MAINTAINER：gongjunru [gongjunru@sjtu.edu.cn](mailto:gongjunru@sjtu.edu.cn)

整个项目部署完成后，于此鸣谢部署过程中提供帮助的所有同学和老师，并记录下路上遇到的惊心动魄小bug与丧尽天良大生态。

数据库部署：

技术栈：docker, navicat

流程：

1. Debian10服务器中安装docker，命令详见菜鸟教程[Docker 教程 | 菜鸟教程 (runoob.com)](https://www.runoob.com/docker/docker-tutorial.html)

踩过的坑：本人不才试图在windows环境下学习docker，花了两个小时还没配好环境后幡然悔悟，docker并不是为windows准备的。遂切换至本机Ubuntu系统，五分钟就安装好了docker。

1. 拉取所需镜像，因为我们没有使用非关系型数据库，这一步变得简单了很多。

>docker pull mysql

等待下载即可，下载完成后可用docker images命令查看镜像拉取有没有成功

1. 用mysql镜像建立容器并运行：

>docker run -it mysql

4. 进入mysql容器内

>docker exec -it mysql /bin/bash

5. 进入后可正常使用mysql命令管理自己的数据库

踩过的坑：是不是只有我这种傻子，会真的试图在一个终端里管理自己的数据库。

明明navicat提供了连接服务器上数据库的功能。



Navicat连接到SSH：

主机：服务器弹性IP地址

端口：默认22，华为云那边可以改。

用户名和密码：SSH服务器的用户名和密码！

配置好SSH连接后，SSH与数据库之间的连接与平日里本机连接数据库无异，输入数据库的用户名和密码即可。

至此，可在本地navicat上实现对服务器端数据库的增，删，改，查

数据库暴露IP为服务器弹性公网IP，暴露端口默认为3306，数据库部署完成。

后端部署：

技术栈：docker，jar

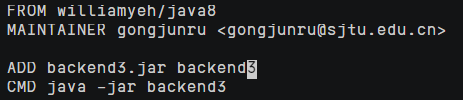
流程：

1. 将springboot项目打包成.jar包

踩过的坑：

package会先进行一次快照测试，并运行所有编写的测试。如果测试没有通过，它不会进行后续的打包工作。大多数情况下，测试环境与后端的运行环境不同，所以打包时要记得先把测试类删掉（bushi

1. 将jar包放进服务器，华为云提供的登录系统和mobaxTerm等终端都支持本地与服务期间的文件拖拽传递。
2. 编写dockerFile，为一会儿生成镜像做准备，dockerFile细节请移步菜鸟教程，这里只以我的dockerFile为例介绍：



第一行：以java8镜像为爸爸镜像创造自己的镜像，对后端来说对java8有依赖很正常

第二行：说明这个dockerFile是谁写的，出了bug锅应该给谁

第三行：将jar包添加到一个名为“backend3”的镜像中

第四行：镜像被创建为容器时运行命令：java -jar backend3

1. 生成镜像：

docker build -f ./dockerFile -t backend3.0 .

其中，-f表示自定义dockerFile路径与名称，因为默认的dockerFile路径比较奇怪

-t表示自己命名新的镜像名称 后面加一个. 我其实不知道这个“所有”指的是是啥意思，我觉得或许是运行dockerFile中所有命令？

5. 用镜像创立容器：

docker run -it backend3.0

-it搞出来的容器会在前台实时汇报后端启动情况，便于debug，看到后端部署成功就万事大吉啦

踩过的坑：

1. 学习过程中，我的后端代码有一个地方写了绝对路径，指向升级为https协议的license，但这直接导致在本地可以顺利运行的后端，打包成jar包后，在服务器端无法正常运行。这个bug困扰了我一整天的时间，以后再写绝对路径我就是狗。
2. 学习过程中，我试图将后端与数据库部署在同一台服务器上，操作天衣无缝没有错误，但是docker的不同容器之间有非常强的隔绝，这正是docker的特性和优越性所在。然而我并不知道这一点，于是并不知道为什么我的后端没有连接到数据库上。这个问题本质上是容器间通信问题，不过由于docker创建的镜像中过于简陋，我在搭建好容器间的通信网络后，并不知道为什么失败，而且没有能力知道报错信息（因为容器内没有apt,无法安装任何用于测试链接的工具，如ping），所以我们使用了更加生草的解决方式：

钞能力：把后端和数据库部署在两台服务器上！

前端部署：

技术栈：npm\_serve, nginx, 杨景凯同学

先吐槽：

nginx作为行业内用了都说好的React项目部署工具，自然吸引了我们的注意。

但我们万万没有想到，在部署过程中发现了nginx的极端恶劣的生态bug，而且不止一个。

罄竹难书，但我会将此事娓娓道来的。

流程：

1. 将前端打包：使用react run build命令，获得build文件夹
2. 开始我是把整个前端项目都转移进了服务器中，在build文件夹中运行serve -p 3000，获得了一个很丑陋的前端部署结果。

踩坑：这种部署在刷新时会导致前端页面崩溃，因为系统把刷新时的url整体看作一个url来处理，这会导致路由的管理失败。

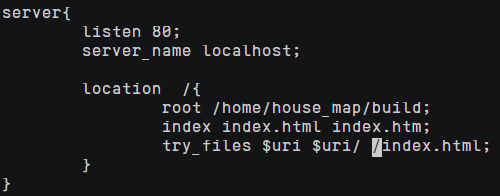
解决：使用了nginx技术。但这也正是罄竹难书的开始。

1. 安装nginx并配置nginx环境，在哪装都装的上，它环境也好配，这里就不赘述了。
2. nginx有两个关键文件夹：nginx程序所在文件夹，和nginx程序的配置文件所在文件夹

在我的服务器上，这两个文件夹分别为：/usr/sbin和/etc/nginx

我查遍网络上的教程，它们都讲nginx的部署只要更改/etc/nginx中的配置文件：nginx.conf就可以了

更改内容：在http段中增加server部分：



语法挺浅显，两个需要注意的地方是：

第一 root后路径为build文件夹所在路径

第二 try\_files那一句正是为了解决上述的刷新崩溃问题。它的原理是：

每次刷新崩溃没有识别出url时，就去uri/这个目录下寻找路由。如果再没找到，就回到index.html中找路由。而index.html作为初始路由，会一路将该url按push顺序定位到应有的位置，这就解决了刷新崩溃的问题……吗？

当然没有，距离解决问题还有两个巨大的坑

第一坑：package.json中默认的home路径为空，这会导致一切home转/

但正常人，我说的是正常人写出来的路由，是要从./开始的

这一个微微的小差异在本地可能无足轻重，但是丢到服务器上就是毁灭性的。所以要把homepage改为“.”，重新打包build，才能正式解决刷新崩溃问题……吗？

第二坑：做完了上述的一切后，我们依旧没有解决问题，甚至没有看到nginx成功部署起来。

实在手足无措的时候，杨景凯同学出现了，真是人间太岁神，过来三下五除二发现了问题所在：

他使用systemctl status敏锐地发现，nginx没有依赖nginx.conf作为配置文件，而是依赖于这个奇怪目录下的奇怪文件：

这个事儿网上可没有人写博客出来说！



这个奇怪的文件中，有着另一个奇怪的文件夹和root命令：



我们实在没有看懂他在干什么，于是我们做出了以下改动：

1. 委曲求全，将我们的build文件夹移动到/var/www目录下
2. 在这个文件中也进行端口，try\_files等一系列我们在.conf文件中进行的改动

然后我们非常不开心的发现还是不行

就回到了nginx.conf中，百无聊赖中发现，欸

为什么他的用户名，不是root呢？

我们修改了用户名，部署圆满成功。

这哪个正常人做到这里不得惊呼一声我超，这nginx什么垃圾生态

到这里为止已经解决了所有的问题，但依旧遗留下了一个疑问：

难道是因为这个用户名的设置，导致了nginx在面对我们这个root用户时，没有启用nginx.conf，而是启用了default配置文件吗！？？

于是我尝试不修改default配置文件，只修改nginx.conf，但是部署失败。猜想是错误的。

这里面到底发生着什么，我不得而知。