docker部署Asp.net core应用的完整步骤

主要给大家介绍了关于docker部署Asp.net core应用的完整步骤，文中通过示例代码介绍的非常详细，对大家学习或者使用Asp.net core具有一定的参考学习价值，需要的朋友们下面来一起学习学习吧

1 容器概念

使用Docker前我们首先要简单了解一下容器的概念。MSDN上有一张虚拟机和容器的对比图，很好的展示了虚拟机和容器的区别，如下所示，虚拟机包括应用程序、必需的库或二进制文件以及完整的来宾操作系统，每台虚拟机都有一个单独的内核，我们完全可以把虚拟机看做是一台真实的物理机。容器包括应用程序及其所有依赖项，与其他容器共享 OS 内核，容器在主机操作系统上作为独立进程运行，我们可以把容器看做是一个应用沙盒。

我们经常会遇到“我机器上可以运行”的问题吧，然后部署到其他机器时就遇到了各种坑，这多是因为其他设备上缺少应用所需的环境或者缺少应用的依赖项造成的。使用容器技术可以很好的解决这个问题，容器是一个“应用的沙盒”，这个沙盒内部包含了应用所需的环境和所有依赖项，只要有这个沙盒存在，应用在任何环境下都能正常的运行，容器技术的典型特点就是“一次封装，到处运行”。

Docker 是一个使用go语言开发的开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的Linux机器上，也可以实现虚拟化。我们先了解Docker中的三个核心概念：

1 镜像(Image)

镜像是一种轻量级、可执行的独立软件包，它包含运行某个软件所需的所有内容，包括代码、运行时、库、环境变量和配置文件。

2 容器(Container)

容器是镜像的运行时实例 ，执行镜像时内存中运行的就是容器。如果把镜像看做面向对象中的类，容器就是对象。使用类时通过new来获取一个对象，类似的使用镜像时通过 docker run <镜像名/镜像id>创建一个容器。

3 仓库(Repository)

仓库时用来存放镜像的地方，Docker hub是docker官方提供的仓库，类似于githhub,，通过push推送镜像到仓库，通过pull命令拉取镜像。当然我们也可以使用其他仓库，或者自己搭建一个仓库。

2安装docker

1 Windows安装docker

docker是运行在Linux系统上的，在Windows上使用docker默认要用到Hyper-V功能，所以我们在Windows系统上安装Docker必须先开启Hyper-V功能。首先打开【控制面板】，找到【启动或关闭Windows功能】，然后勾选【Hyper-V】，点击确定即可安装Hyper-V。

安装好Hyper-V后，到docker官网下载Windows版本的docker，docker需要登录账户才能下载，我们可以创建一个账户，牢记这个账号，我们以后会经常用到。docke下载地址：docker下载。下载完成后，双击安装包，一路next即可。安装完成后需要重启电脑，打开命令行，输入docker run hello-world，如果出现以下界面，表示docker已经安装成功了。docker run hello-world命令首先会在本地找名字为hello-world的镜像，如果本地不存在这个镜像就会从docker hub上拉取hello-world镜像，然后运行。

2 Centos安装docker

Linux系统安装常用软件都十分方便，执行下边的命令即可安装docker

```
#删除旧版本
sudo yum remove docker \
   docker-client \
   docker-client-latest \
   docker-common \
   docker-latest \
   docker-latest-logrotate \
   docker-logrotate \
   docker-engine
#安装一些需要的工具
yum install -y yum-utils device-mapper-persistent-data lvm2
#添加软件源信息
sudo yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
#更新yum缓存
```

```
yum makecache fast
#安装docker
yum -y install docker-ce
#启动docker服务
systemctl start docker
#测试docker
docker run hello-world
```

安装完成后执行docker run hello-world命令，如果出现下边的界面表示docker安装成功了

3 Docker基本使用

1 使用官方docker实例

可以查看官方教程详细地了解docker的使用，这里我们先用两分钟分钟展示一下微软官方提供的docker栗子，执行命令docker run -p 8000:80 --name aspnetcore_sample mcr.microsoft.com/dotnet/core/samples:aspnetapp，-p 8000:80表示将服务器的8000端口和容器的80端口建立映射，访问服务器8000端口时会去访问容器的80端口。--name设置镜像的名字

打开浏览器，访问虚拟机的8000端口如下，说明我们的容器已经成功启动了！

我们使用docker images和docker container ls查看一下本机的容器和镜像，结果如下：

2 Docker常用命令

其实我们实际开发中能用到的docker命令并不多，这里简单汇总一下最常用的docker命令。

```
## 查看docker版本和信息
#查看docker-cli的版本
docker --version
#查看docker版本和信息
docker version
docker info

## docker镜像相关命令
#拉取镜像
docker pull [imgName:tag]
#运行镜像,执行后会生成一个容器
docker run [imgName:tag/imgID]
#查看所有的镜像
docker images
#删除镜像 执行rmi命令前必须删除所有该镜像的container
docker rmi [imgName:tag/imgID]
#查看所有的容器，没有-a表示查询所有正在运行中的容器
docker container ls -a
#查看docker正在运行和已经停止的容器
docker ps -a
#启动容器
docker start [ctnName/cID]
#停止容器
docker stop [ctnName/cID]
#删除容器，执行rm命令前必须先停止该容器
docke rm [ctnName/cID]

##清理镜像和容器
#删除所有的镜像和容器
docker kill $(docker ps -q) ; docker rm $(docker ps -a -q) ; docker rmi $(docker images -q -a)
#删除所有的容器
docker kill $(docker ps -q) ; docker rm $(docker ps -a -q)
#清除名称为none的镜像
docker ps -a | grep "Exited" | awk '{print $1 }'|xargs docker stop
docker ps -a | grep "Exited" | awk '{print $1 }'|xargs docker rm
docker images|grep none|awk '{print $3 }'|xargs docker rmi
```

有时候我们需要和容器交互，可以使用命令docker exec -it [name/id] common。如我们启动一个mysql容器mysqlx1，执行docker exec -it mysqlx1 mysql -p表示和容器mysqlx1交互，执行的命令是【mysql -p】进行登录，如下所示。Ctrl +p+q退出容器，回到服务器目录。

3 定制容器

大多数用户使用docker的核心目的是更方便的交付项目，就是将我们的应用程序构建成镜像，然后交付镜像即可，这里演示怎

么构建一个Aspnet core项目的镜像。

1. 开发aspnet core应用

我们创建一个Asp net core MVC项目，项目名为DockerDemo，为了方便不勾选Https和Docker支持，如下：

简单修改首页，运行一下，测试项目没有bug，运行首页如下：

2 发布项目，添加Dockerfile

发布项目，然后在发布文件夹下添加Dockerfile文件，结构如下：

Dockerfile的内容如下：

```
#拉取runtime父镜像，运行aspnet core应用必须要用runtime
FROM microsoft/dotnet:2.2-aspnetcore-runtime
#设置容器工作目录
WORKDIR /DockerDemo
#把当前目录的所有文件copy到工作目录中
COPY . /DockerDemo
#暴露一个端口让外部可以访问
EXPOSE 80
#容器入口命令，即容器启动时执行dotnet DockerDemo.dll命令
ENTRYPOINT ["dotnet", "DockerDemo.dll"]
```

Dockerfile常见指令简单说明

Dockerfile常用指令：

```
#FROM:通常情况下，我们创建的镜像时基于另外一个镜像的，这时就要用FROM，当然我们也可以完全从头创建
#WORKDIR:设置容器的工作目录
#MAINTAINER:该镜像的维护人
#COPY:经常需要把源码复制到容器中，只用COPY就可以实现这一点
#RUN:这里可以定义一些需要运行的命令，执行的目录是WORKDIR。如npm install ,dotnet restore，dotnet run等
#ENTRYPOINT:定义容器的入口，通常是一些dotnet/node等命令,如dotnet xxx.dll
#CMD:设置容器运行的默认命令和参数,即容器启动时执行的命令。当容器运行时，可以通过命令行覆盖CMD定义的命令
#EXPOSE:暴露端口
#ENV:设置环境变量
```

3 创建镜像

docker创建镜像很简单，执行docker build -t dockerdemo:v1.0 .命令即可创建镜像，其中dockerdemo:v1.0是镜像的名字和版本tag，名字和tag可以随便设置。然后通过docker images命令查看所有镜像，如下：

4 运行容器

使用命令docker run -d -p 8080:80dockerdemo:v1.0启动容器，-d 表示后台运行，-p 8080：80表示服务器的8080端口映射到容器的80端口，我们也可以使用服务器的其他端口，容器的端口为Dockerfile中Expose的端口。执行docker container ls查看容器，这里我们的容器已经启动了，如下：

打开浏览器，输入【服务器IP:服务器端口】接口访问容器，如下：

到这一步，本地创建镜像和运行容器成功了。

5 使用docker hub存储和分发镜像

docker提供了远程仓库用于存储和分发镜像，其作用类似于github,区别在于github托管代码，而docker hub托管镜像。我们创建好了镜像可以推送(push)到远程仓库，然后在其他设备上拉取(pull)镜像，有了docker hub我们可以更方便的管理和分发镜像。docker hub的使用非常简单，只记住三个命令就可以了

```
#标记镜像，就是设置远程镜像名和标记
docker tag localname:localtag dockeruser/repname:reptag
#推送镜像文件到docker hub
docker push username/repname:reptag
#拉取远程镜像
docker pull repname:reptag

#注：localname是本地镜像名 localtag是本地镜像tag
# repname远程仓库的镜像名 reptag是远程仓库的tag
# username是我们的dockerhub用户名
```

这里将上边创建的本地镜像推动到docker hub。使用命令docker tag dockerdemo:v1.0 wyyxxx/resdockerdemo:v1.0标记，使用命令docker push wyyxxx/resdockerdemo:v1.0推动镜像到远程仓库，打开镜像仓库，登陆后就可以看到我们自定义的镜像了，如下所示。

然后使用另一台安装了docker的服务器，执行命令docker run -p 8080:80 wyyxxx/resdockerdemo:v1.0拉取远程镜像并启动，打开浏览器输入地址，如下：

我们在通过拉取远程镜像并运行容器时，不需要事先部署环境(如安装runtime)等，因为镜像中已经设置了需要的所有环境和依赖资源，只需要使用docker run命令运行即可。这也是使用容器交付的方便之处。

4 使用Docker的一点建议

① “3C原则”一个容器只运行一个应用

② 使用镜像交付应用程序，而不是直接部署

如果要在centos上部署一个aspnet core应用程序。错误的做法是：在docker中先安装一个centos系统，在centos系统中部署环境，然后部署我们的netcore应用程序,,最后把部署的应用程序生成镜像。正确的做法是通过Dockerfile定义我们需要的环境和依赖项，然后生成镜像，通过镜像去分发和执行。

③ 分层构建镜像

还是上边centos部署netcore应用的栗子，如果从零定制镜像时，我们应该把centos作为一层镜像，环境runtime作为一层镜像，最后我们的应用做为一层镜像，使用Dockerfile的FROM 指令拉取父镜像。这样做的好处是节省空间和复用镜像。如上边我们使用了2.2版本的runtime镜像，当我们创建了另一个web应用再来定制镜像时，这两个应用程序镜像可以 共用同一个runtime镜像。

④ 不要把数据存储在容器中

容器是随时可能销毁的，销毁后容器内部的数据就不存在了，所以最好不要在容器中存储需要持久化的数据。

总结

以上就是这篇文章的全部内容了，希望本文的内容对大家的学习或者工作具有一定的参考学习价值，谢谢大家对我们的支持。