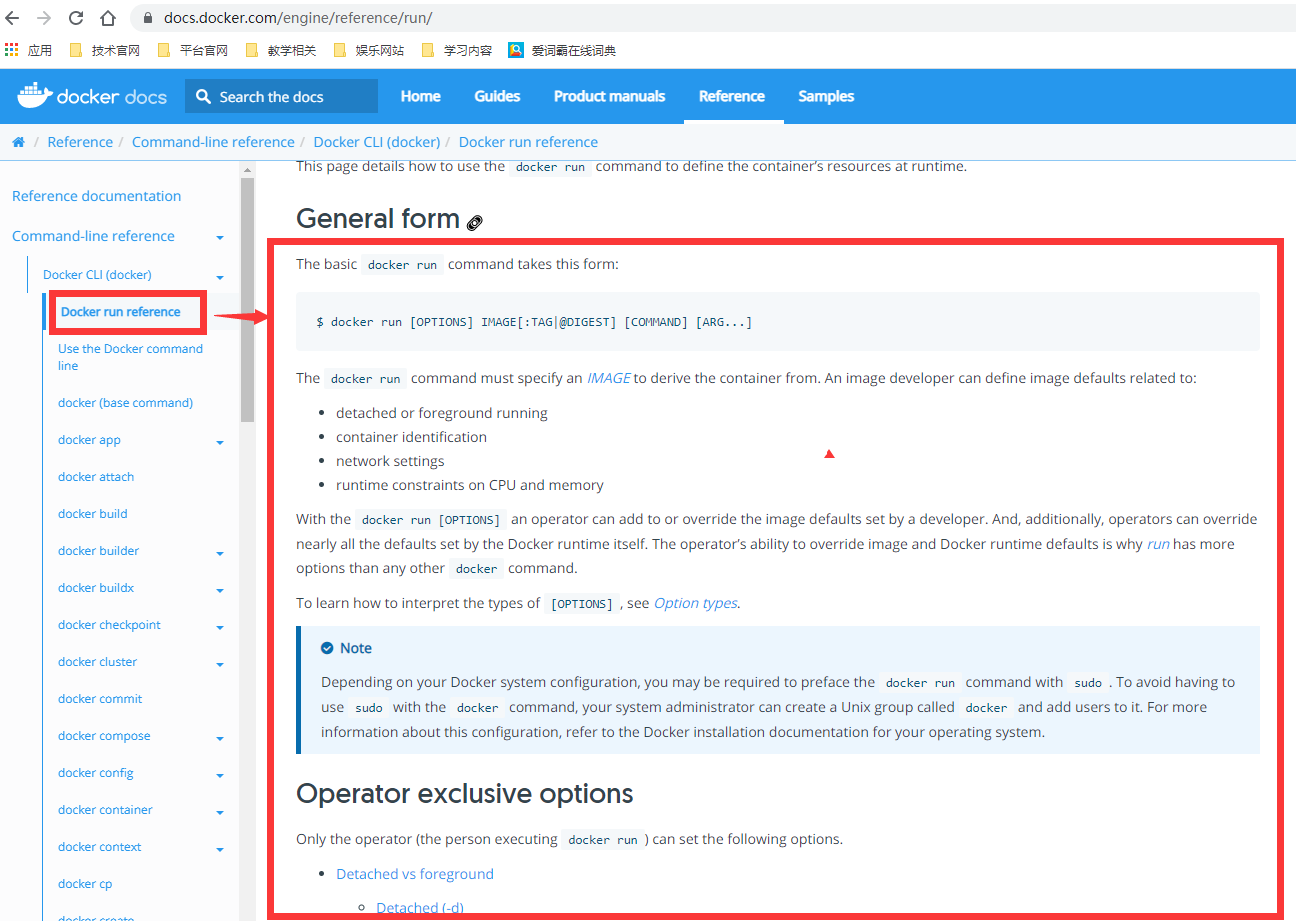


图-15 docker run官方文档

### docker images

docker images主要用来查看当前本地镜像有哪些。例如：

docker images

可以看到返回结果，如图-16所示：

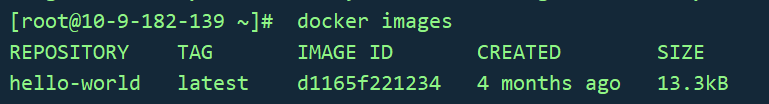


图-16 docker images命令

这个命令的返回结果显示：

* REPOSITORY:镜像仓库名，也叫作镜像名。
* TAG：标签，常用版本号标识仓库，如果是latest就是最新版本。
* IMAGE ID：镜像id。
* CREATED:创建镜像时间。
* SIZE:大小。

调用docker images的常用选项如下：

* -a：显示所有信息。
* -q：只显示镜像id，这个在批量操作镜像文件时，经常用来嵌套使用。

### docker search

我们需要的镜像可以在docker的仓库里进行搜索，使用docker search命令。例如我们可以搜索mysql的镜像。

docker search mysql

搜索结果如图-17所示：



图-17 搜索mysql镜像

这个命令的返回结果显示：

* NAME:镜像名称。
* DESCRIPTION:镜像描述。
* STARS：镜像星级，越高表示越热，使用人越多。
* OFFICIAL:是否官方镜像。
* AUTOMATED：是否支持自动化部署。

如果我们想要了解镜像更多内容，建议到官方仓库docker hub进行搜索。地址是：

<https://hub.docker.com/>

如图-18所示：

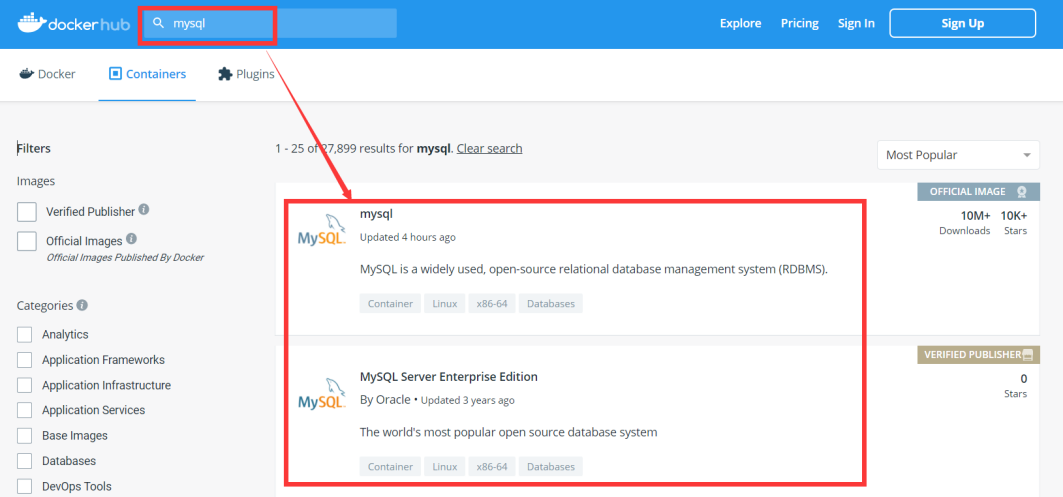


图-18 docker hub搜索mysql镜像

进入到镜像，官方会提供镜像描述，镜像版本tag，和镜像的使用方式。如图-19所示：

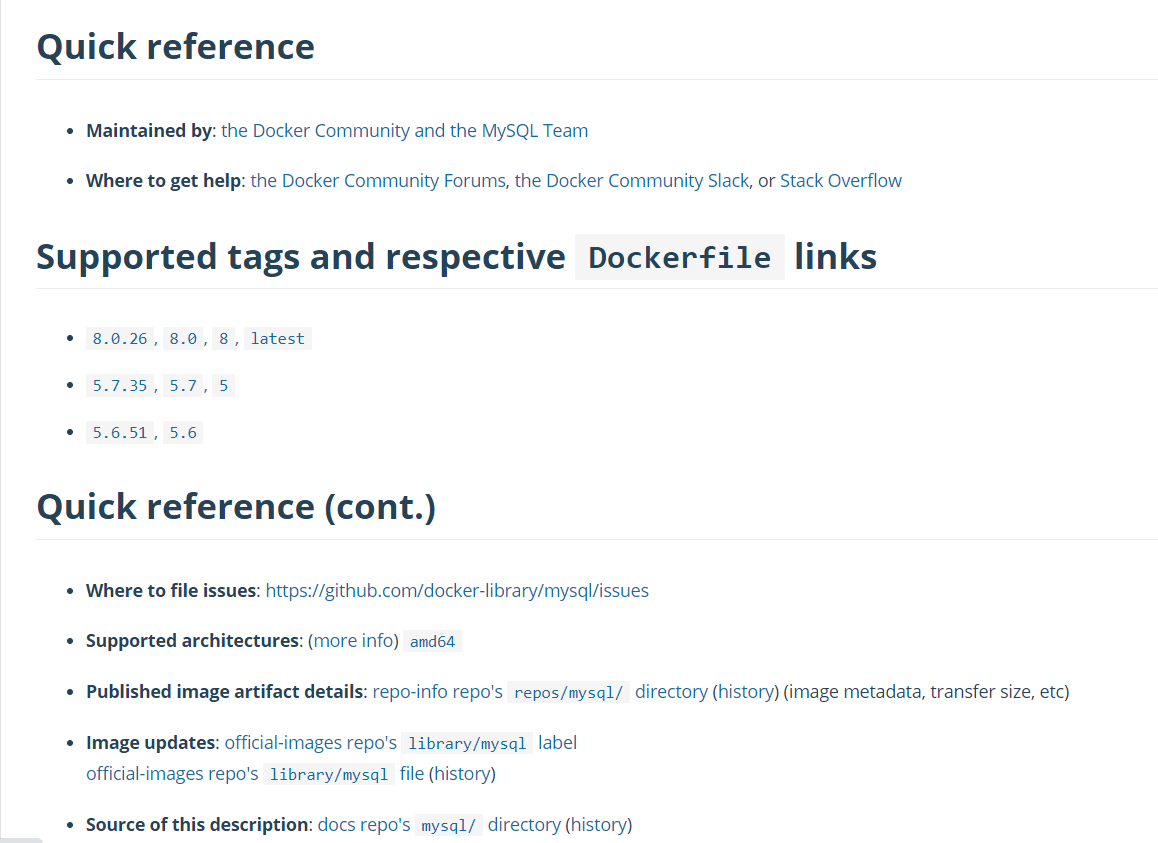


图-19 mysql镜像详细内容

### docker pull

当我们需要用到某个镜像的时候，可以利用pull命令将镜像从远程仓库下载到本地。

docker pull mysql

执行这个命令可以看到如图-20所示：

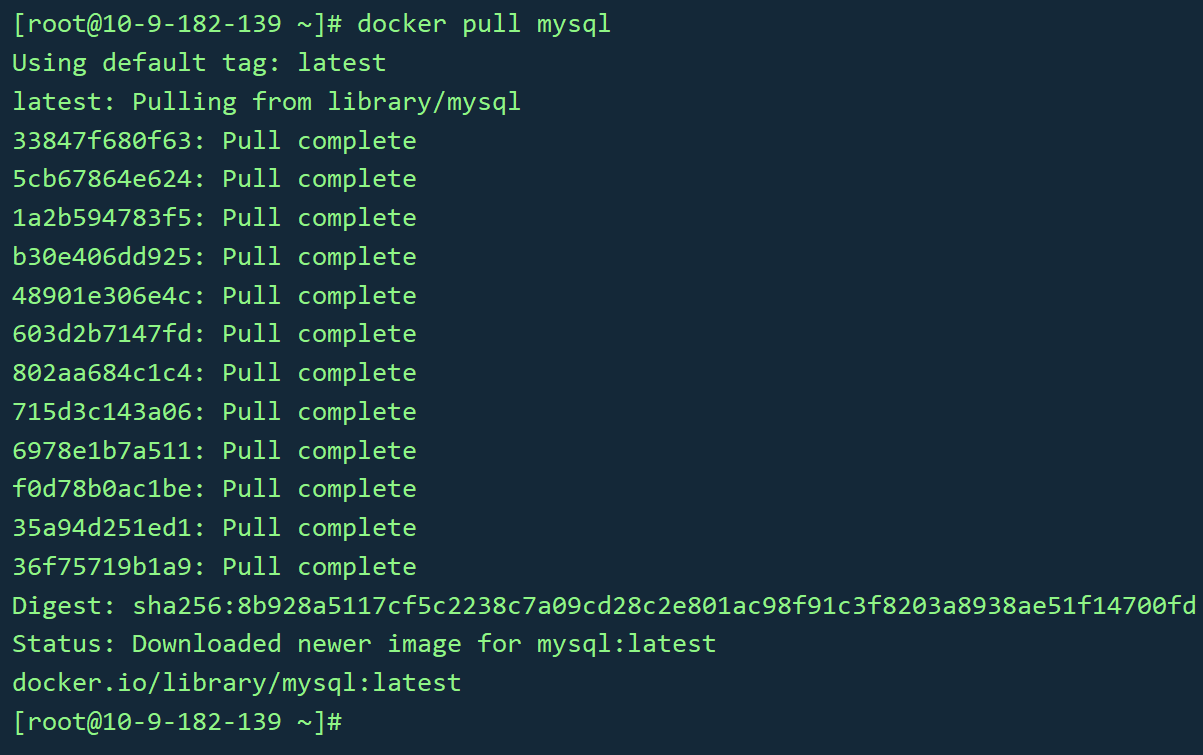


图-20 下载mysql镜像

对返回结果详细解释如下：

* Using default tag: latest

执行pull命令，后面没有跟随具体的tag标签，默认使用最新版本latest，默认最新版本就是从仓库中看到的。我们也可以这样运行命令，指定下载tag标签的mysql：

docker pull mysql:5.7.35

* 33847f680f63: Pull complete ……

前面的表示id，后面表示下载进度，这些都是分层依赖数据，每当下载一个镜像，这个镜像需要的依赖会分层下载到本地，这样如果我们下载多个镜像，有公用的依赖则不会在重新下载。如图-21所示：

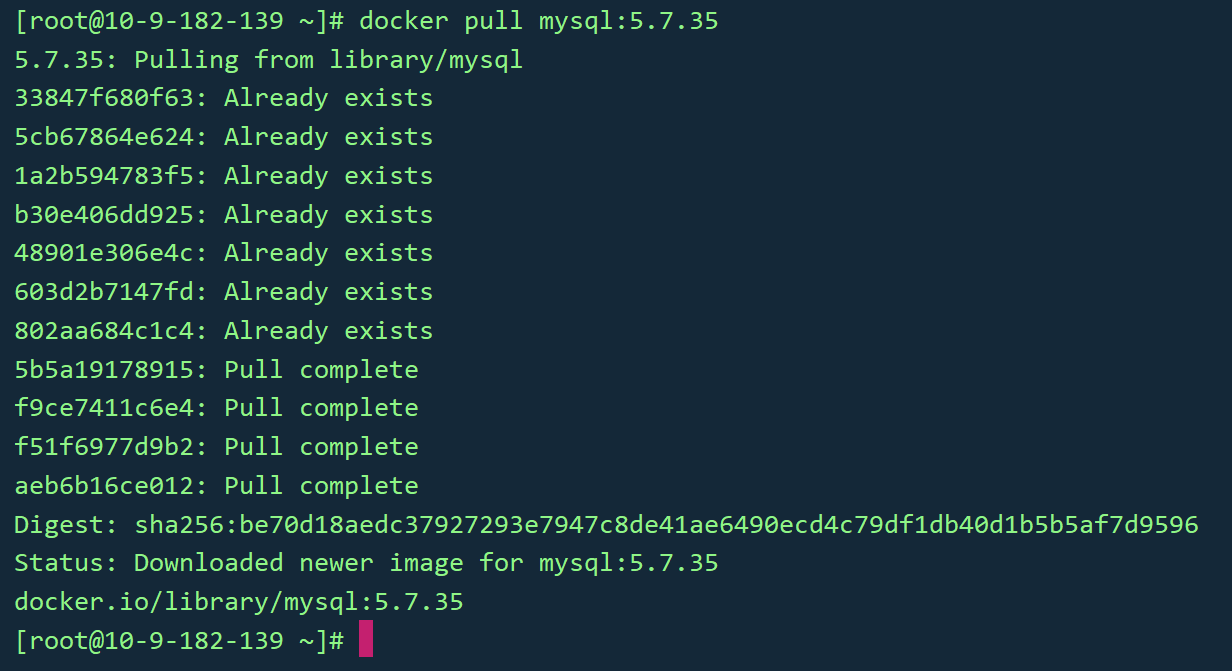


图-21 下载mysql:5.7.35镜像

上图中我们发现有很多依赖是现实Already exists，这就表示某个其他镜像和mysql:5.7.35共享这些依赖。

pull命令的常用选项有：

* -a:表示下载所有。
* -q:表示下载不显示详细进度。

### docker rmi

当下载的镜像越来越多，不需要用到镜像时，可以执行rmi删除镜像命令。

docker rmi d1165f221234

这里可以利用docker images查询到的镜像id进行镜像删除，如图-22所示：

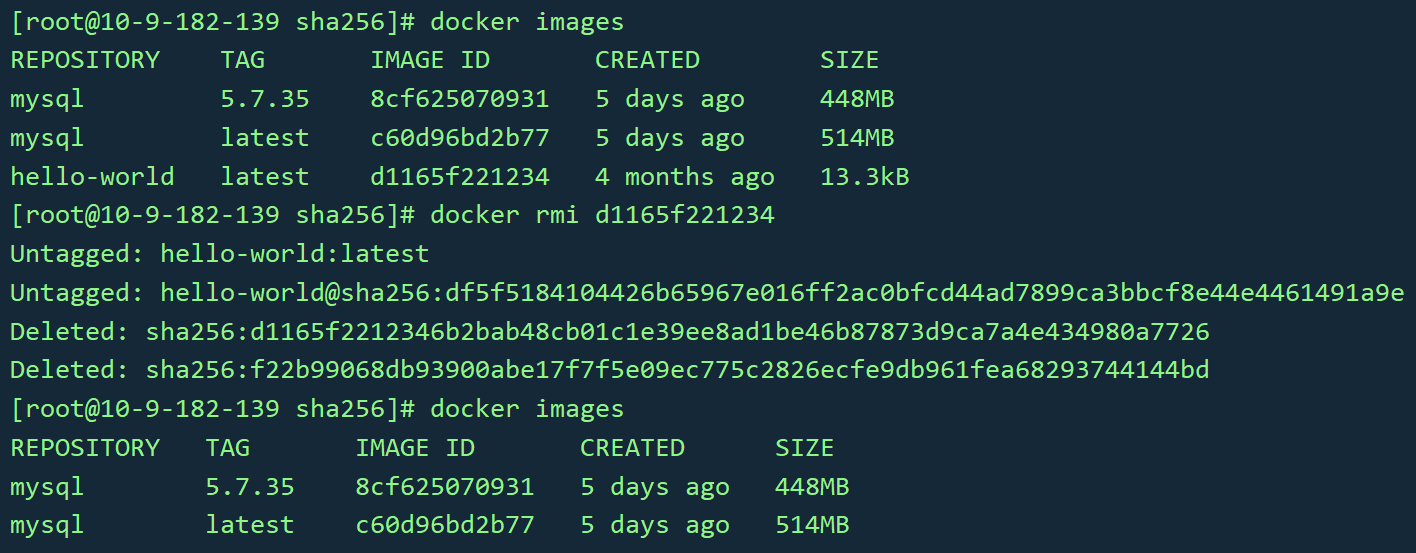


图-22 删除hello-world镜像

rmi命令的选项：

* -f:强制删除这个镜像，无论是否正在使用。

如果不加这个选项，当镜像创建了容器之后，就不能删除了，表示正在使用。

除此以外，我们在删除镜像时还有一种嵌套的命令执行可以删除全部，但是慎用，毕竟镜像下载是需要耗费带宽资源的，这里感受一下这种嵌套。

docker rmi -f $(docker images -qa)

docker images -qa 返回全部的镜像id作为docker rmi -f 的参数使用，相当于我们手动执行docker rmi -f 镜像id1 镜像id2 镜像id3…。如图-23所示：

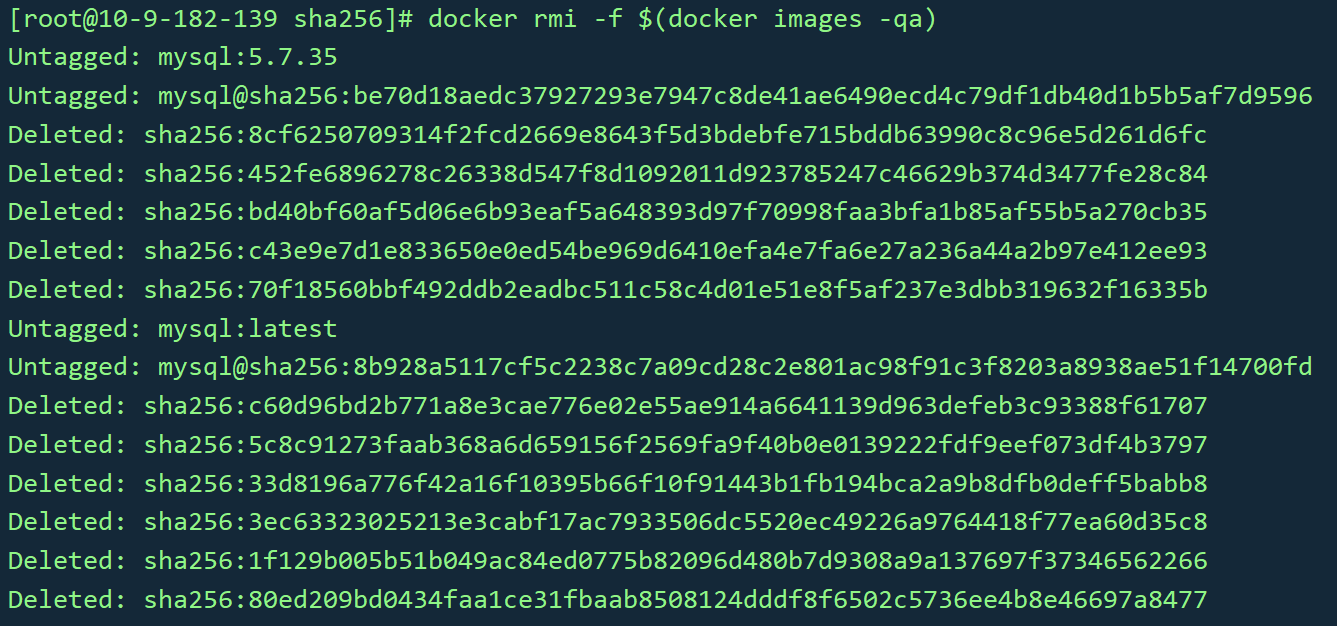


图-23 docker删除所有镜像

到此为止，关于docker镜像相关操作的基础命令，我们就介绍完了，需要大家数量掌握这些命令，接下来我们看看容器相关命令。

### docker run

我们可以利用docker run的命令使用我们下载好的镜像，来创建和启动容器。前面我们介绍过，镜像好似模具，而容器就是利用这模具创建的实例。一个镜像可以创建运行多个容器。

首先我们为了测试容器相关命令先拉取下来一个centos的镜像，这个镜像可以创建一个最小化依赖的centos系统。

docker pull centos:centos7

这样在本地我们就下载好了一个centos的测试镜像，如图-24所示：

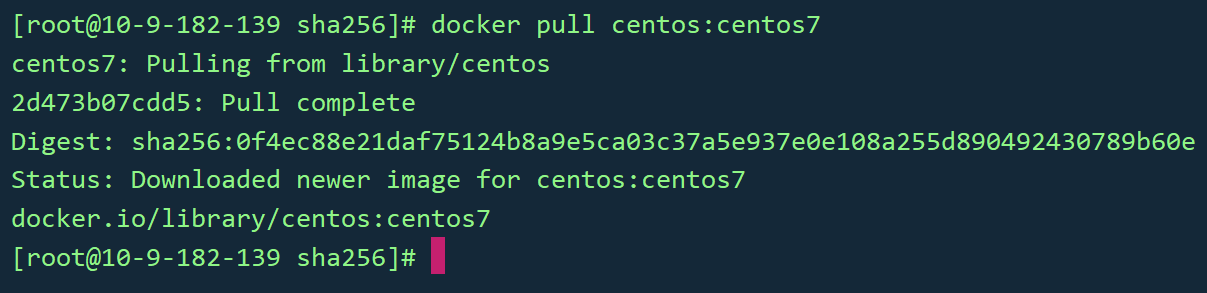


图-24 下载一个centos镜像

run命令的常用选项：

* --name="name" 启动后赋值一个容器名。容器名字不能重复。
* -d 后台方式运行容器。如果不加，运行容器之后会进入到这个容器。
* -i 以交互模式运行容器,一般和-t一起使用
* -t 给启动的容器分配一个终端客户端,如果不使用-i 不起作用。
* -P 随机端口。
* -p 指定端口有四种写法。

p:hostPort:containerPort

ip::containerPort

hostPort:containerPort (常用)

containerPort

这样我们可以运行命令启动一个centos镜像创建的容器：

docker run -it centos:centos7 /bin/bash

执行后我们就进入到这个容器的一个终端控制台，如图-25所示：

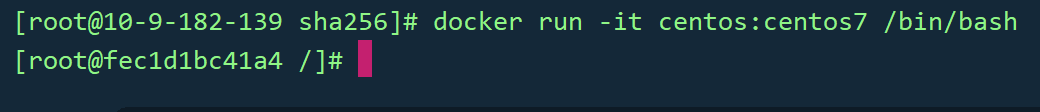


图-25 进入到容器centos中

-it：表示以交互模式启动并且创建终端控制器来操作管理容器，

centos:centos7：表示启动的容器名称和tag标签名字，如果不加tag默认会使用latest，如果本地没有则在run之后自动pull镜像。

/bin/bash: /bin/bash的作用是表示载入容器后运行bash ,docker中必须要保持一个进程的运行，要不然整个容器启动后就会马上kill itself，这个/bin/bash就表示启动容器后启动bash。

运行完毕候我们看到的命令提示符就表示已经进入到这个容器中了。由于这个容器是最小化centos系统，所以很多我们在云服务器中执行的功能否不存在，比如vim编辑器等，如图-26所示：

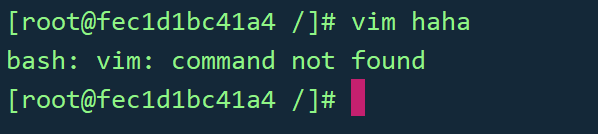


图-26 无法在centos容器使用vim编辑器

### docker ps

当容器启动后，我们在宿主机可以通过ps命令查看容器相关的信息。

docker ps

使用ctrl+P+Q使容器不停止退出，可以运行ps命令，如图-27所示：

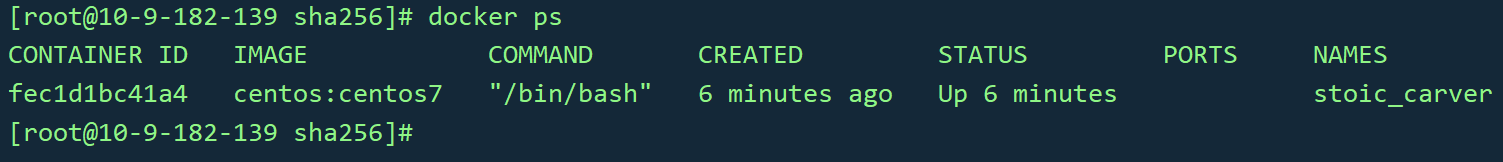


图-27 查看运行容器

其中显示内容：

* container id：容器id，很多操作容器命令都需要用到的参数。
* image：容器创建使用的镜像。
* command：容器中在运行的进程或者命令。
* created：创建时间。
* status：容器状态。
* ports：容器的端口映射情况，这里没有用到端口。
* names：容器的名字，启动没有指定—name选项，会默认使用一个名字。

ps命令中常见的选项如下：

* -a：显示所有容器，如果不加只显示正在启动运行的容器，停止的不会显示。
* -l：显示最近的启动创建的容器。
* -n=[数字]：显示最近n个容器。
* -q：只显示容器id。经常和-a一起使用，获得当前宿主机所有容器id参数集合。

### 容器退出

如果这时候我们想从容器中退出，可以有两种方式：

* exit # 容器停止退出。
* ctrl+P+Q # 容器不停止退出。

### 容器的启动和停止

我们使用镜像创建容器候，可以重启，启动，停止容器。

* docker start (容器id or 容器名) 启动容器。
* docker restart (容器id or 容器名) 重启容器。
* docker stop (容器id or 容器名) 停止容器。
* docker kill (容器id or 容器名) 强制停止容器。

### docker rm

使用镜像创建的容器，可以使用rm进行删除。

首先使用docker ps -a查看当前宿主机启动所有容器。然后运行命令。

docker rm b74009251bbf

这样就可以将对应容器删除，如图-28所示：

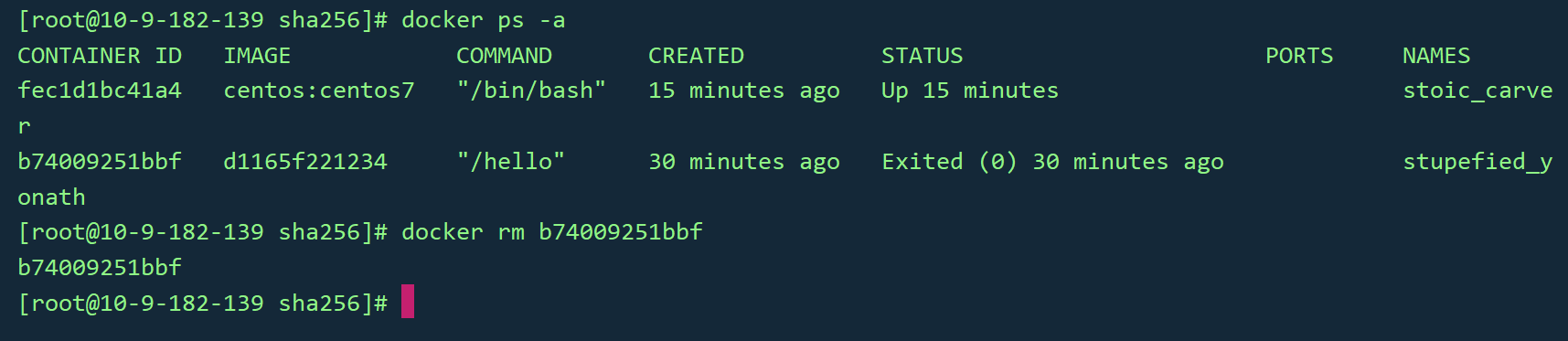


图-28 删除hello-world测试容器

rm的选项有：

* -f:强制删除容器，无论是否运行，如果不加，运行的容器无法删除。

我们还可以利用嵌套，将所有容器强制删除。

docker rm -f $(docker ps -aq)

### docker logs

我们在运行一个容器之后，一般容器内部是会有运行的进程的，否则容器会自动退出。当进程运行时，可以生成一些日志文件，比如tomcat，nginx的相关镜像创建的容器会自动运行这个进程，这时我们在宿主机调用logs命令可以查看对应的日志信息。

首先我们先启动一个tomcat的容器。

* 拉取镜像

docker pull tomcat:9.0

* 运行容器

docker run -d --name="tomcat9-demo01" -it tomcat:9.0

这样，一个在后台运行的，利用tomcat:9.0镜像创建的，名称为tomcat9-demo01的容器就启动了，这时可以使用logs命令查看日志显示，这个命令类似tail -f的作用。

docker logs -f -t --tail 10 ff2927aece15

其中选项为：

-f: 跟踪日志输出。

-t: 显示时间戳。

--tail 10：仅列出最新10条容器日志.

ff2927aece15：容器id。

所以我们可以看到如图-29所示：

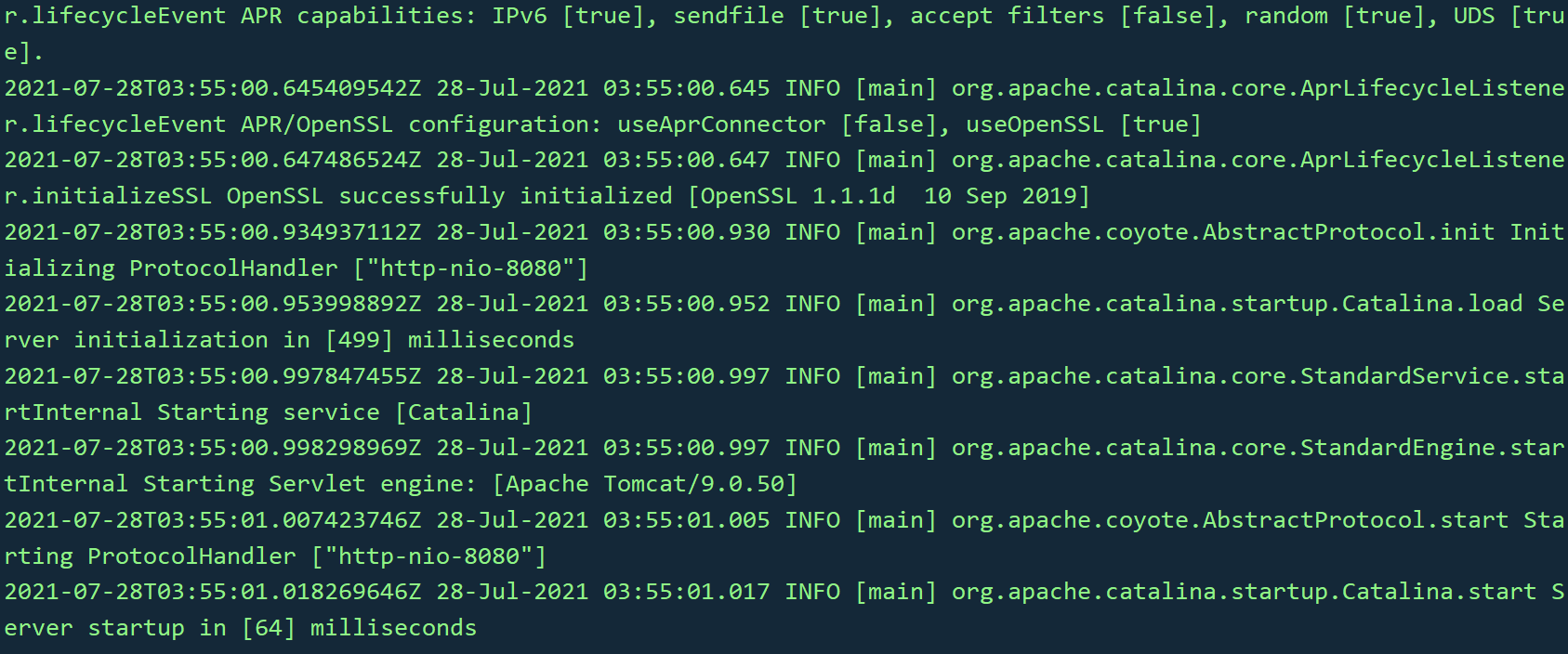


图-29 docker logs查看tomcat日志

不难看出这个显示结果就是tomcat启动日志。

### 进入容器中

当我们启动了一个容器，在宿主机可以进入容器进行命令的调用。这里有2种方法。

* attach

docker attach 2453c8daf4f0

docker attach后面添加容器id，可以进入当前运行的容器，而且不会启动新的进程终端。如图-30所示：

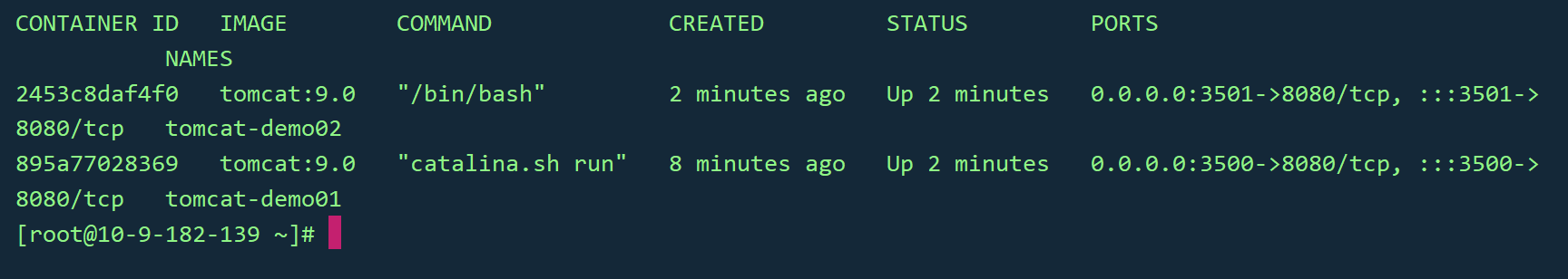


图-30 2个tomcat容器

图中我们有2个tomcat容器，使用的都是tomcat:9.0这个镜像，不同的是，启动之初，一个默认调用的进程命令catalina.sh run也就是tomcat启动进程，一个是/bin/bash支持shell命令提示符进程。

这时候我们使用不同的容器id，attach命令效果不同，比如我们进入第一个，正在运行的是/bin/bash。如图-31所示：



图-31 进入/bin/bash客户终端

可以看到进入command为/bin/bash的tomcat容器，我们attach结果就是看到一个命令提示符。

再比如我们进入另一个tomcat容器，如图-32所示：

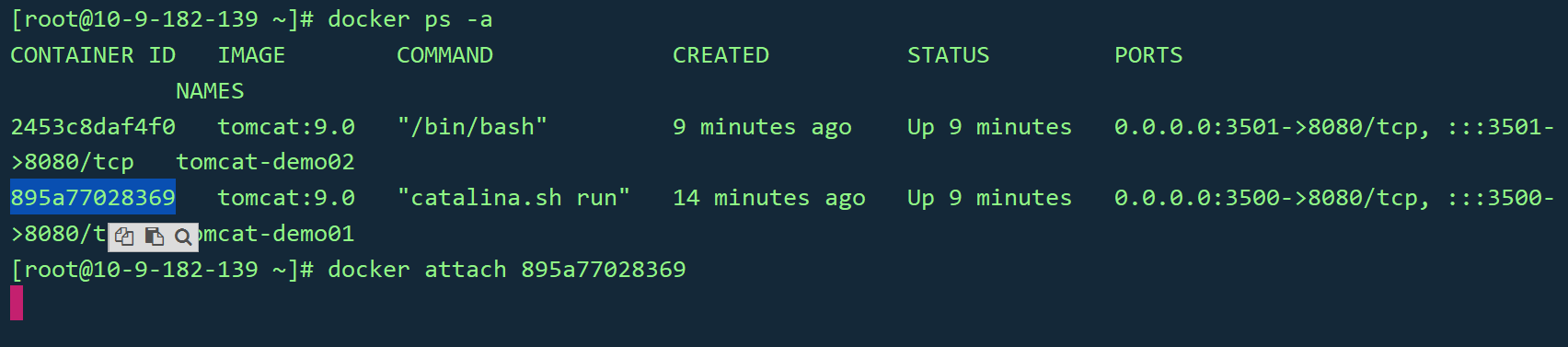


图-32 进入catalina.sh run命令终端

结果就是没有任何显示，因为tomcat运行的进程没有任何输出，所以我们attach就看不到任何内容，如果这个时候tomcat输出了日志，name可以在attach的终端看到。

注意：如果这个时候你使用ctrl+c 强制停止了当前程序，相当于这个容器的catalina.sh run就结束了，那么这个容器就没有任何运行的进程，就是自动关闭。所以可以使用ctrl+P+Q退出。

* exec

和attach附加命令不同，exec是单独运行一各客户端终端。

docker exec -it 895a77028369 /bin/bash

-i：以交互模式打开终端，通常与-t一起使用。

-t：打开一个伪终端，通常和-i一起使用。

895a77028369：容器id。

/bin/bash：exec打开的终端运行的进程，或者命令。

后面我们还会见到更多的exec进入容器的方式。

### docker cp

如果我们运行了一个容器，在容器里面创建的一些文件，想要将文件拷贝到宿主机，可以使用cp命令。

例如，首先我们运行一个centos的镜像容器。

docker run -d -it --name="centos-demo01" centos:centos7 /bin/bash

紧接着我们进入到容器里面。

docker exec -it 0add39d407ba /bin/bash

在容器终端界面，我们配置一个文件并且保存。

touch my-file.txt

退出到宿主机，ctrl+P+Q。调用docker cp命令。

docker cp 0add39d407ba:/my-file.txt /home

0add39d407ba:/my-file.txt：表示容器id中的某个文件绝对路径。

/home：想要把文件拷贝到的宿主机路径。

到此为止，我们就把docker的部分基础命令介绍完毕了，下面我们来看看本章节的作业题目。

## 课堂作业

### docker安装nginx

* 查询镜像

可以search搜索，也可以官网搜索nginx，找到镜像名称和tag

* 拉取镜像

根据搜索的结果，拉取镜像到本地

* 运行容器

使用镜像运行容器，可以根据官网文档提供的如何使用去运行。

* 访问测试

启动后，利用宿主机暴露的端口访问测试nginx

* 扩展题目：进入容器配置nginx文件

exec进入容器执行编辑命令。重启nginx。

### docker安装nginx答案

查询镜像，如图-33所示：

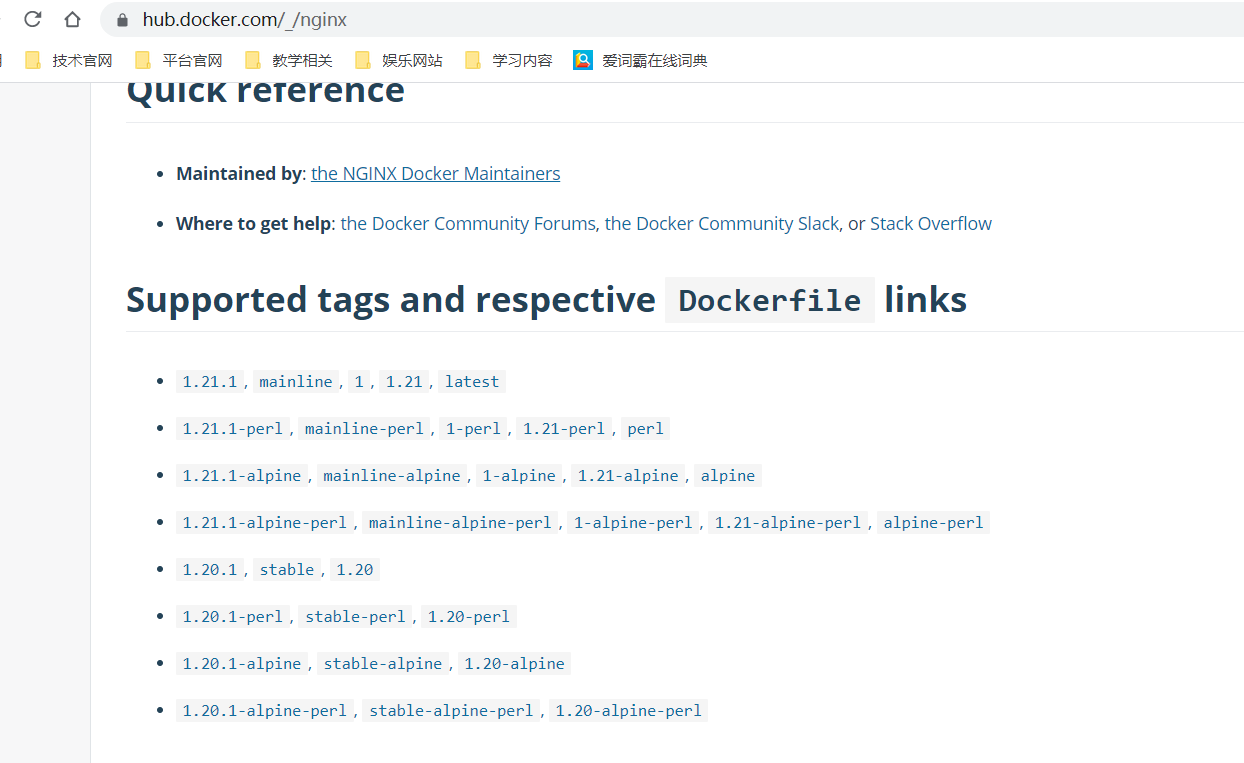


图-33 官方仓库搜索nginx

拉取镜像，如图-34所示：

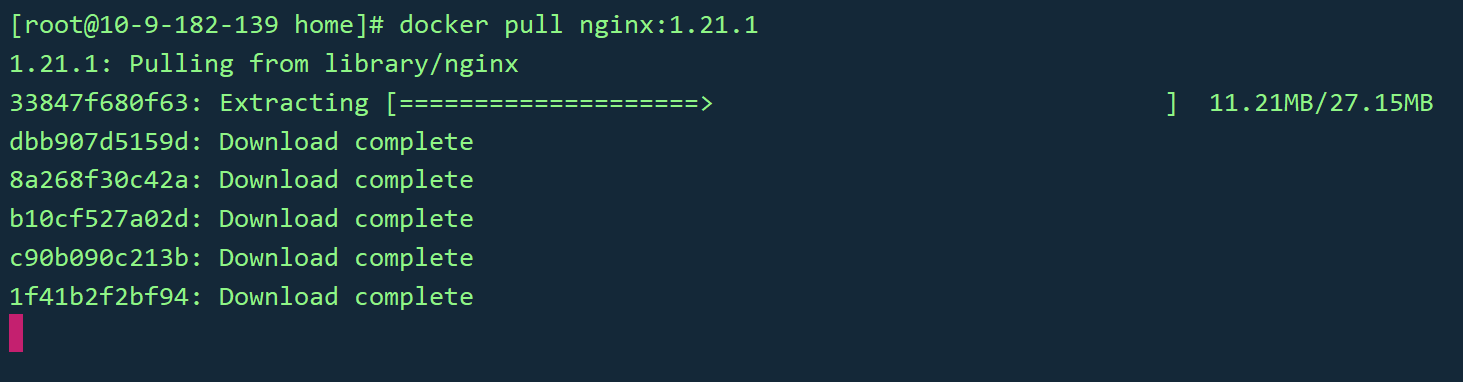


图-34 docker拉取nginx镜像

运行容器，如图-35所示：

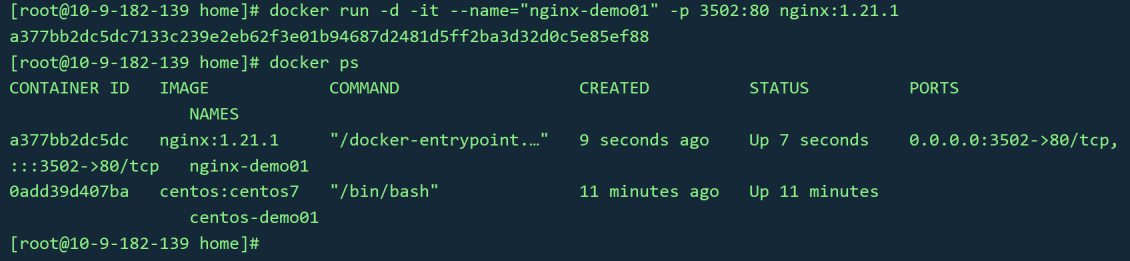


图-35 启动nginx容器

访问测试。如图-36所示：

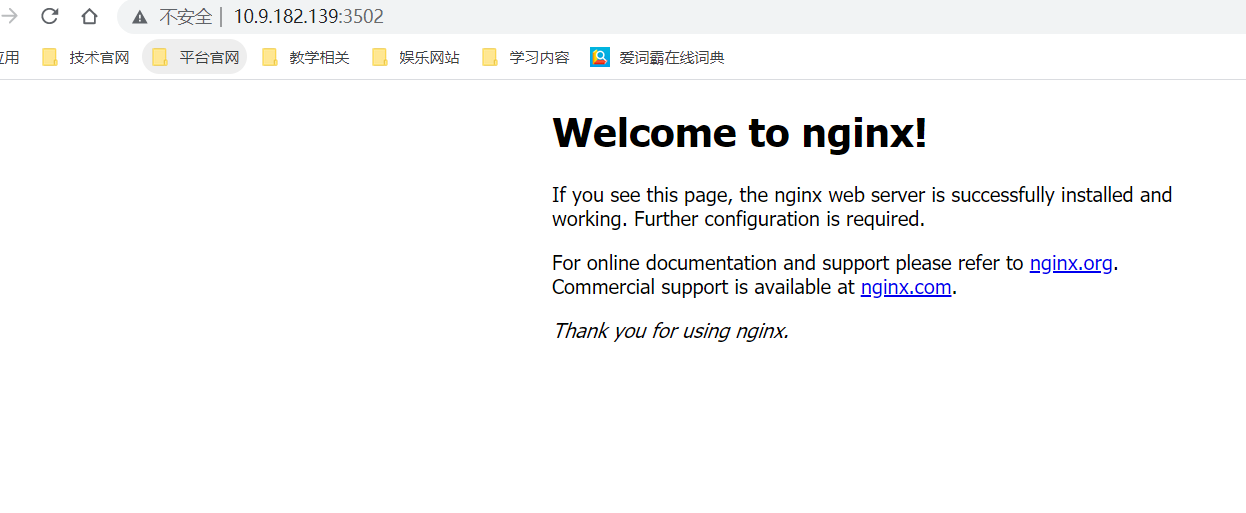


图-36 访问测试nginx

扩展题目：进入到nginx容器。

docker exec -it a377bb2dc5dc /bin/bash