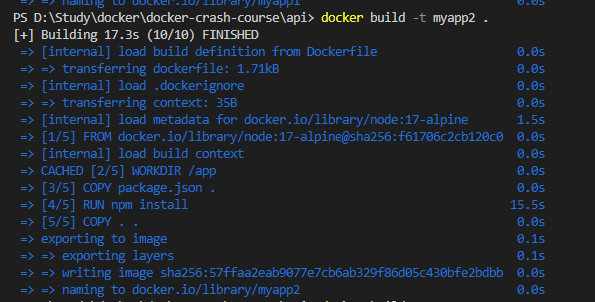
Docker

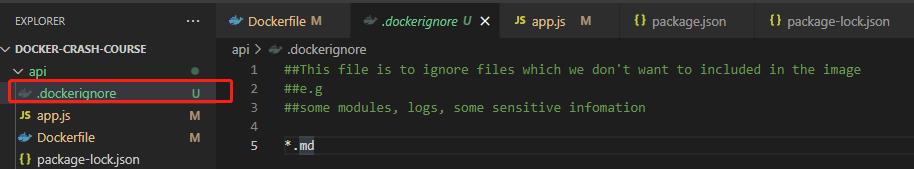
# Dockerfile



# 使用Docker caching优化代码



# 使用.dockerignore文件去ignore一些文件，不用copy进入container



# docker build

建立image的代码

# container的使用

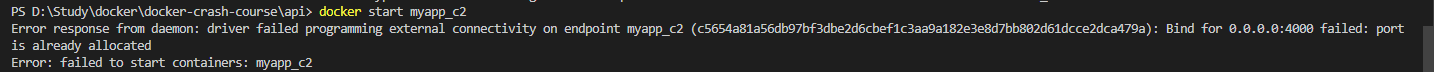
没有-d

descript

有-d

descript

有时候会报错如下，其实是端口被别的container使用了



# Image的使用

# delete all images, containers, volumes

descript

# Volumes

如果我们修改了app.js里头的内容，但是考虑到没有rebuild image, 那么继续运行image和container是不能反映app.js的变化的，这时候要不就是rebuild image, 要不就是用到volumes

Volumes can be useful for that directory mapping between the project in the local computer and the container so that we can see the changes without rebuilding the images.

1. 现在我们修改app.js的内容从：

'listening for requests on port 4000'

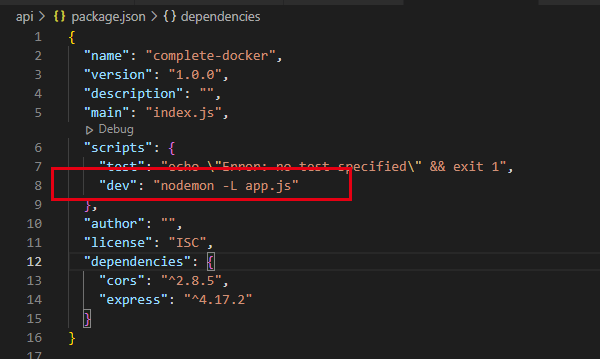
to

'listening for requests on port 4000......'

1. 修改app.js文件

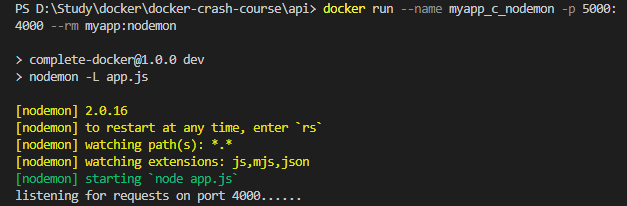
3）修改package.json文件

注意：在package.json的dev处增加了nodemon -L app.js， -L是因为我的电脑是windows系统，必须加



4）rebuild image

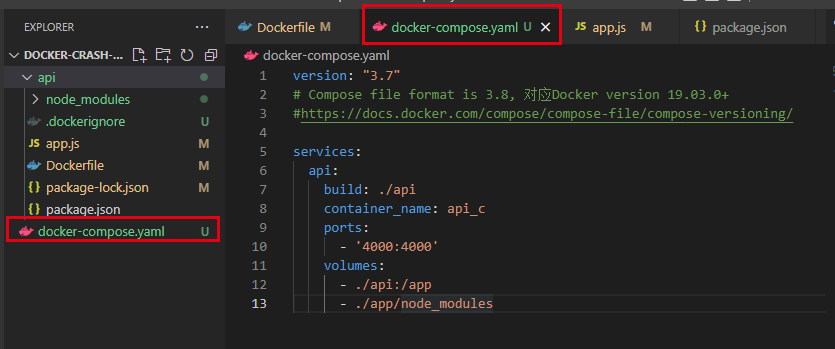
5) 新建一个container



6) 如果app.js一直有改变，可以使用

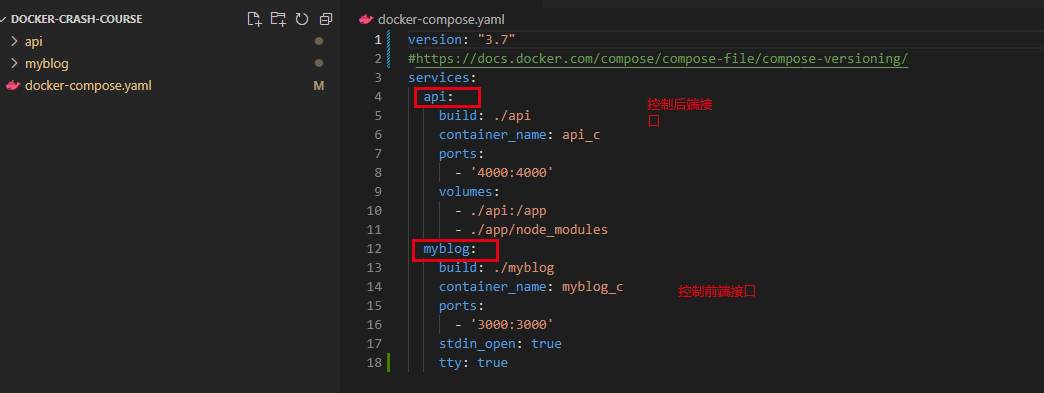
# Docker Compose

每次写docker build, docker run之类的语句不organized， 可以通过在根目录下写docker-compose.yaml文件去organize



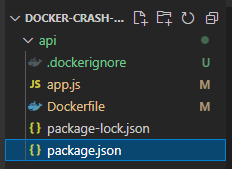
写完之后，在docker-crash-course这个根目录下运行下面代码即可自动生成images&container&volumes

# 利用两个container分别控制前端后端



# 附录

实验文件包括



app.js内容

package-lock.json内容

package.json内容