1. **实验名称：**

Linux系统下Docker的基本操作

1. **实验目的**
2. 掌握Docker的安装应用程序、更新应用程序及共享应用程序
3. 掌握Docker如何持久化数据库及使用绑定挂载
4. 掌握Docker使用多容器应用程序
5. 掌握Docker Compose的使用
6. **实验环境**

虚拟机所在环境：Windows 10

虚拟机版本：VMware workstation Pro 16，

虚拟机内部Linux发行版本：Ubuntu22.04

1. **实验内容及步骤**
2. **安装应用程序**
3. 从 <https://codeload.github.com/docker/getting-started/zip/refs/heads/master>上下载示例使用的项目，并将压缩包解压到Ubuntu文件夹中
4. 在解压的文件夹app内（保证dockerfile文件与package.json文件同层）打开控制台，用touch Dockerfile代码创建Dockerfile文件（不用加任何后缀不然会变成文本文件，无法使用），并打开Dockerfile文件输入以下代码，保存

# syntax=docker/dockerfile:1

FROM node:12-alpine

RUN apk add --no-cache python2 g++ make

WORKDIR /app

COPY . .

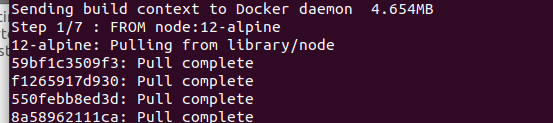
RUN yarn install --production

CMD ["node", "src/index.js"]

EXPOSE 3000

1. 用sudo su -登录获取超级用户权限，用cd ???/getting-started-master/app 转到Dockerfile文件所在目录（问号内为下载路径，之后为解压后的文件夹的名称）在该路径内使用docker build -t getting-started . （注意最后面有个点）来构建应用程序

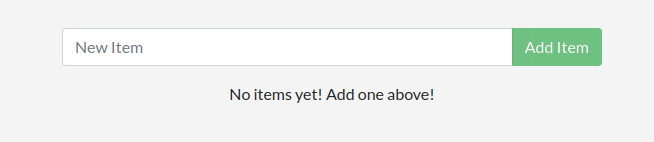
开始构建后出现该图片（因为我们指示构建器我们想从node:12-alpine镜像开始。但是，由于我们的机器上没有该镜像，因此需要下载该镜像）



显示该图片意为安装完成



1. 构建完成后就可以使用命令docker run -dp 3000:3000 getting-started启动应用程序
2. 在浏览器访问<http://localhost:3000> 便可以看到效果

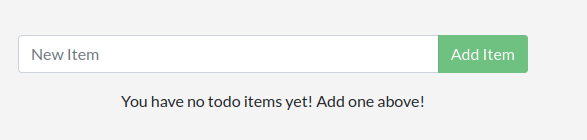


1. **更新应用程序**
2. 更新src/static/js/app.js文件第56行中的

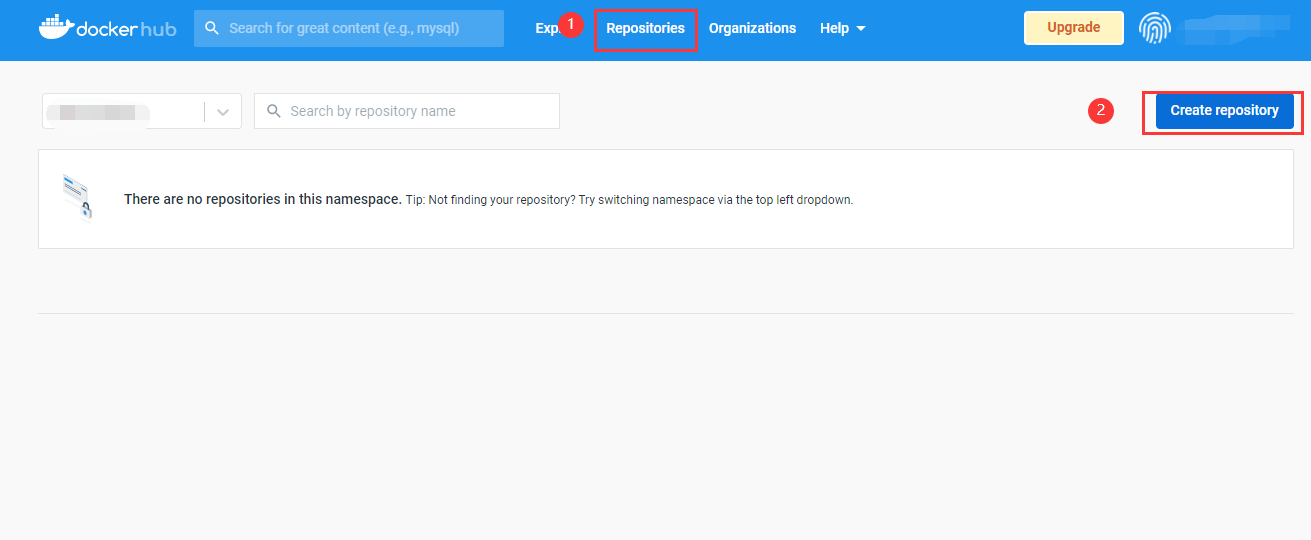
<p className="text-center">No items yet! Add one above!</p>替换为

<p className="text-center">You have no todo items yet! Add one above!</p>

1. 使用docker ps命令获取容器ID，使用docker stop <容器的id>来停止该容器，最后使用docker rm<容器的id>来删除容器
2. 再次使用docker build -t getting-started . （注意后面有个点）更新镜像，并用老命令docker run -dp 3000:3000 getting-started启动，
3. 在浏览器访问http://localhost:3000 便可以看到效果



1. **共享应用程序**
2. 注册登录Docker Hub网站，并在网站上创建一个仓库，名称自取，权限定为Public



创建完成后，会出现Docker命令



1. 在Ubuntu的终端使用命令docker login -u （你的用户名） 登录到Docker Hub中登陆成功后会显示下图字段



1. 使用docker tag getting-started （你的用户名）/getting-started 命令为getting-started 镜像指定一个新的名称
2. 再次使用docker push （你的用户名）/getting-started命令上传至远程仓库，且若没有指定版本号，它会默认指定为latest
3. **持久化数据库**
4. 在Ubuntu的终端中使用sudo su -获取超级用户权限后，使用以下命令创建一个容器，且在该容器内创建一个包含一个1到10000之间的随机数的data.txt文件（第一部分选择随机数写入data.txt，第二部分只是监视文件以保持容器运行）

docker run -d ubuntu bash -c "shuf -i 1-10000 -n 1 -o /data.txt && tail -f /dev/null"

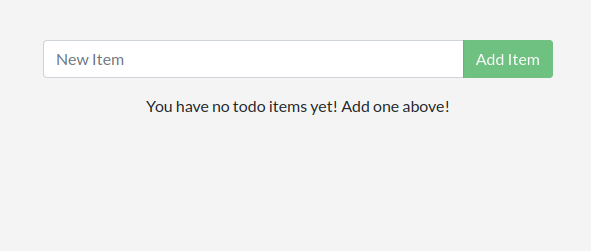
1. 使用docker exec <容器id> cat /data.txt命令查看data.txt内的数据



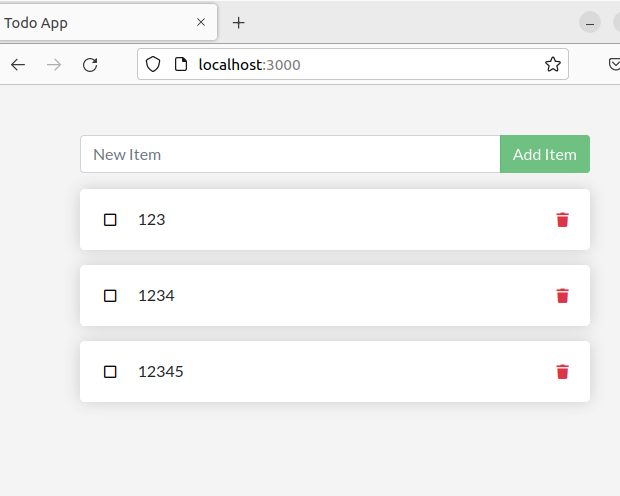
1. 使用以下命令继续创建另一个容器，由于data.txt文件仅被写入第一个容器的暂存空间，所以会发现第二个容器内并没有data.txt文件：

docker run -it ubuntu ls /

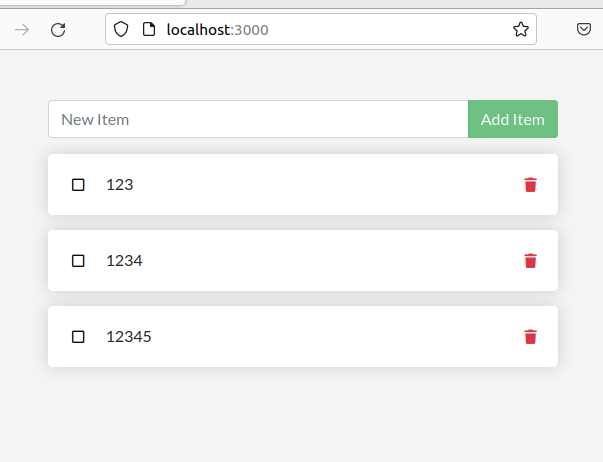
1. 继续并使用docker rm -f <容器id>删除第一个容器
2. 使用docker volume create <数据卷名称>命令创建数据卷，并使用查看是否存在同名的容器，若有则使用docker rm -f <容器id>将其删除
3. 使用命令docker run -dp 3000:3000 -v todo-db:/etc/todos getting-started创建启动数据卷并挂载到/etc/todos路径上，且可以通过浏览器启动



可以在其中添加一些代办事项



1. 停止并删除todo容器，并将其删除，使用上述命令创建一个新容器，打开页面后可以看到待办事项仍在列表中



1. **使用绑定挂载**
2. 确保没有docker内没有活跃的容器后，用cd /home/???/getting-started-master/app (问号为getting-started-master所在位置)转到app文件夹中，使用命令

docker run -dp 3000:3000 \

-w /app -v "$(pwd):/app" \

node:12-alpine \

sh -c "yarn install && yarn run dev"

（-dp 3000:3000 ：以后台模式运行并创建端口映射

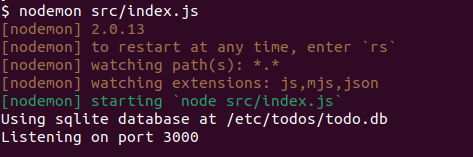
-w /app ：设置“工作目录”或命令将运行的当前目录

-v "$(pwd):/app" ：将容器中主机的当前目录绑定挂载到/app目录中

node:12-alpine ：要使用的镜像。这是来自 Dockerfile 的应用程序的基础镜像

sh -c "yarn install && yarn run dev"：使用sh（alpine 没有bash）启动一个 shell 并运行yarn install以安装所有依赖项，然后运行yarn run dev。）

1. 使用docker logs -f <容器id>查看日志，如果出现以下图片，便表示已经就绪

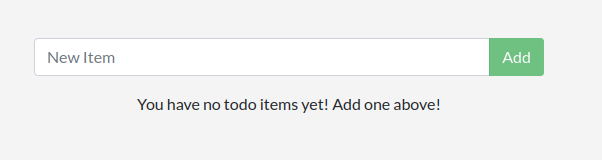


1. 对src/static/js/app.js文件第109行进行修改，将

{submitting ? 'Adding...' : 'Add Item'}修改为

{submitting ? 'Adding...' : 'Add'} ，

刷新或重新打开浏览器页面后，页面会变为以下样式



1. 任意修改后，使用docker stop <容器id>将容器停止，并使用docker build -t getting-started . （注意：最后有个点）构建新镜像，出现下图即为成功



1. **多容器应用**
2. 启动MySQL，首先用命令docker network create todo-app创建网络，再使用以下命令启动MySQL容器并将其连接到网络

docker run -d \

--network todo-app --network-alias mysql \

-v todo-mysql-data:/var/lib/mysql \

-e MYSQL\_ROOT\_PASSWORD=secret \

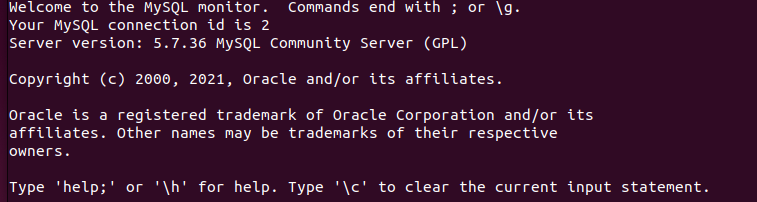
-e MYSQL\_DATABASE=todos \

mysql:5.7

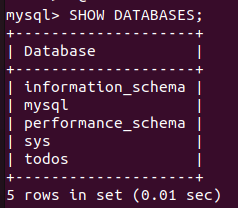
显示以下图片即为成功安装，并使用docker exec -it <MySQL容器id> mysql -u root -p 命令进行检测是否已连接



出现密码提示，输入 secret 后进入mysql shell，命令行开头变为“mysql>”



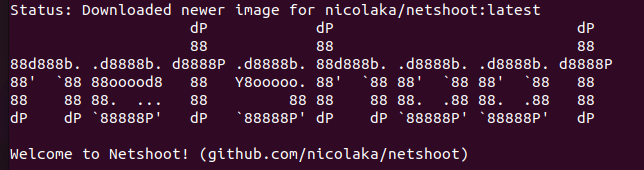
继续输入SHOW DATABASES;（注意：这最后有个分号）,便会出现以下输出



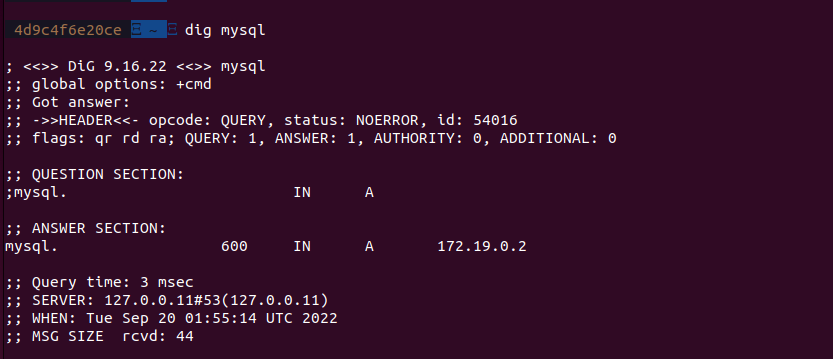
1. 用命令exit 返回机器上原本的shell，并使用nicolaka/netshoot 镜像启动一个新容器。确保将其连接到同一网络，命令如下：

docker run -it --network todo-app nicolaka/netshoot

由于之前并没有安装过netshoot镜像，所以系统会先安装，安装完成后会显示以下图片



同时，它会进入容器内部，在内部使用dig mysql命令，就可以查找主机名mysql的IP地址，显示为下图



1. 指定运行MySQL的主机名、用户名、密码及连接后使用的数据库，并将容器连接到应用程序网络

docker run -dp 3000:3000 \

-w /app -v "$(pwd):/app" \

--network todo-app \

-e MYSQL\_HOST=mysql \

-e MYSQL\_USER=root \

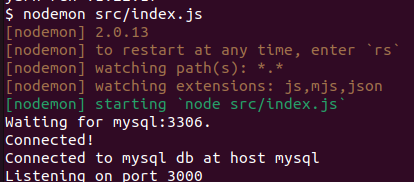
-e MYSQL\_PASSWORD=secret \

-e MYSQL\_DB=todos \

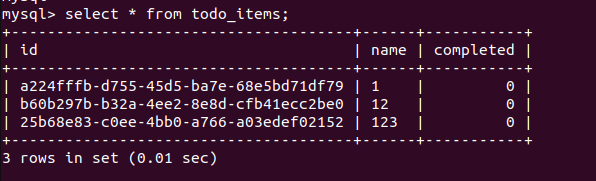
node:12-alpine \

sh -c "yarn install && yarn run dev"

此时，若使用docker logs <容器id> 命令，便可看到其中有以下信息，表明它正在使用mysql数据库



打开浏览器，将一些待办事项添加到列表，并使用命令docker exec -it < mysql容器id> mysql -p todos （todos是数据库名）连接数据库（密码是SECRET），在mysql shell中运行 select \* from todo\_items; 命令，便会显示结果



这里，多容器应用就告一段落。

1. **使用Docker Compose**
2. 首先安装docker compose，使用命令sudo apt-get update 和sudo apt-get install docker-compose-plugin更新报索引，并使用apt install docker-compose安装最新版本Docker Compose，安装完成后使用命令docker-compose -version 便可查看是否正确安装。如下图所示，格式为Docker Compose version vN.N.N



1. 在app文件夹中创建一个名为docker-compose.yml 的文件，撰写文件内容时首先需要指定版本，大部分情况下最好使用最新版本，如果不想使用最新版本，可以参考 https://docs.docker.com/compose/compose-file/ 该网站来使用。
2. 将应用服务迁移到compose文件中，以下为shell内原命令：

docker run -dp 3000:3000 \

-w /app -v "$(pwd):/app" \

--network todo-app \

-e MYSQL\_HOST=mysql \

-e MYSQL\_USER=root \

-e MYSQL\_PASSWORD=secret \

-e MYSQL\_DB=todos \

node:12-alpine \

sh -c "yarn install && yarn run dev"

以下为需要写入到compose文件内的代码

version: "3.7"

services:

app:

image: node:12-alpine

command: sh -c "yarn install && yarn run dev"

ports:

- 3000:3000

working\_dir: /app

volumes:

- ./:/app

environment:

MYSQL\_HOST: mysql

MYSQL\_USER: root

MYSQL\_PASSWORD: secret

MYSQL\_DB: todos

App:image：为容器定义服务入口和镜像

Command：接近镜像定义的内容，可直接移入

Ports：迁移自“-p 3000:3000”，这里使用的是短语法，也可使用长语法

Working\_dir和Volumes：迁移工作目录（-w/app）和数据卷映射（-v ”$(pwd):/app”）

Environment：定义环境变量

1. 将MySQL服务迁移到compose文件中，原shell命令如下：

docker run -d \

--network todo-app --network-alias mysql \

-v todo-mysql-data:/var/lib/mysql \

-e MYSQL\_ROOT\_PASSWORD=secret \

-e MYSQL\_DATABASE=todos \

mysql:5.7

以下为写入到compose文件中的代码

version: "3.7"

services:

app:

# The app service definition

mysql:

image: mysql:5.7

volumes:

- todo-mysql-data:/var/lib/mysql

environment:

MYSQL\_ROOT\_PASSWORD: secret

MYSQL\_DATABASE: todos

volumes:

todo-mysql-data：

而最终整体的compose文件为

version: "3.7"

services:

app:

image: node:12-alpine

command: sh -c "yarn install && yarn run dev"

ports:

- 3000:3000

working\_dir: /app

volumes:

- ./:/app

environment:

MYSQL\_HOST: mysql

MYSQL\_USER: root

MYSQL\_PASSWORD: secret

MYSQL\_DB: todos

mysql:

image: mysql:5.7

volumes:

- todo-mysql-data:/var/lib/mysql

environment:

MYSQL\_ROOT\_PASSWORD: secret

MYSQL\_DATABASE: todos

volumes:

todo-mysql-data:

1. 完成docker-compose.yml文件后，就可以启动（启动前先确保没有其他正在运行的容器）。使用docker-compose up -d使用命令启动应用程序堆栈（-d标志表示后台运行所有内容），结果为以下输出



再使用命令docker-compose logs -f 查看日志，会看到很多日志信息，其中有以下输出：





1. 全部结束只需要输入docker-compose down即可，容器将停止，网络将被删除，而若要删除数据卷，需要添加--volumes标志