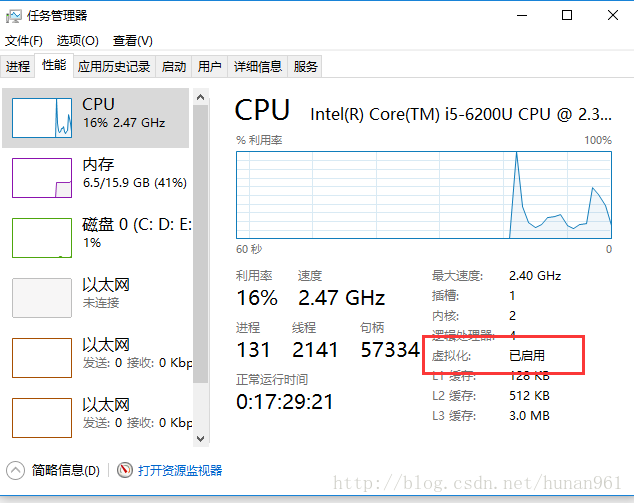
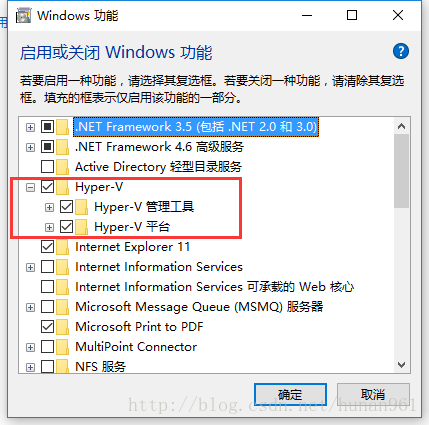
## 1.docker的安装

### 环境准备

Docker for Windows是一个Docker Community Edition（CE）应用程序。Docker for Windows安装包包含了在Windows系统上运行Docker所需的一切。如果你不想装虚拟机，想直接在你的Windows操作系统中安装与学习使用docker，那么你首先得查看你的系统是否满足Docker for Windows的安装与使用要求。

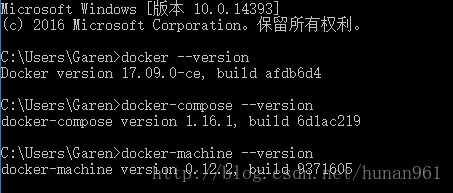
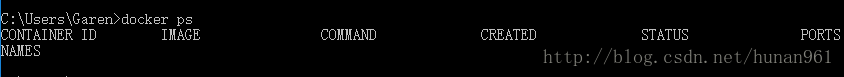
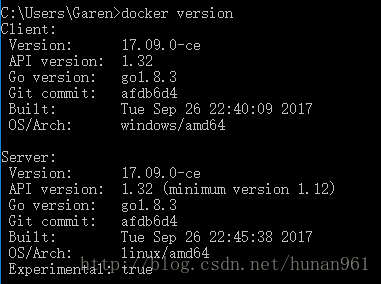
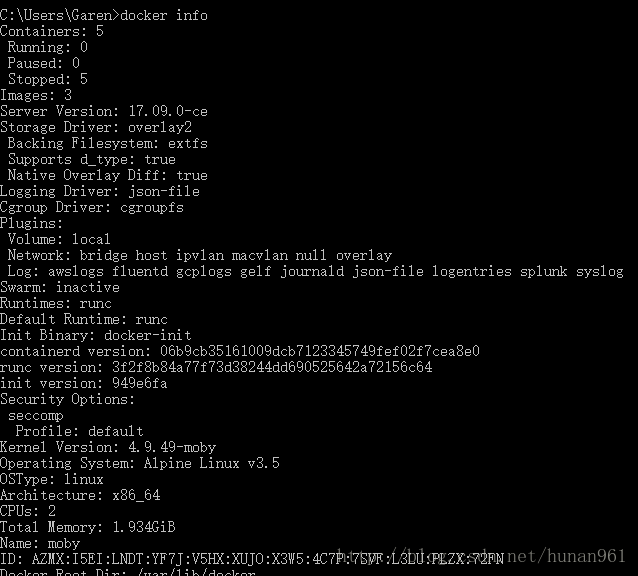
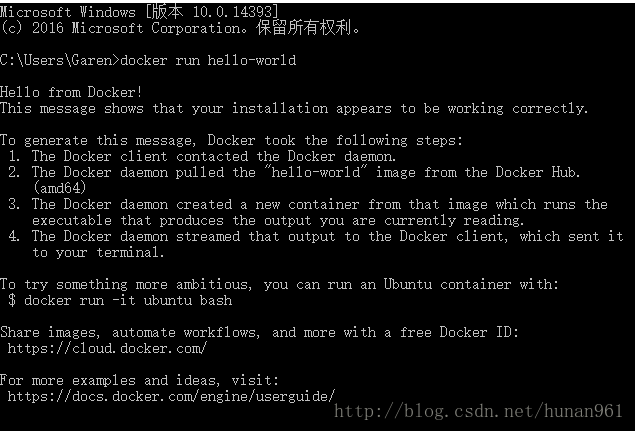
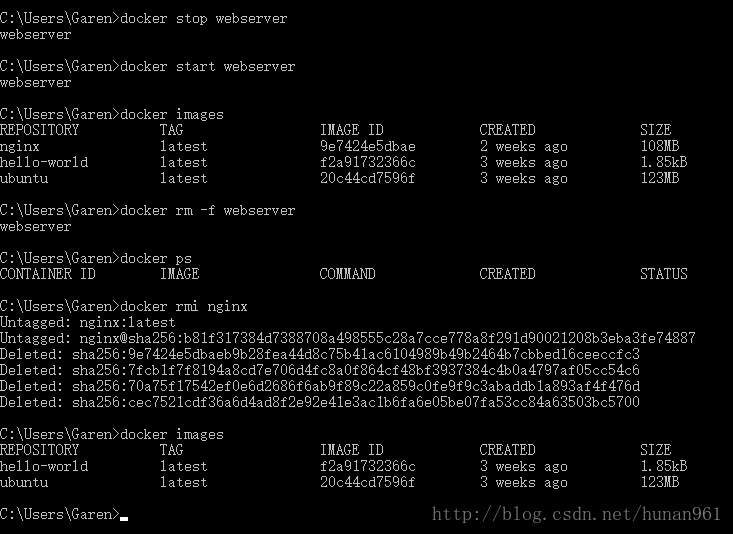
1. Docker for Windows的当前版本运行在64位Windows 10 Pro，专业版、企业版和教育版（1607年纪念更新，版本14393或更高版本）上。Ps:家庭版是不行的，如果你是家庭版，那么一是升级到专业版，破解专业版推荐个地址：<http://blog.csdn.net/SONGCHUNHONG/article/details/78006389> ，二是安装Docker Toolbox，自行网上百度<http://blog.csdn.net/tina_ttl/article/details/51372604>
2. 如果你满足Docker for Windows的环境条件了，那么首先检查电脑的虚拟化开启了没有：进入任务管理器（ctrl+alt+delete），点击性能->cpu ,查看虚拟化是否已启用，如果虚拟化是已禁用，那么你需要重启电脑进入bios开启虚拟化（我们的发的笔记本cpu都是支持虚拟化的，重启时进入bios按esc -> 再按f12 -> 去开启虚拟化）
3. 开启虚拟化重启后，进入任务管理器看虚拟化是否已启用。  
   
4. 然后再是进入电脑的控制面板->程序->启用或关闭Windows功能->把Hyper-v勾上，启用后电脑会重启，后面就可以下载并安装Docker for Windows了。  
   

### 下载安装

1. 进入网址<https://docs.docker.com/docker-for-windows/install/#download-docker-for-windows> 下载并安装。我安装的是稳定版。安装过程没什么要注意的。  
   
2. 启动以后会出现在桌面的右下角区域，鼠标放上去以后显示Docker is running表示启动成功，第一次安装启用好像是会弹出个Docker Cloud登录界面，去注册然后登录，使用和git有点类似，可以pull图像等等  
   

## 2.docker的入门

### 开始使用

1. 检查Docker，Compose和Machine的版本  
   
2. 检查版本信息，并确保docker命令正常工作  
     
     
   
3. 运行docker run hello-world以测试从Docker Hub中拉取图像并启动容器  
   
4. 使用命令docker run -it ubuntu bash运行一个Ubuntu容器，我之前已经拉取过这个容器了，大概是几十兆吧，所以直接启用了，输入exit命令停止容器  
   
5. 运行命令docker run -d -p 80:80 –name webserver nginx 启动一个Dockerized webserver 会下载nginx容器图像并启动它，然后再打开浏览器键入[http://localhost](http://localhost/)  
   
6. 运行docker ps 命令，检查容器的详细信息  
   
7. 停止或移除容器和图像。如果你想停止网络服务器，输入：docker stop webserver然后重新启动docker start webserver。要使用单个命令停止并删除正在运行的容器，请键入： docker rm -f webserver。这将删除容器，但不是 nginx图像。您可以列出本地图像docker images。你可能想要保留一些图片，这样你就不必再从Docker Hub中取出它们了。要删除不再需要的图像，请使用docker rmi后跟图像ID或图像名称。例如docker rmi nginx  
   

## 3.docker的常用配置

### 在PowerShell中设置 tab键自动补全（其实用的都是cmd.exe）

1. 启动一个的PowerShell（即以管理员身份运行）。搜索PowerShell，右键单击，然后选择以管理员身份运行。在PowerShell提示符下键入：  
   Set-ExecutionPolicy RemoteSigned
2. 检查策略设置是否正确，运行：get-executionpolicy  
   应该返回RemoteSigned。
3. 安装posh-dockerPowerShell模块以自动完成Docker命令，键入：Install-Module posh-docker或者，要仅为当前用户安装模块，键入：  
   Install-Module -Scope CurrentUser posh-docker
4. 安装完成后，只能为当前PowerShell启用自动完成功能，输入：Import-Module posh-docker
5. 为了在所有PowerShell会话中保持Tab完成状态$PROFILE，请在PowerShell提示符处输入：

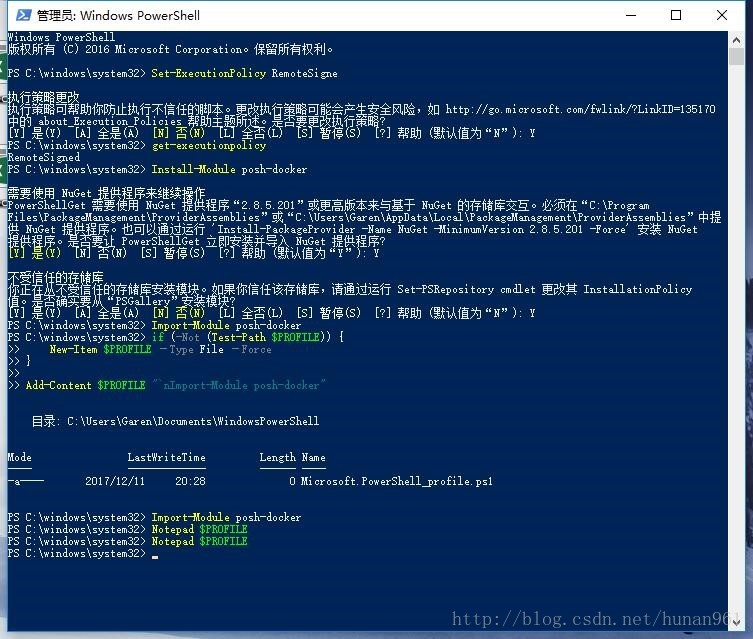
if (-Not (Test-Path $PROFILE)) {

New-Item $PROFILE –Type File –Force

}

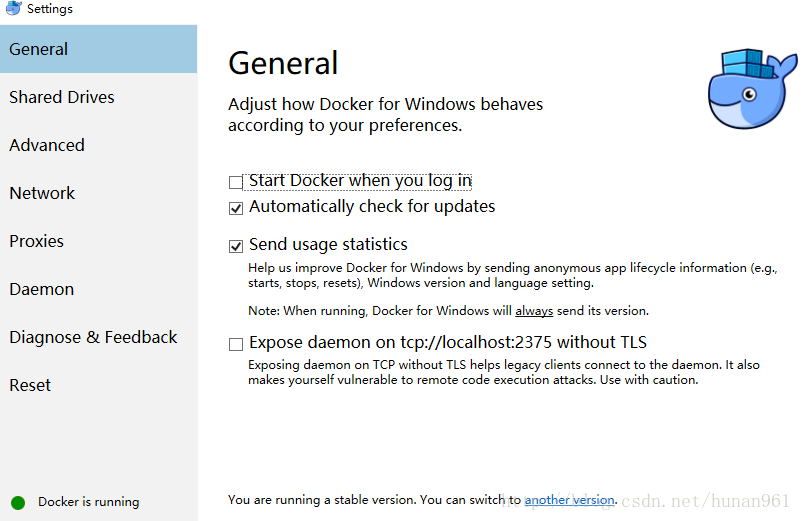
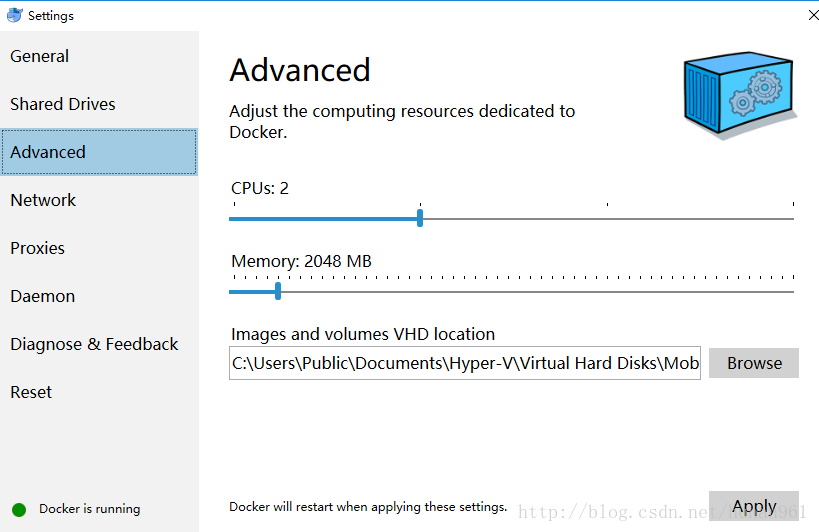
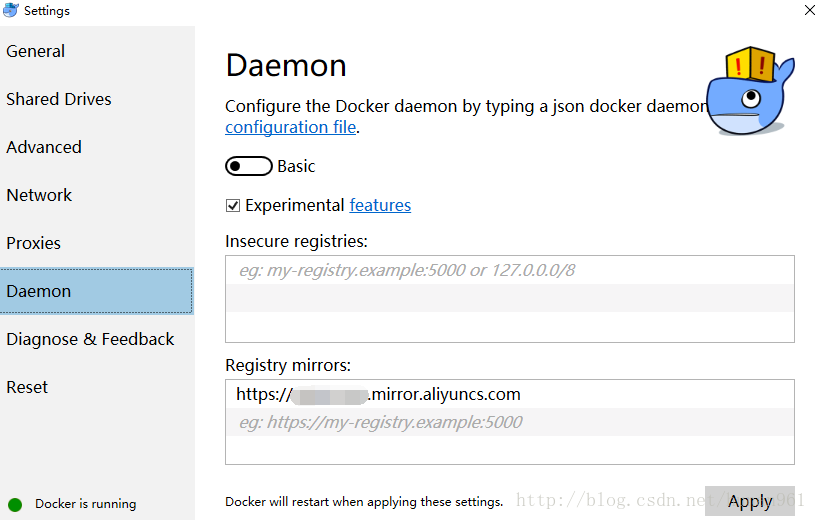
Add-Content $PROFILE "`nImport-Module posh-docker"

* 1
* 2
* 3
* 4

这将创建一个$PROFILE如果不存在，并将此行添加到文件中：  
Import-Module posh-docker  
要检查文件是否已正确创建，或只需手动编辑，请在PowerShell中键入以下内容：  
Notepad $PROFILE  
打开一个新的PowerShell会话。现在，当你键入的前几个字母后按Tab键，Docker命令（如开始，停止，运行及其选项）以及容器和映像名称现在都应该自动完成。  


### Settings

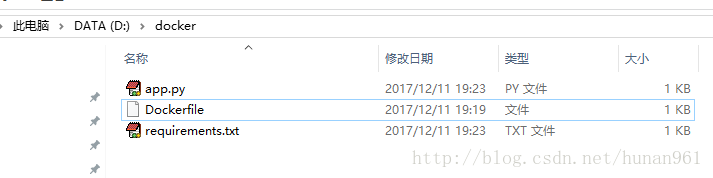
找到右下角的docker图标，右击选择settings进去

1. General：这里是设置docker开机自启，应用程序启动时检查更新，发布使用情况统计信息  
   
2. Advanced：分配cpu数量与内存量  
   
3. Daemon：Docker for windows10 可以配置阿里云镜像，到<https://cr.console.aliyun.com/> 注册一个账户，登录进去后再列表选择加速器，把你的专属加速器地址复制粘贴到Daemon的Registry mirrors中  
     
   

## 4.用Dockerfile定义一个镜像

在过去，如果你要开始编写一个Python应用程序，你的第一步就是在你的机器上安装一个Python运行库。但是，这会造成您的机器上的环境必须满足一定条件以使您的应用程序可以运行。  
使用Docker，你可以将一个可移植的Python运行库作为一个映像，不需要安装。然后，您的构建可以将基础Python镜像与应用程序代码一起包括在内，确保您的应用程序，依赖项和运行时都一起运行。  
这些可移植的镜像是由一个叫做Dockerfile的东西来定义的

### 新建目录文件

创建一个空目录，我取名叫docker，然后分别在里面新建三个文件：Dockerfile，app.py，requirements.txt  


三个文件中的内容分别设计为：

1.Dockerfile：

# Use an official Python runtime as a parent image

FROM python:2.7-slim

# Set the working directory to /app

WORKDIR /app

# Copy the current directory contents into the container at /app

ADD . /app

# Install any needed packages specified in requirements.txt

RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container

EXPOSE 80

# Define environment variable

ENV NAME World

# Run app.py when the container launches

CMD ["python", "app.py"]

2.app.py：

from flask import Flask

from redis import Redis, RedisError

import os

import socket

# Connect to Redis

redis = Redis(host="redis", db=0, socket\_connect\_timeout=2, socket\_timeout=2)

app = Flask(\_\_name\_\_)

@app.route("/")

def hello():

try:

visits = redis.incr("counter")

except RedisError:

visits = "<i>cannot connect to Redis, counter disabled</i>"

html = "<h3>Hello {name}!</h3>" \

"<b>Hostname:</b> {hostname}<br/>" \

"<b>Visits:</b> {visits}"

return html.format(name=os.getenv("NAME", "world"), hostname=socket.gethostname(), visits=visits)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=80)

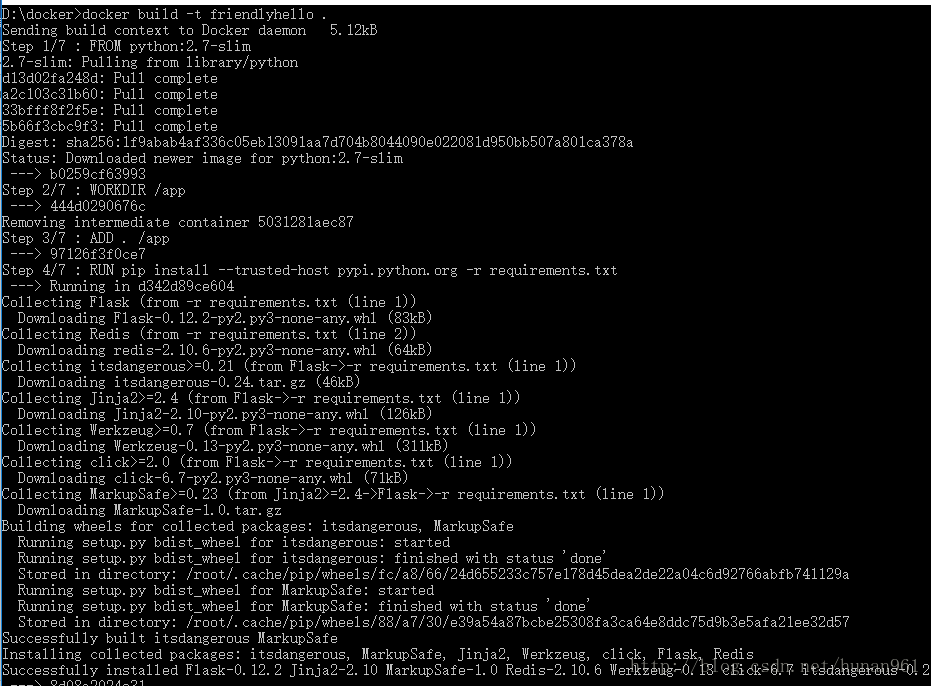
3. requirements.txt：

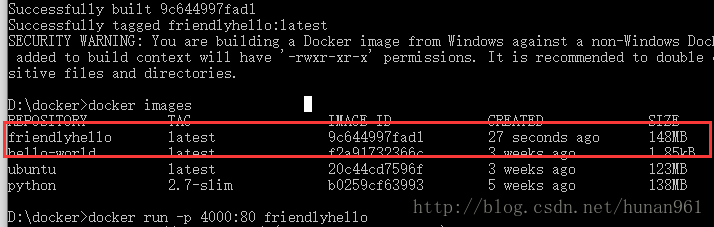
Flask

Redis

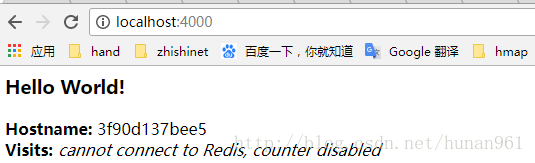
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51

### 构建镜像

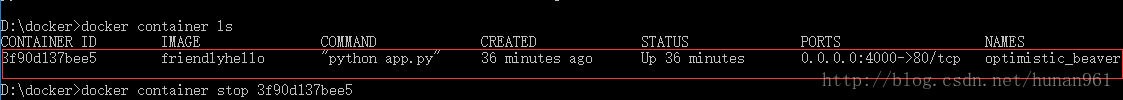
在docker目录下打开cmd.exe运行命令（确保能够找到Dockerfile文件，镜像取名叫friendlyhello）：  
docker build -t friendlyhello .  
ps：千万不要落了上面那行后面的那个点，曾经入过坑的。。命令包括后面的那个点  


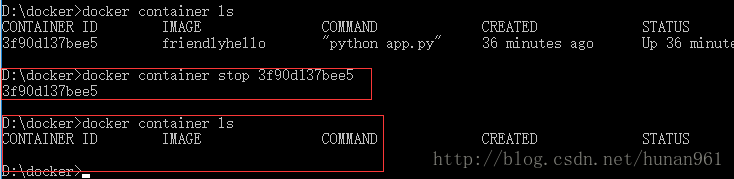
然后再执行命令：  
docker images  


运行镜像程序：  
docker run -p 4000:80 friendlyhello  


可以看到Python正在为应用程序提供消息的[http://0.0.0.0:80](http://0.0.0.0/)。但是，这个消息来自容器内部，它不知道我们将该容器的端口80映射到4000，从而打开URL： [http://localhost:4000](http://localhost:4000/)  


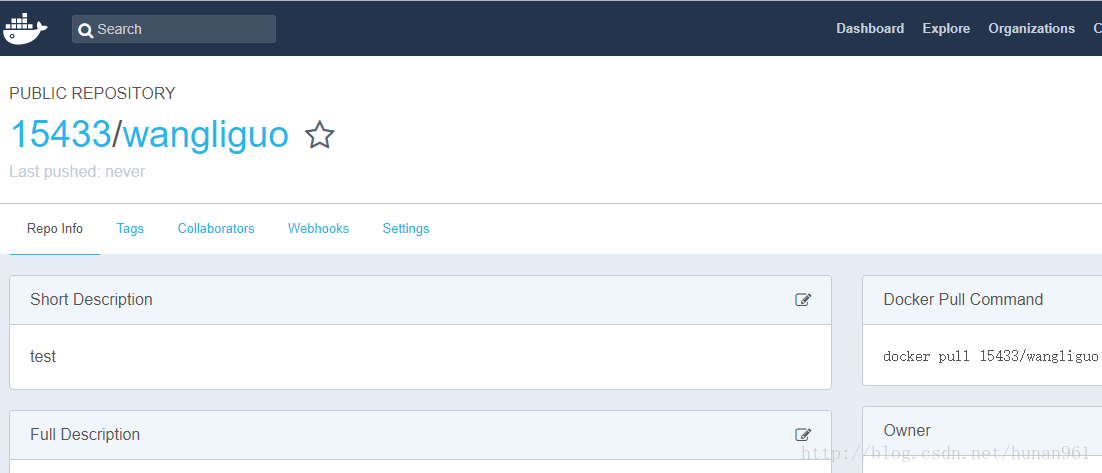
### 停止容器运行

首先在接着上面的操作步骤后按下ctrl+c在终端退出，这并不意味着镜像停止运行了  
键入docker container ls 列出正在运行的容器  
  


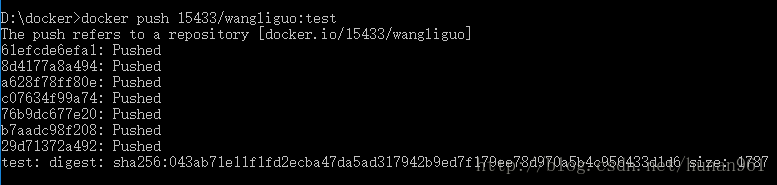
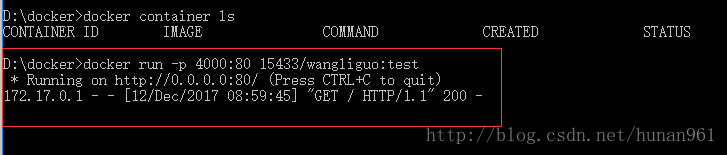
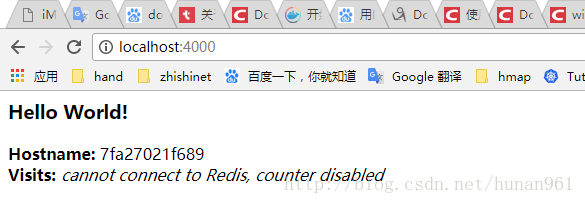
运行命令：  
docker container stop <Container NAME or ID>  
停止容器。否则，在下一步中重新运行容器时，将会收到错误响应。  


## 5.联系Docker Hub的常用操作

### 登录到Docker Hub

前面已经说过了，docker和git的操作有类似之处，所以docker也  
有远程仓库，如果前面已经注册过并登录了docker cloud，那么  
访问网址：[https://hub.docker.com](https://hub.docker.com/) 在里面创建存储库，否则先  
注册吧。  


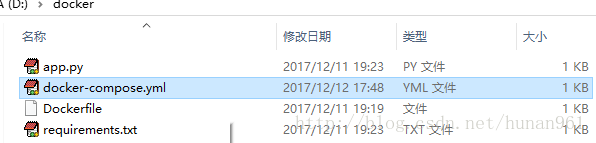
### push镜像

前面在本地创建了一个friendlyhello的镜像，现在要把它push到  
自己的docker hub的存储库中去，首先：  
1. 登录docker hub （我已经登录过了。。再登录一次吧）  
  
2. 标记镜像：  
把镜像放入wangliguo存储库并标记为test  
这里写图片描述  
3. 查看镜像：  
  
4. 发布镜像（推送镜像）  
  
5. Docker Hub上查看镜像：  
  
6. 从远程存储库中提取并运行镜像：现在当远程存储库有了镜像后，就可以从远程存储库提取并运行了  
  


## 6.服务

在分布式应用程序中，应用程序的不同部分被称为“服务”。例如，想象一个视频共享站点，它可能包括用于将应用程序数据存储在数据库中的服务，用于用户上传东西的视频转码服务，为前端服务等等。  
服务实际上只是“生产中的容器”。服务只运行一个镜像，但它编码镜像运行的方式 - 应该使用哪个端口，容器应该运行多少个副本，以便服务具有所需的容量，以及等等。缩放服务会更改运行该软件的容器实例的数量，从而为流程中的服务分配更多的计算资源。  
使用Docker平台定义，运行和扩展服务非常简单 - 只需编写一个docker-compose.yml文件即可。

### 创建一个docker-compose.yml文件

  
键入：

version: "3"

services:

web:

# replace username/repo:tag with your name and image details

image: 15433/wangliguo:test

deploy:

replicas: 5

resources:

limits:

cpus: "0.1"

memory: 50M

restart\_policy:

condition: on-failure

ports:

- "80:80"

networks:

- webnet

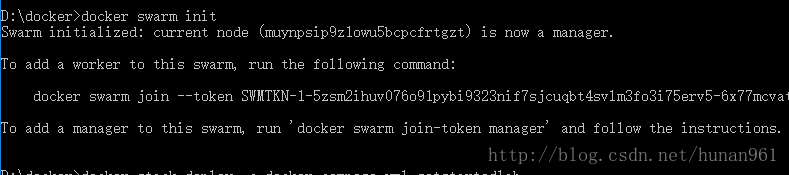
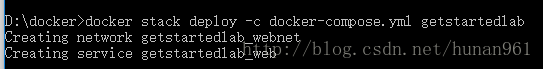
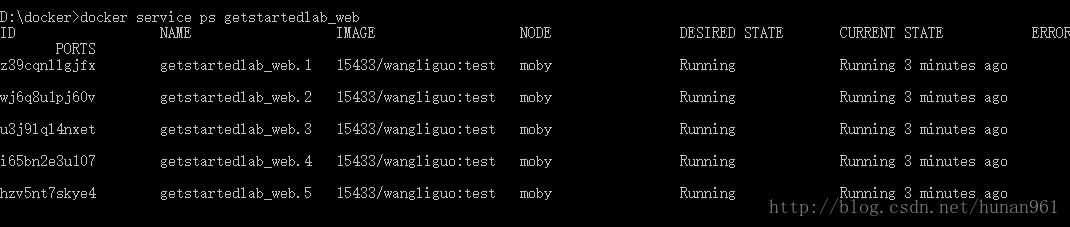
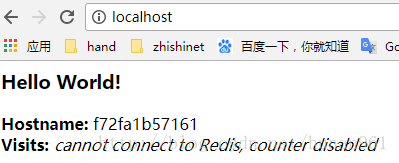
networks:

webnet:

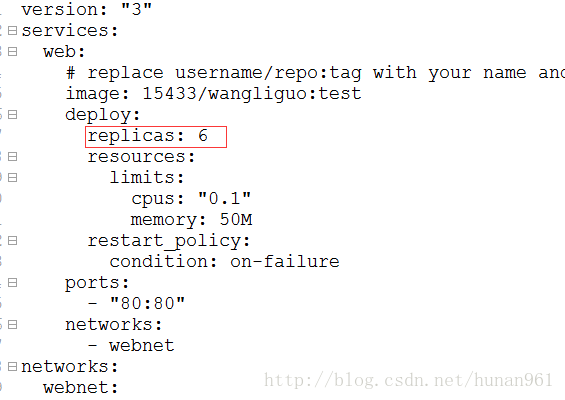
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

拉取的是上个步骤中的存储库的tag。  
运行该镜像的5个实例作为一个服务调用web，限制每个使用，最多10％的CPU（跨所有核心）和50MB的RAM。  
如果一个失败，立即重新启动容器。  
将主机上的端口80映射到web端口80。  
指导web容器通过一个负载平衡的网络共享80端口webnet。（在内部，容器本身将web在临时端口上发布到 端口80）。  
webnet使用默认设置（这是一个负载平衡覆盖网络）定义网络。

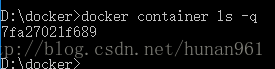
### 运行新的负载均衡应用程序

先运行命令：docker swarm init  
  
然后再运行命令：docker stack deploy -c docker-compose.yml getstartedlab  
给它取名叫getstartedlab  
  
服务堆栈在这台主机上运行了5个部署镜像的容器实例  
运行命令查看：docker service ls  
  
在服务中运行的单个容器称为任务，可以看到上面有个getstartedlab\_web的服务  
运行命令docker service ps getstartedlab\_web 查看此服务下的任务：  
  
如果只列出系统中的所有容器，也会显示任务，但不会被服务过滤：  
运行命令：docker container ls -q  
  
然后打开浏览器，键入[http://localhost](http://localhost/)  
  
点击刷新多次，可以发现Hostname的更改，以循环方式选择5个任务中的一个来响应。容器ID将与前一个命令（docker container ls -q）的输出相匹配。

### 更改应用程序

比如更改docker-compose.yml中的replicas值，保存更改并重新运行docker stack deploy命令来更新应用程序：  
  
运行命令：  
docker stack deploy -c docker-compose.yml getstartedlab  
这里写图片描述  
Docker会做一个就地更新，然后重新运行docker container ls -q以查看重新配置的已部署实例  
  
可以看到之前是6个，现在是7个，刚好多了一个任务  
ps:电脑中还运行这之前步骤中从docker hub中拉取并运行着的那个任务，所以会看到6个和7个

### 关闭应用程序和群

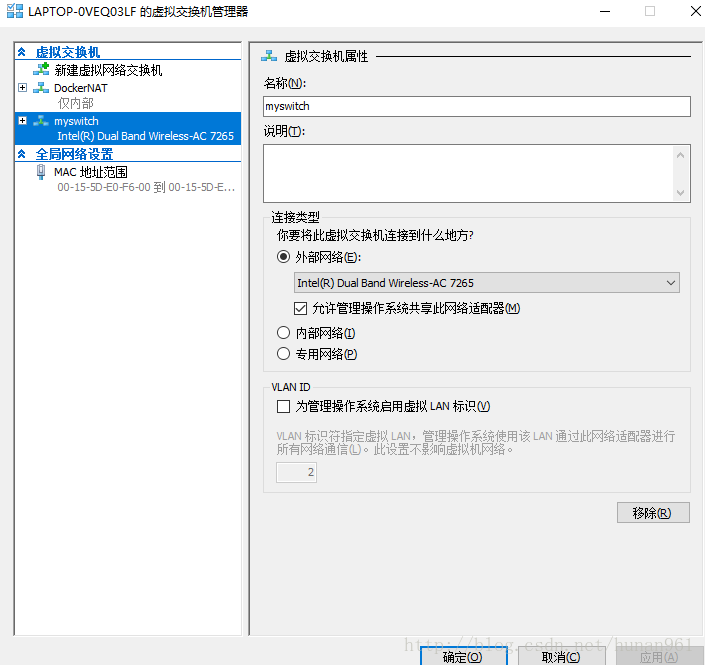
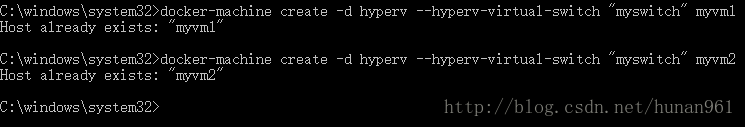
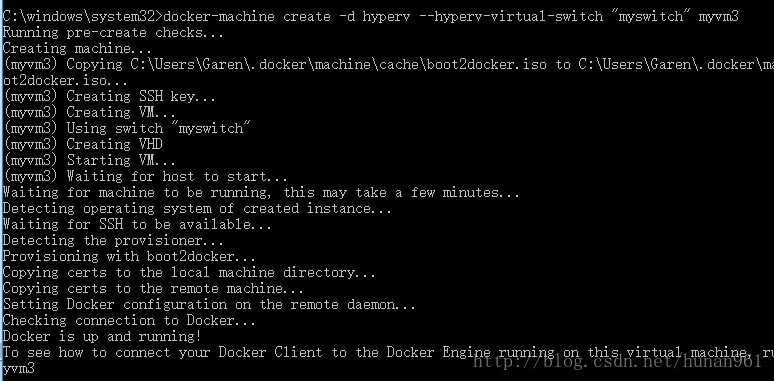
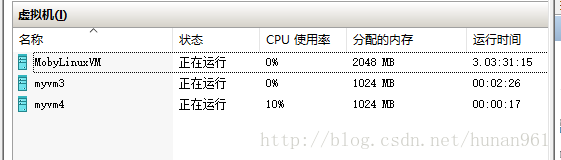
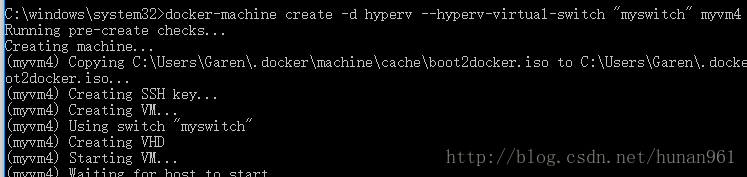
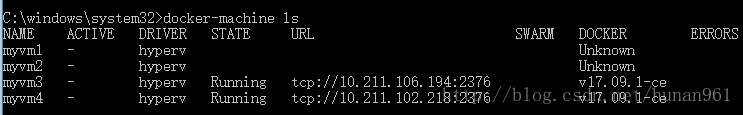
关闭应用程序docker stack rm getstartedlab  
  
关闭群docker swarm leave –force  
这里写图片描述  
现在服务中的任务都关闭了以后再运行命令：  
docker container ls -q  
  
这就是上面说的那个之前步骤中从docker hub中拉取并运行着的那个任务

## 7.集群

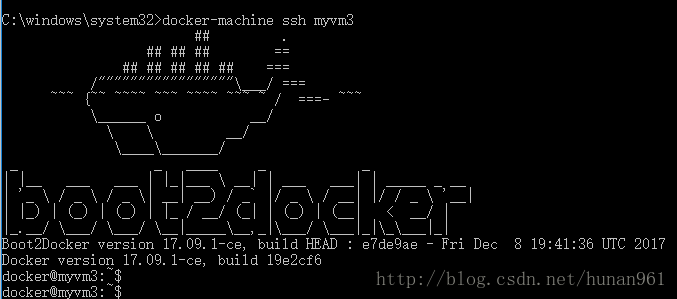
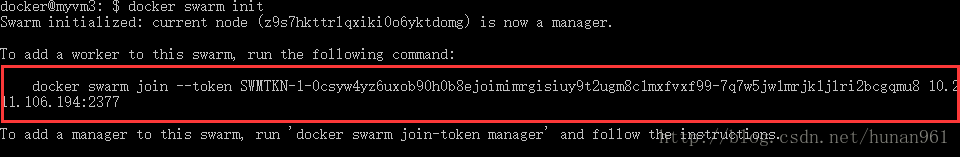
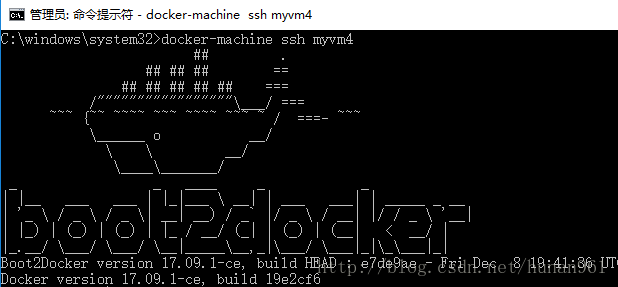
### 了解集群

swarm是运行Docker并加入到一个集群中的一组机器。但是现在它们将由群集管理器在群集上执行。群体中的机器可以是物理的或虚拟的。加入群体后，他们被称为节点。  
Swarm管理人员可以使用多种策略来运行容器，比如“最空的节点”（emptiest node） - 它使用容器填充最少使用的机器。或“全局”，这确保了每台机器只能得到指定容器的一个实例。您可以指示swarm manager在Compose文件中使用这些策略。  
群体管理者是群体中唯一可以执行你的命令的机器，或者授权其他机器作为工作者加入群体。工人提供能力，并没有权力告诉任何其他机器可以做什么和不可以做什么。  
到目前为止，之前都是在本地机器上以单主机模式使用Docker。但是Docker也可以切换到群集模式，这就是使用群集的原因。启用群模式使当前机器成为群管理器。则Docker将运行您正在管理的群集上执行的命令，而不仅仅是在当前的机器上。

### 创建一个集群

一个群由多个节点组成，可以是物理机或虚拟机。基本的概念很简单：运行docker swarm init启用群模式，使当前的机器成为群管理器，然后docker swarm join在其他机器上运行 ，让它们作为工人加入群体。下面将使用虚拟机快速创建一个双机群集，并将其变成群集。  
步骤：  
1. 以管理员运行cmd.exe 这里必须是管理员运行，不然后续操作权限不够  
  
2. 运行docker swarm init启用群模式，使当前的机器成为群管理器，然后docker swarm join在其他机器上运行 ，让它们作为工人加入群体。  
3. 启动Hyper-V管理器  
点击开始◊windows管理工具◊Hyper-V管理器  
  
4. 单击右侧菜单中的虚拟交换机管理器  
  
5. 单击创建类型为外部网络的虚拟交换机，给它的名称myswitch，并检查框共享您的主机的活动网络适配器  
  
6. 使用节点管理工具创建几个虚拟机docker-machine：  
docker-machine create -d hyperv –hyperv-virtual-switch “myswitch” myvm1  
docker-machine create -d hyperv –hyperv-virtual-switch “myswitch” myvm2  
  
之前我已经创建过了，所以我现在再新建两个：myvm3 和 myvm4  
docker-machine create -d hyperv –hyperv-virtual-switch “myswitch” myvm3  
docker-machine create -d hyperv –hyperv-virtual-switch “myswitch” myvm4  
  
  
  
7. 现在创建了两个虚拟机，分别命名为myvm3和myvm4。使用命令  
docker-machine ls 列出机器并获取其IP地址。  


### 初始化群并添加节点

1. 先使用命令docker-machine ssh myvm3  
     
   然后让myvm3 成为一个管理员：docker swarm init  
   注意红框中的内容，这是后面的myvm4 加入集群要执行的命令  
     
   运行命令：docker node ls  
   这里写图片描述  
   可以看到myvm3 已经成为管理员了
2. 以管理员身份再运行一个cmd.exe.然后运行命令：docker-machine ssh myvm4  
     
   然后再运行命令：（这就是上一页中图片里红框中的内容，下面的token是我这里的，正常运行到这里的时候是去myvm3的docker swarm init命令中把红框框位置里的命令复制粘贴过来执行）

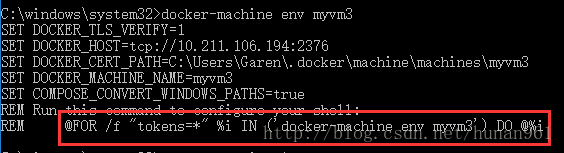
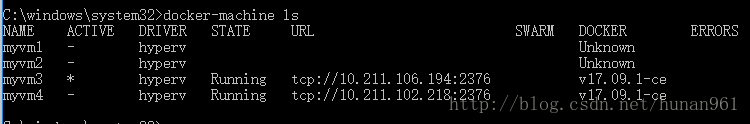
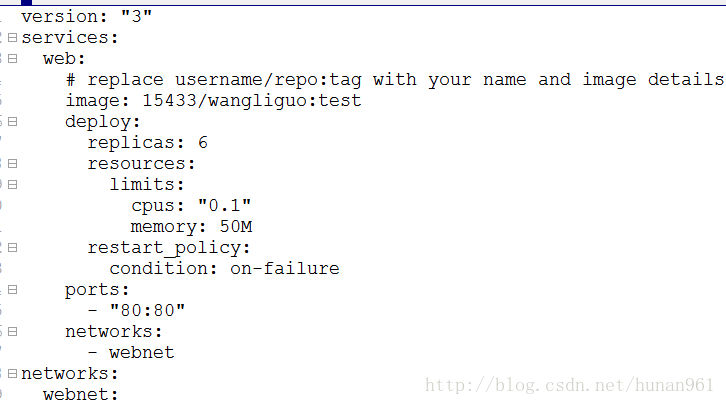
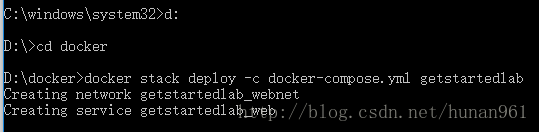
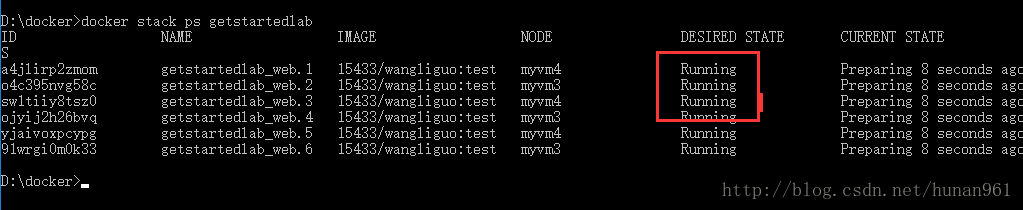
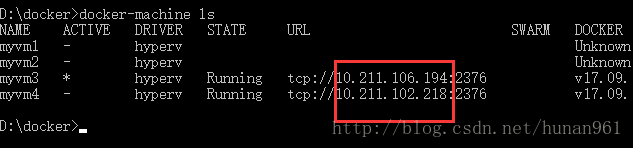
docker swarm join --token SWMTKN-1-0csyw4yz6uxob90h0b8ejoimimrgisiuy9t2ugm8c1mxfvxf99-7q7w5jw1mrjk1jlri2bcgqmu8 10.211.106.194:2377

* 1

这里写图片描述  
然后再切换到myvm3 的cmd.exe中执行命令：docker node ls  
  
可以看到，我们已经创建了一个简单的集群。、

附离开群命令：docker swarm leave

### 在集群上部署应用程序

1. docker-machine为swarm管理器配置一个shell  
   运行命令：docker-machine env myvm3  
     
   注意最后一行的内容  
   我这里是@FOR /f "tokens=\*" %i IN ('docker-machine env myvm3') DO @%i  
   复制粘贴运行它  
   这里写图片描述  
   再运行docker-machine ls以验证它myvm3 是否为活动机器  
   
2. 在swarm管理器上部署应用程序  
   以部署我们之前docker-compose.yml服务为例  
     
   首先在这个以管理员身份打开的cmd中进入到docker-compose.yml文件的所在的目录中去，我的因为在D/docker中，如果不过来，那么执行命令：  
   docker stack deploy -c docker-compose.yml getstartedlab会出现：  
   这里写图片描述  
   进入目录后执行上面那条命令：  
     
   再执行命令docker stack ps getstartedlab 查看服务详情：  
     
   如果发现state存在shutdown的情况（也有可能出现你的图片上的实例数量大于你在服务文件中定义的数量的情况，这都是我爬过的坑，现在我这里定义的是6，图片上也是6，），那应该是你在之前服务哪一章运行过命令：  
   docker stack deploy -c docker-compose.yml getstartedlab 而没有把getstartedlab移除掉  
   这时应该执行命令：docker stack rm getstartedlab 把getstartedlab移除掉  
   Ps：我是d，c盘都执行了这个命令，因为之前运行docker stack deploy -c docker-compose.yml getstartedlab命令是在d盘的时候  
   然后重新运行：docker stack deploy -c docker-compose.yml getstartedlab 和  
   docker stack ps getstartedlab 就好了
3. 浏览器访问集群的网址：  
   如图红框框所示，浏览器上访问其中的任何一个，然后刷新，将看到6个可能的容器ID，它们都是随机循环的，展示了负载平衡  
   
4. 其他的一些命令：

比如说如果修改了docker-compose.yml文件后，执行命令：

docker stack deploy -c docker-compose.yml getstartedlab

再次运行以部署这些更改即可

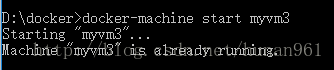
比如说前面提到的移除应用程序：docker stack rm getstartedlab

离开群：docker swarm leave –force

重新启动已停止的虚拟机，执行：

docker-machine start <machine-name>

* 1
* 2
* 3
* 4
* 5
* 6
* 7



## 8.参考文档

<https://docs.docker.com/>  
<http://blog.csdn.net/tina_ttl/article/details/51372604>  
<http://blog.csdn.net/u011686226/article/details/53906976>  
<http://blog.csdn.net/heatdeath/article/details/70432417>  
<http://blog.csdn.net/fromsilence520/article/details/73480219>

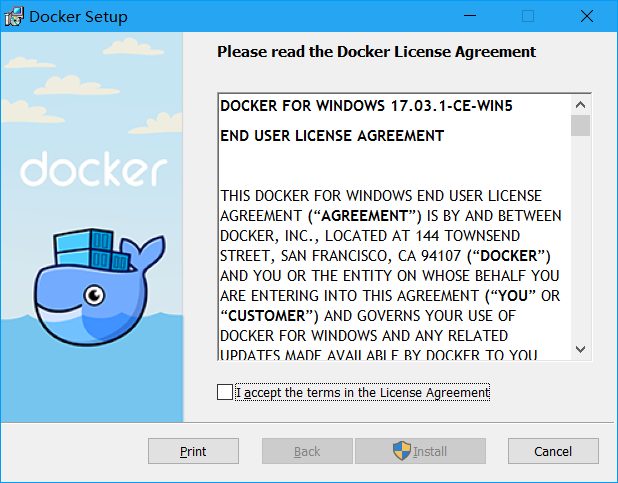
转自：<https://blog.csdn.net/hunan961/article/details/79484098>

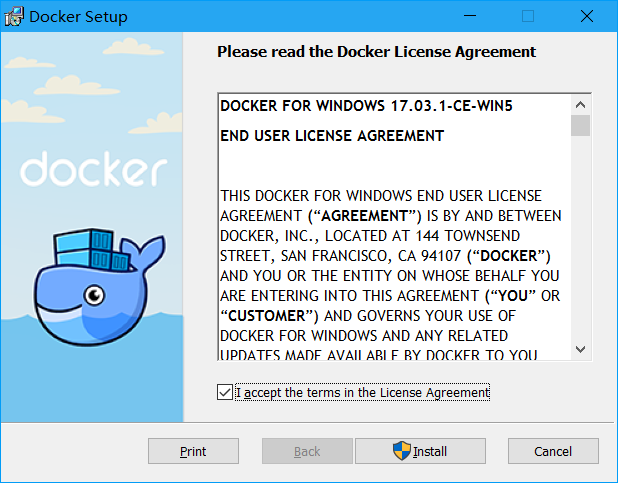
# install

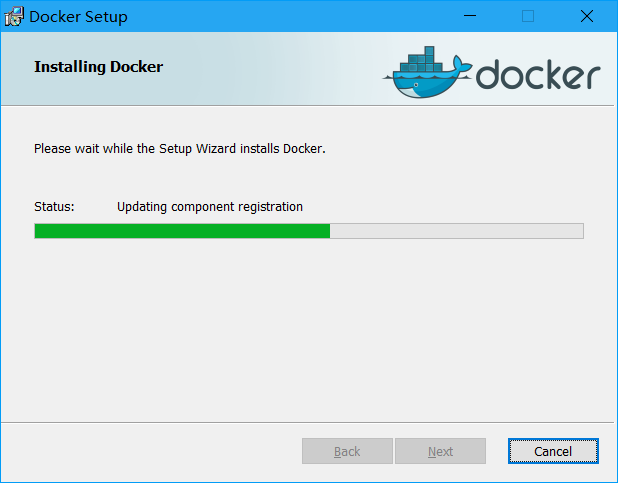
## Docker

Windows (Requires Windows 10 Professional or Enterprise 64-bit) <https://download.docker.com/win/stable/InstallDocker.msi>

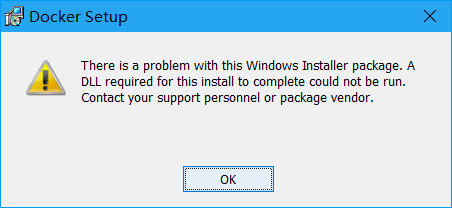


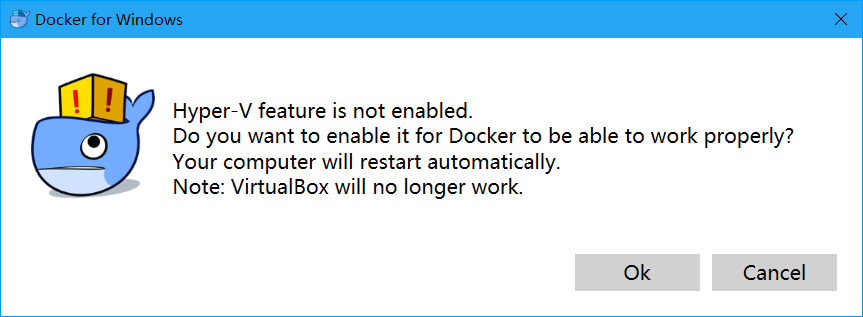




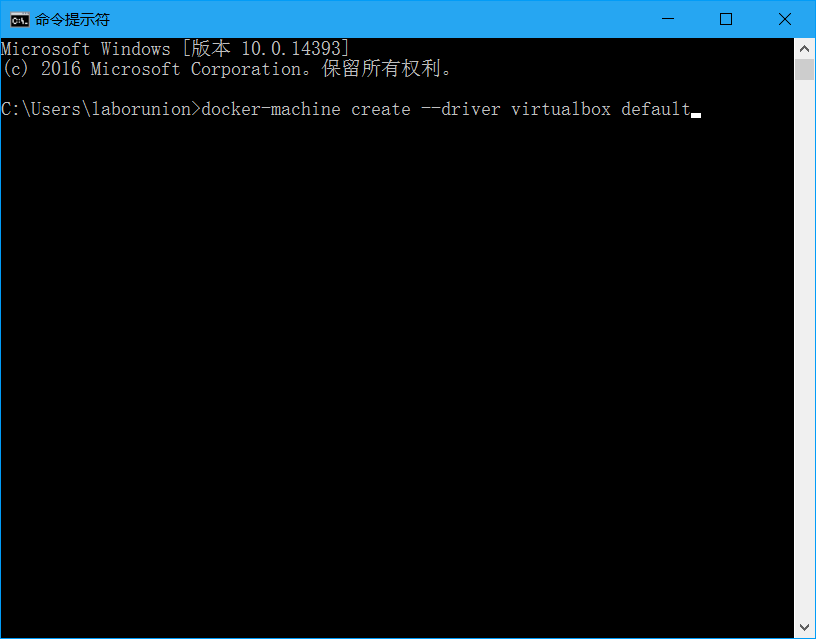


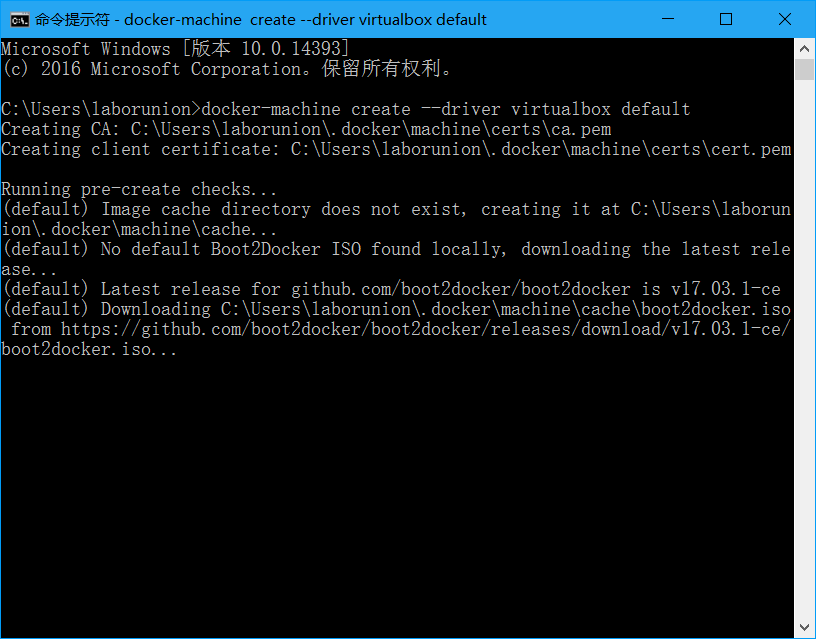


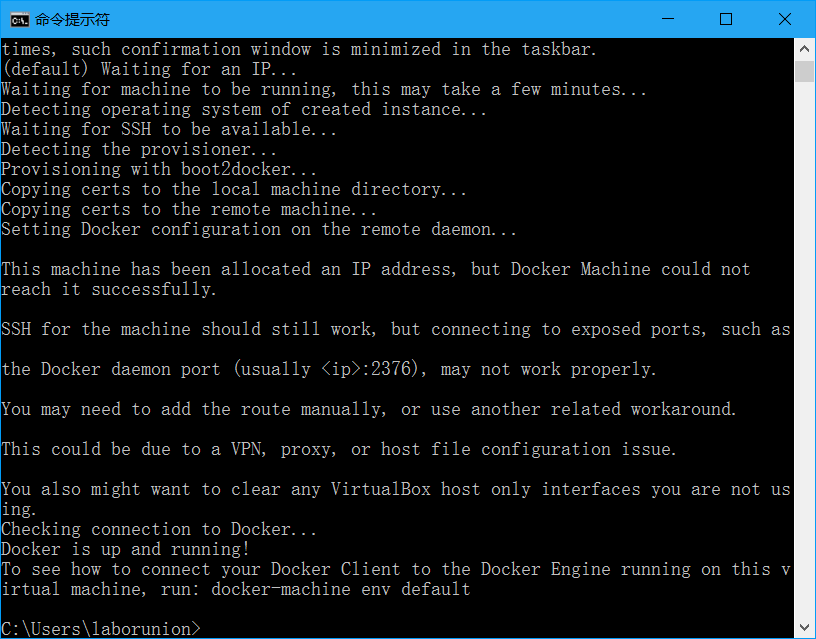




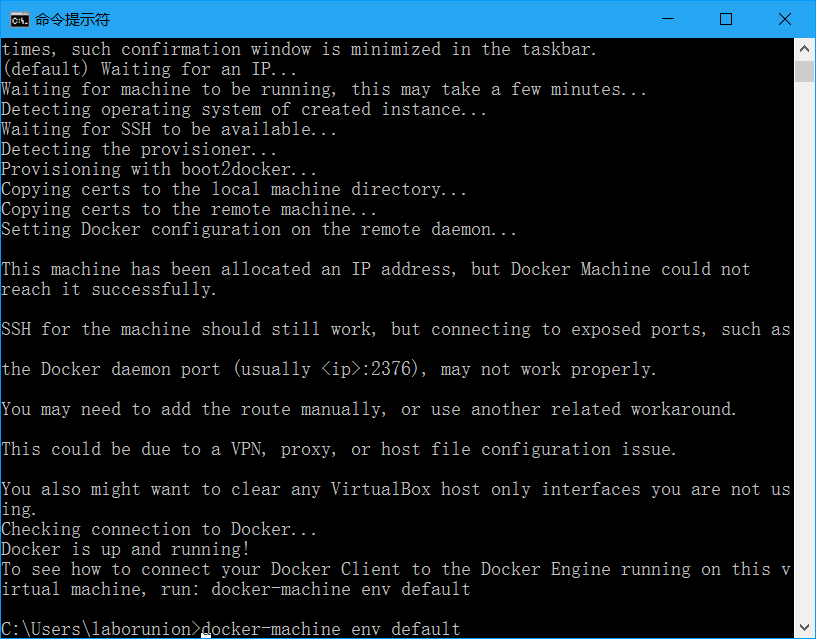
docker-machine create --driver virtualbox default

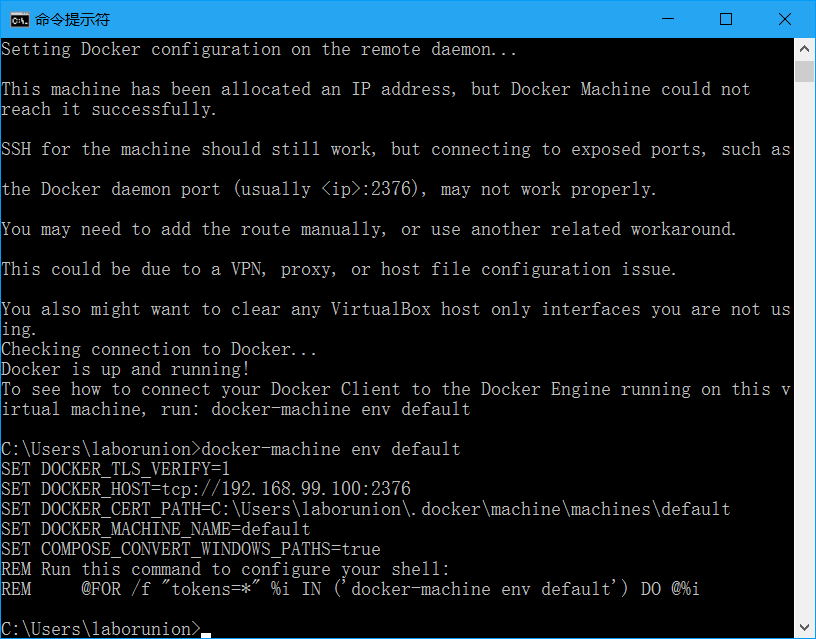




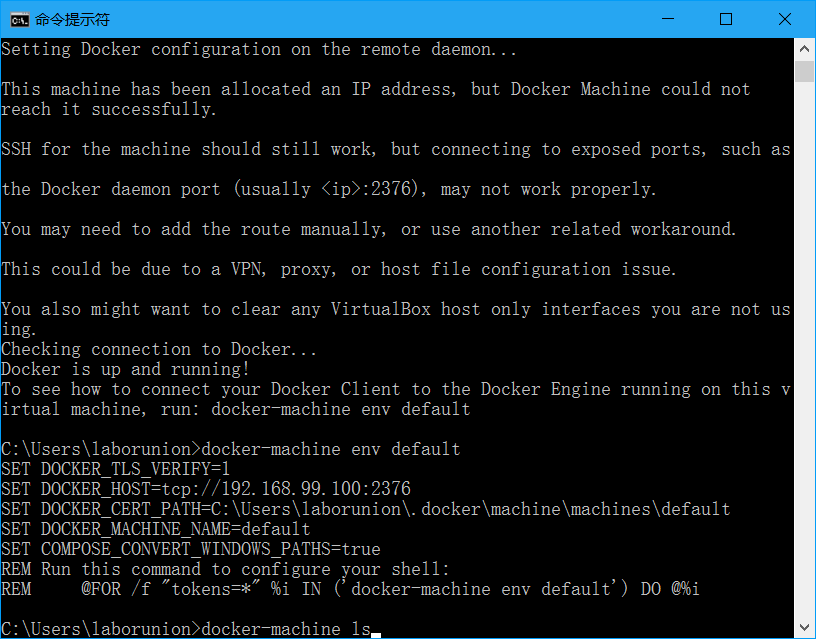


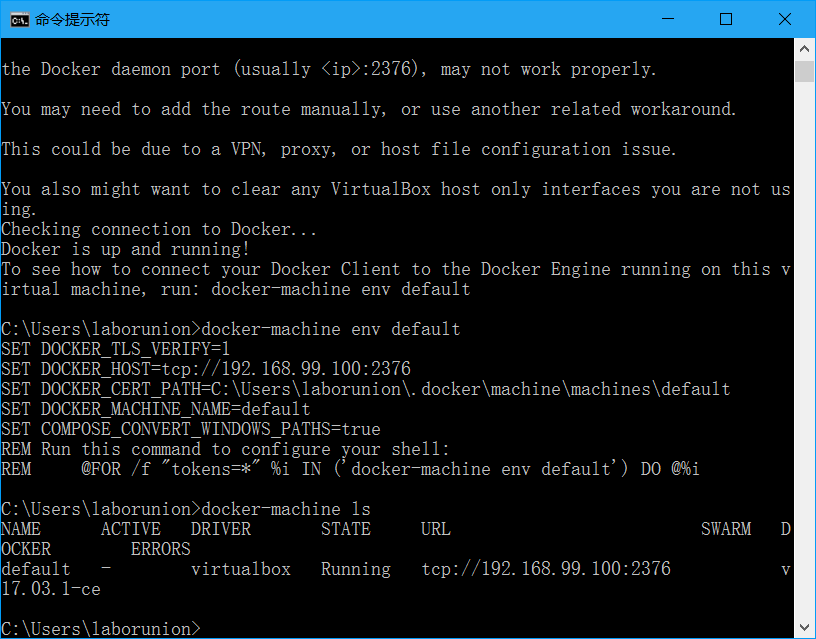
docker-machine env default



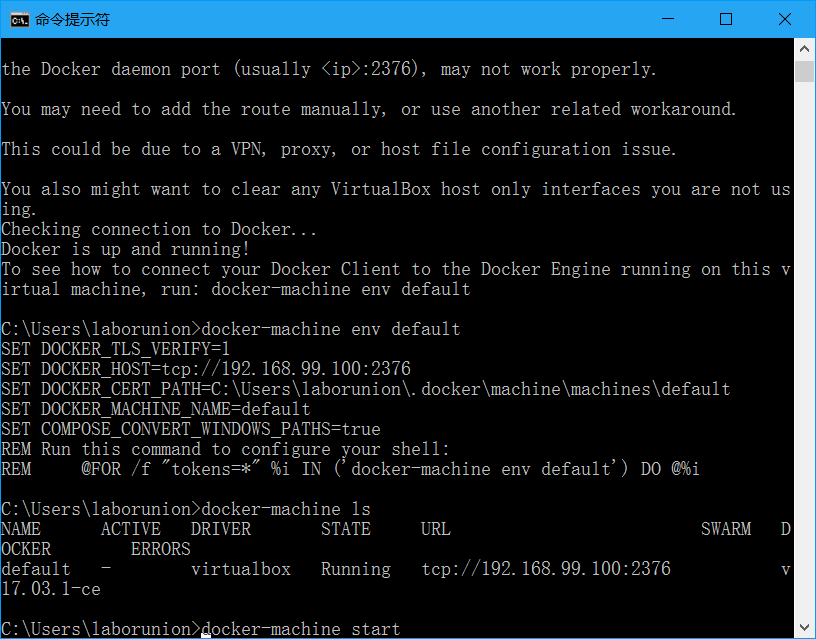


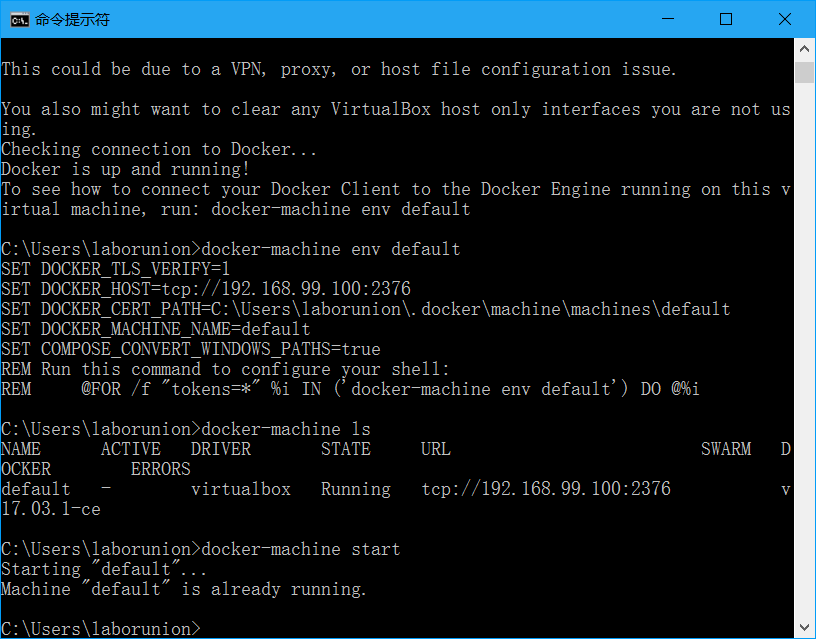
docker-machine ls



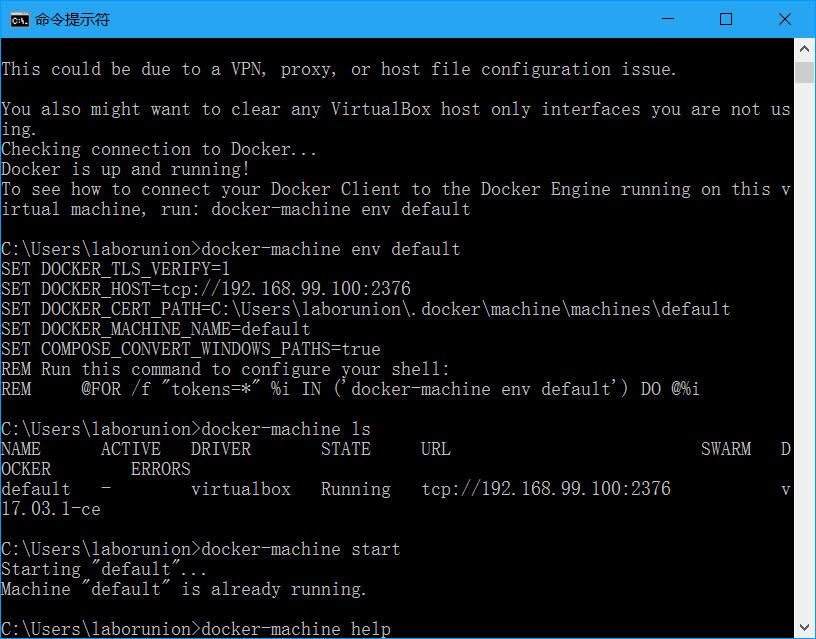


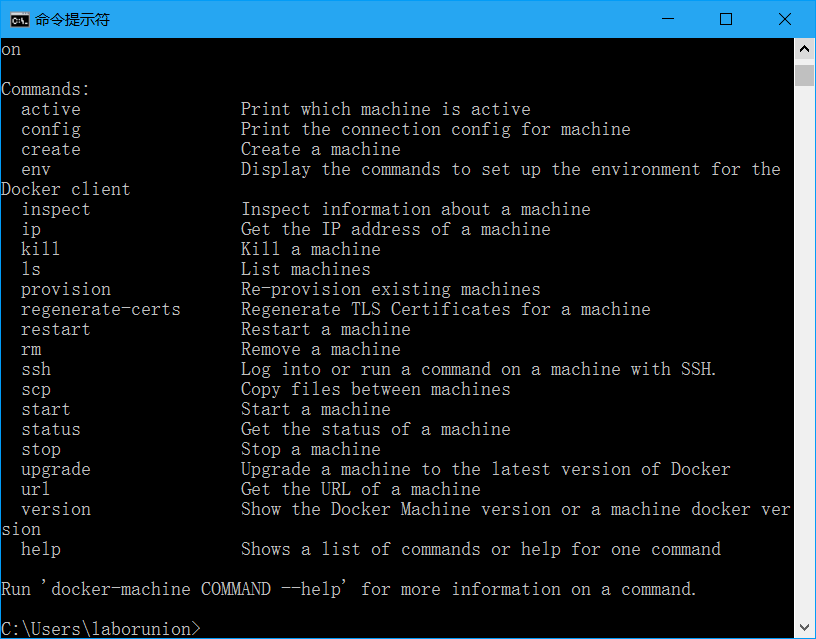
docker-machine start





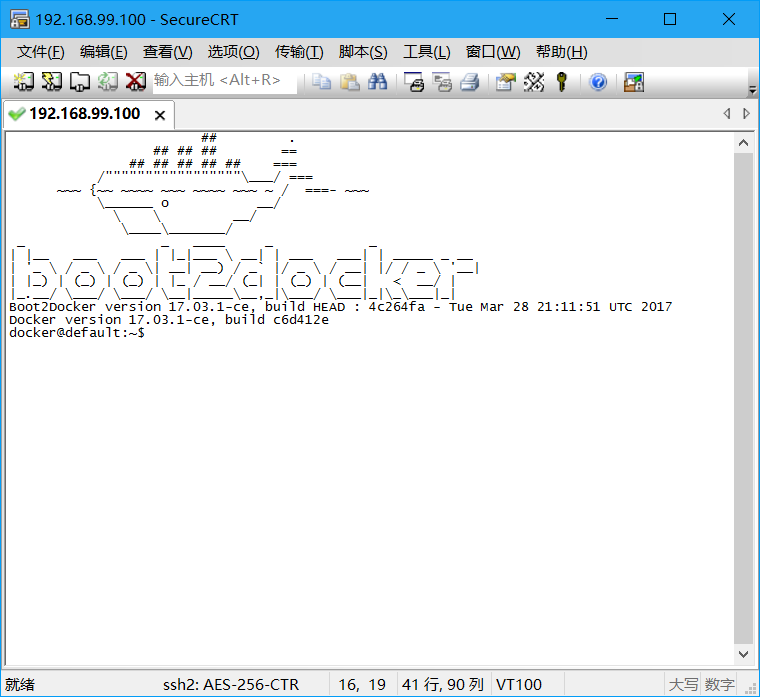
docker-machine help

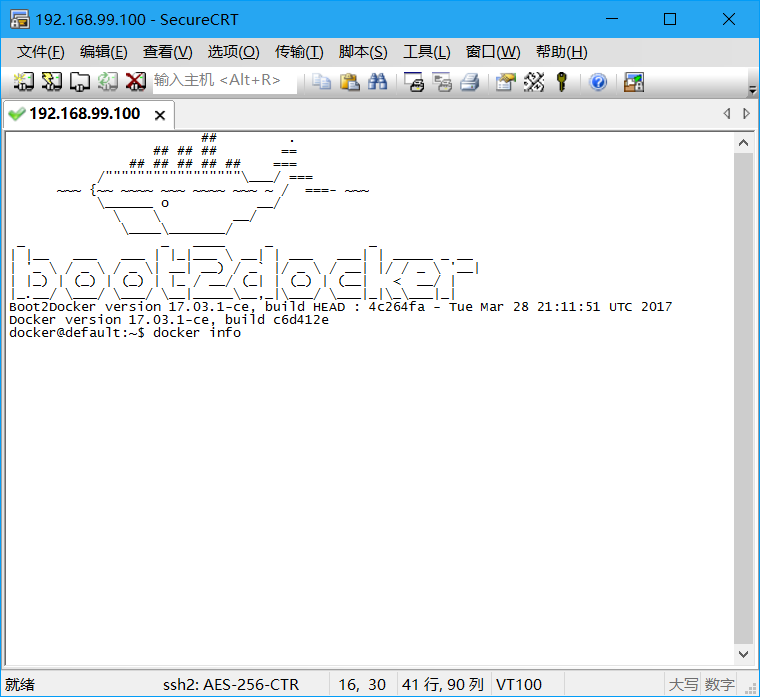


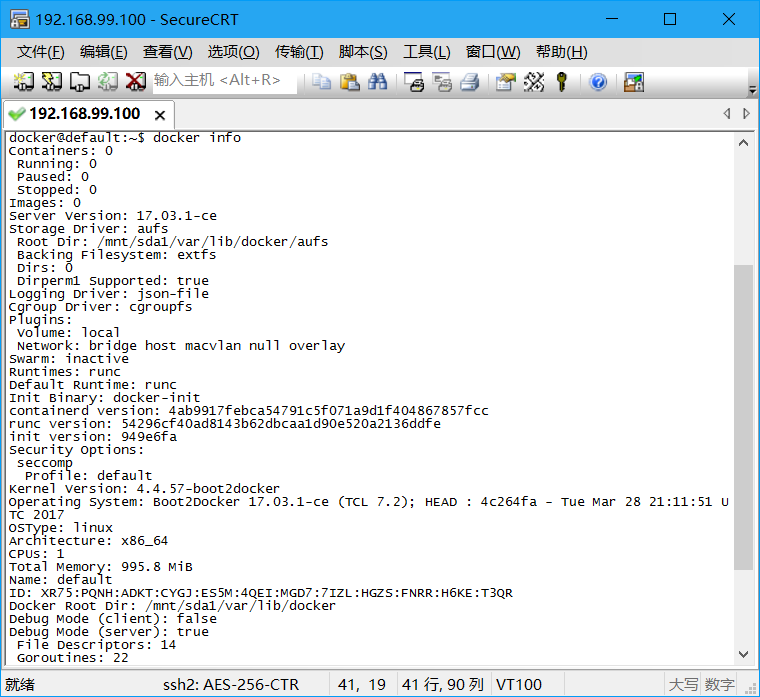


User : docker

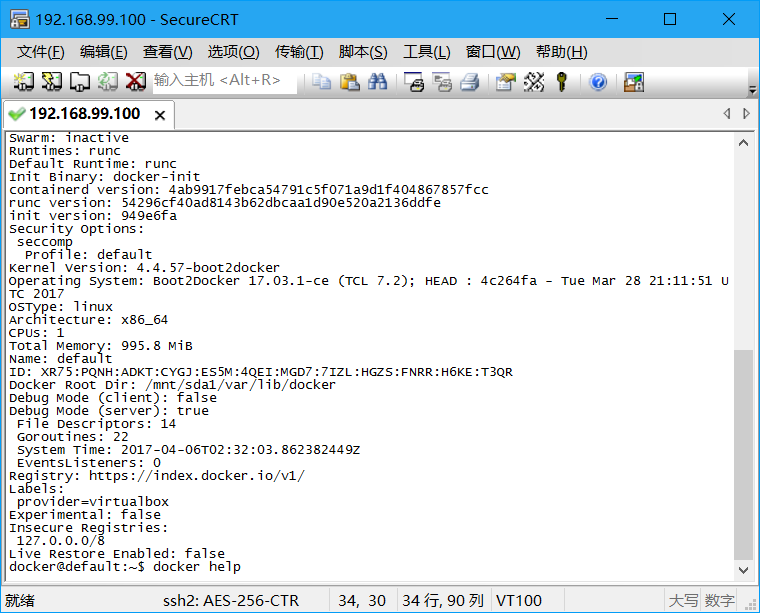
tcuser

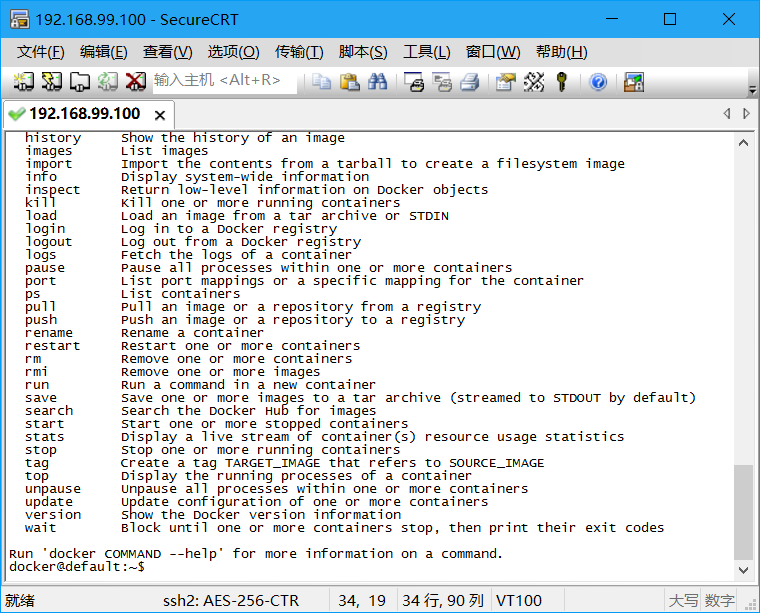






docker help

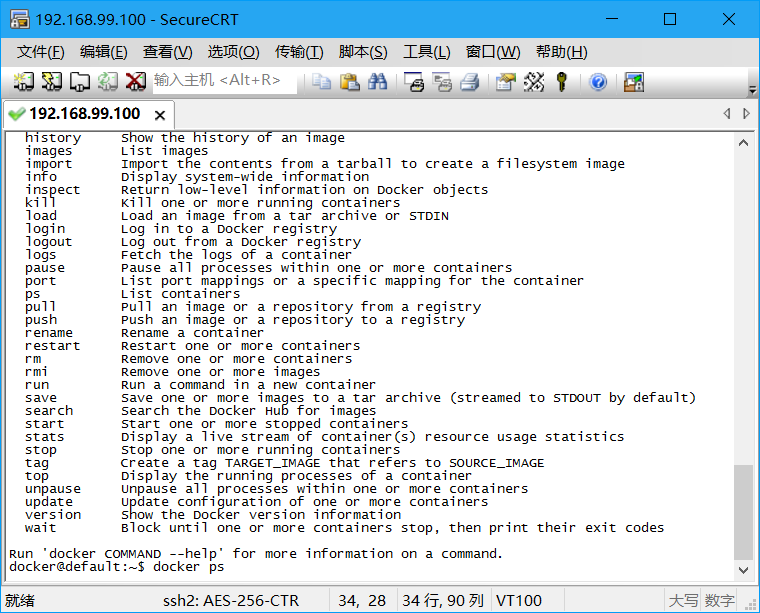


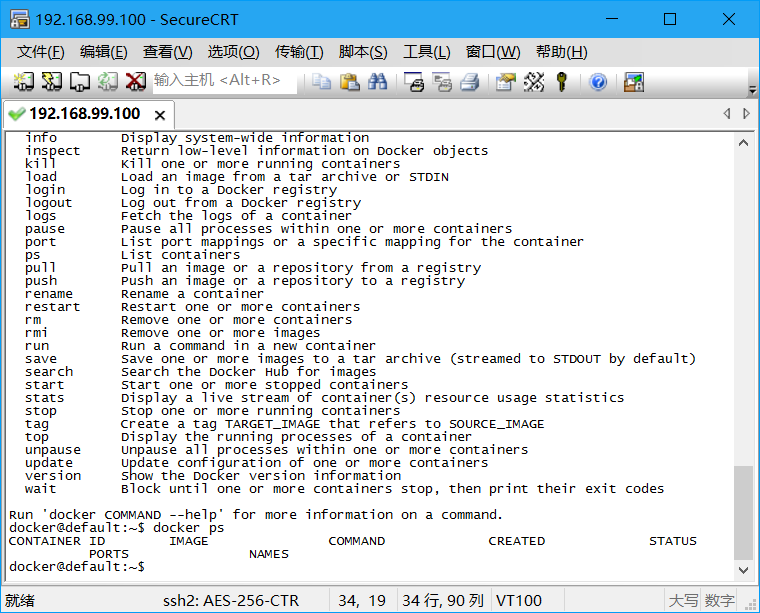


查看正在运行的容器

docker ps

docker ps -a为查看所有的容器，包括已经停止的。





查看所有镜像

docker images

运行一个新容器，同时为它命名、端口映射、文件夹映射。以redmine镜像为例

docker run --name redmine -p 9003:80 -p 9023:22 -d -v /var/redmine/files:/redmine/files -v /var/redmine/mysql:/var/lib/mysql sameersbn/redmine

容器连接到mmysql容器，并将mmysql容器重命名为db。这样，sonar容器就可以使用db的相关的环境变量了。

https://hub.docker.com

拉取镜像

docker pull <镜像名:tag>

例如

docker pull sameersbn/redmine:latest

**https://hub.docker.com/r/wadmiraal/drupal/**

停止一个容器

docker stop <容器名orID>

启动一个容器

docker start <容器名orID>

杀死一个容器

docker kill <容器名orID>

删除单个容器

docker rm <容器名orID>

删除所有容器

docker rm $(docker ps -a -q)

从Container中拷贝文件出来

sudo docker cp 7bb0e258aefe:/etc/debian\_version .

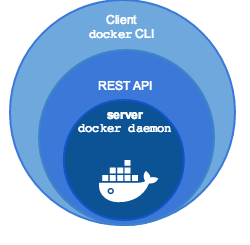
拷贝7bb0e258aefe中的/etc/debian\_version到当前目录下。

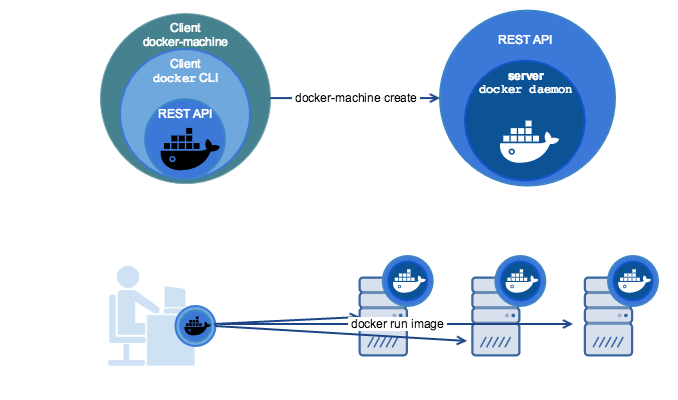
1. eval "$(docker-machine env default)"

**docker-machine**

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

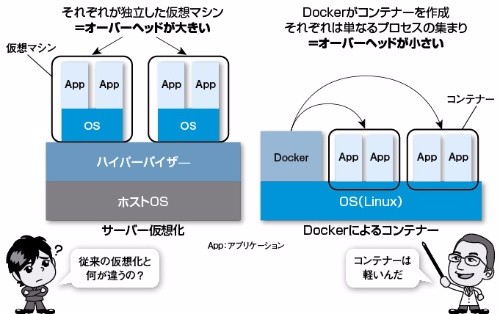


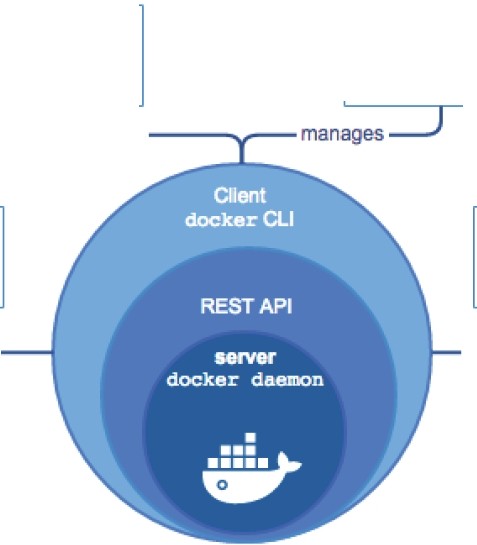




**docker**

<https://docs.docker.com/engine/understanding-docker/>





containe「

image

manages

network

da国volumes

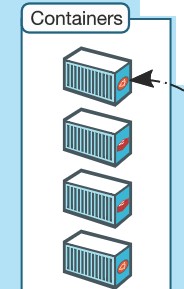
manages

manag臼



### docker pull \_,



T’－

r3－

nu－

UH－

nn－

vn－

rL－

∞一，对

l斗E

·／

」U

HM

－nu

a』

．

K

C

O

A

U

### run

／

？古L

、、 ＠ 嗡

二＠ 附



.

·

l

l

### docker

i问×

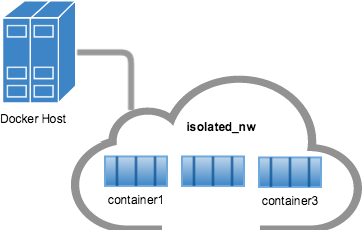
*I*

／

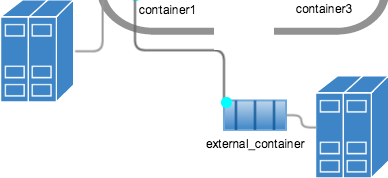
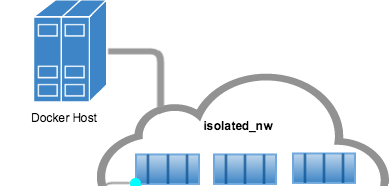
／

# docker network

<https://docs.docker.com/engine/userguide/networking/>



container2



．阳blished po同

container2

externalhost