从零开始部署pigx

Ver 1.0.1

PigX内部群资料

目录

写在前面

鉴于群里不少萌新总想着先在线上部署一版但是又不得其门而入，故而准备了这篇教程。

我使用Oracle的VirtualBox虚拟了一个系统，核心关键的参数如下：

参数	示例环境
操作系统	~~ArchLinux，已滚动到最新~~ 现已换成大家喜闻乐见的CentOS
内存	8G
CPU	2Core I7-2860QM

因为从零开始，安装各种环境实在是太麻烦了,也为了下一篇讲docker的方便，我将通过docker安装各种必须的生产环境。

同时~~因为懒，并且这个是一个虚拟机系统且在我已经做好虚拟机快照的情况下，我将直接通过root账户进行操作，因为我有恃无恐呀。~~ 我将新建一个名为`pigx`的用户并赋予`sudoer`权限，在任何情况下，我们都不应该直接通过`root`账号进行操作。

注意:

“root敢死队”不是一个好习惯，这个是一种错误的姿势！在生产环境中，应该禁止`root`用户的远程登录权限，同时涉及到需要root权限操作的问题应该通过`sudo`上去才是合理的。

运行环境搭建

系统环境准备

一台真实的线上服务器，或者虚拟机，案例中将会用于演示的系统是最小化安装的最新的CentOS7。我采用的是这个镜像。具体系统的安装过程略过不表。如果CentOS装不上的话，那~~么建议砸钱去买能自动装系统的云服务器。~~ 那么可能这篇教程可能并不适合现在的你。

不管是网络地址转换的方式还是桥接的方式，要保证的就是安装的CentOS能够联网。如果想要模拟更真实的机器环境可以用桥接，我的案例中使用的将是桥接的方式，桥接方式存在的一个问题是ip可能会随着网络环境的变化而变化，至于这个问题的解决方案网上有很多，本文将不进行讨论。执行以下命令将系统环境升级到最新：

```
sudo yum -y update
```

这样系统的准备工作就算做完了。

运行环境准备

pigx这个项目，以目前最新的稳定版本1.6.0为例，大概需要以下的组件:

环境	版本	说明
jdk	1.8	undertow容器在最新的jdk10上好像还是有点问题，因此这次的案例将会采用1.8的版本
maven	3.5.4	后端项目构建管理
node	10.10.0	前端项目构建管理
docker	18.06.1-ce	项目中需要用到的第三方组件我们都将通过docker进行安装
mysql	5.7.22	mysql提供数据库支持，mysql8需要更换连接驱动jar包，所以这里我们还是以mysql5.7为例

环境	版本	说明
redis	4.0.11	缓存中间件
nginx	1.15.3	静态资源服务器、反向代理服务器
zookeeper	3.4.13	分布式应用程序协调服务
git	2.19.0	可选，我的演示将不会通过将本地的代码上传至Linux的方式，所以需要git来克隆最新的代码

上面的这些软件的版本基本上都是写文章时最新的一个版本。

除了jdk,maven,node和git以外的所有工具我们都将通过docker进行安装。

安装jdk

我采用的是最小化安装，所以什么卸载jdk之类的就直接省了。如果你不是，自行卸载吧。
在jdk官网下载Linux下可用的jdk,我采用的是`jdk-8u181-linux-x64.rpm`这个包,然后`cd`到你上传的目录直接一条命令下去：

```
sudo rpm -ivh jdk-8u181-linux-x64.rpm
```

jdk安装完毕了。不放心的话可以通过:

```
java -version
```

确认一下。

安装node.js

下载并解压node包

这里采用的是node的二进制安装包，安装位置我将会采用`/opt/node`这个路径，该路径不存在的话可以直接通过:

```
sudo mkdir /opt/node
```

新建该路径。

`cd`到/opt/node之后，直接通过命令：

```
sudo wget https://nodejs.org/dist/v10.10.0/node-v10.10.0-linux-x64.tar.xz
```

就可以完成node二进制包的下载。

然后通过:

```
sudo tar -xvf node-v10.10.0-linux-x64.tar.xz
```

emmm,这个目录不喜欢，处理一下:

```
sudo mv node-v10.10.0-linux-x64/* /opt/node|sudo rm -rf node-v10.10.0-linux-x64 node-v10.10.0-linux-x64.tar.xz
```

这样，node的Linux二进制文件都应该位于`/opt/node`这个路径下了

配置node环境变量

输入：

```
sudo vim /etc/profile
```

编辑全局的环境，添加环境变量：

```
export NODE_HOME=/opt/node

export PATH=$PATH:$NODE_HOME/bin

export NODE_PATH=$PATH:$NODE_HOME/lib/node_modules
```

之后通过：

```
source /etc/profile
```

使当前更改生效即可。

Tips:

上面的设定是全局有效的，可能有人不喜欢这种方式，那么直接使用`vim ~/.bash_profile`编辑当前用户的配置即可。

这样,node的安装基本上就算完事了，想要确认的话，输入

```
npm -v
```

或者

```
node -v
```

能够看到输出基本上就可以确定安装没有问题。

安装其他软件

其他的软件可以通过一条命令搞定。

```
sudo yum install git  maven docker -y
```

配置镜像

由于众所周知的原因，默认配置下，docker，npm之类的下载速度非常感人，所以我们需要配置国内镜像。

npm镜像配置

npm的话一条命令就可以搞定：

```
npm config set registry https://registry.npm.taobao.org
```

通过以下命令确定配置是否生效:

```
npm config get registry
```

docker镜像配置

新版的docker只需要键入命令:

```
sudo vim /etc/docker/daemon.json
```

输入以下内容:

```
{
     "registry-mirrors": [
        "https://docker.mirrors.ustc.edu.cn"
    ]
}
```

然后保存退出，通过:

```
sudo systemctl restart docker
```

重启docker，之后通过:

```
sudo docker info
```

查看docker当前的配置，如果能够在打印出来的信息中看到：

```
Registry Mirrors:
 https://docker.mirrors.ustc.edu.cn
```

那么就证明配置成功

Tips:

强烈建议通过：

```
sudo systemctl enable docker
```

使docker开机自启动。

maven镜像配置

maven的话这边就不进行配置了，因为`pigx`的顶级父工程`pom`中已经包含了阿里的镜像源。

用docker安装必备的软件

其他的一些必备的软件我们将通过docker进行安装。

Tips:

截至目前，docker并不具备自动在宿主机上建立目录的功能，所以需要映射的路径要自己提前建好，在本篇案例中，我们所有的docker映射的文件夹都将放在`/opt/docker`下

安装mysql

docker安装mysql

首先建立好数据卷映射到宿主机的路径：

```
sudo mkdir -p /opt/docker/mysql
```

然后一条命令搞定mysql:

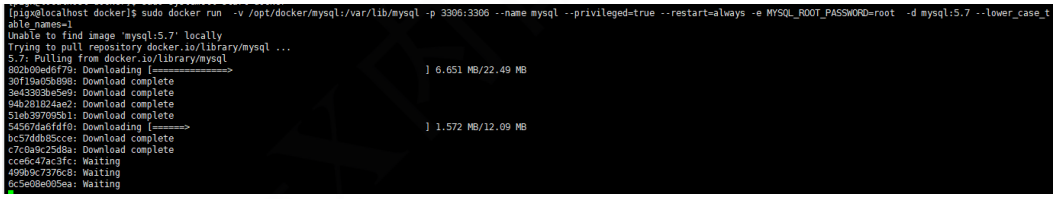
```
sudo docker run  -v /opt/docker/mysql:/var/lib/mysql -p 3306:3306 --name mysql --privileged=true --restart=always -e MYSQL_ROOT_PASSWORD=root  -d mysql:5.7 --lower_case_table_names=1
```

可能有人没有接触过docker，我解释一下上面的这条命令：

整条命令以`docker run`开头，代表创建一个新的容器并运行一个命令，

- `-v`参数代表将docker里的文件映射到宿主机上，格式为`宿主机文件路径:容器中的文件路径`，避免每次重启容器时数据被销毁
- `-p`参数表示端口映射，格式为：`宿主机端口:容器端口`
- `--name`表示`为容器指定一个名称`
- `--privileged=true`这个参数可以`使container内的root拥有真正的root权限。否则，container内的root只是外部的一个普通用户权限。`
- `-e`表示`设置docker容器的环境变量`
- `--restart=always`可以在任何情况下都使docker容器随docker的启动而自动重启,- `-d`表示`以后台运行容器，并返回容器ID`
  一般跟在最后的就是要拉取的容器了，格式为：`镜像名:标签`，默认拉取`lastest`标签，这里我们要手动指定一下,不然拉取的mysql会是8.0的版本，因为只指定了mysql5.7这个大版本，所以默认会拉取最新的小版本。
  一般的docker指令到这里就可以结束了，这条指令还有一句：
- `--lower_case_table_names=1`是由这个mysql容器提供的能力,使mysql对数据库和表名大小写不敏感。

出现以下界面安心地等着就行：



拉取完后容器会自动运行，`docker run`可以理解为整合`docker pull`和`docker start`的命令，然后我们通过:

```
sudo docker ps
```

当观察到`STATUS`为`UP`状态时，基本上mysql就已经可以使用了。

通过图形界面操作mysql

为了简化操作难度，我们可以在外部通过`DataGrip`或者`navicat`之类的软件进行操作。

Tips:

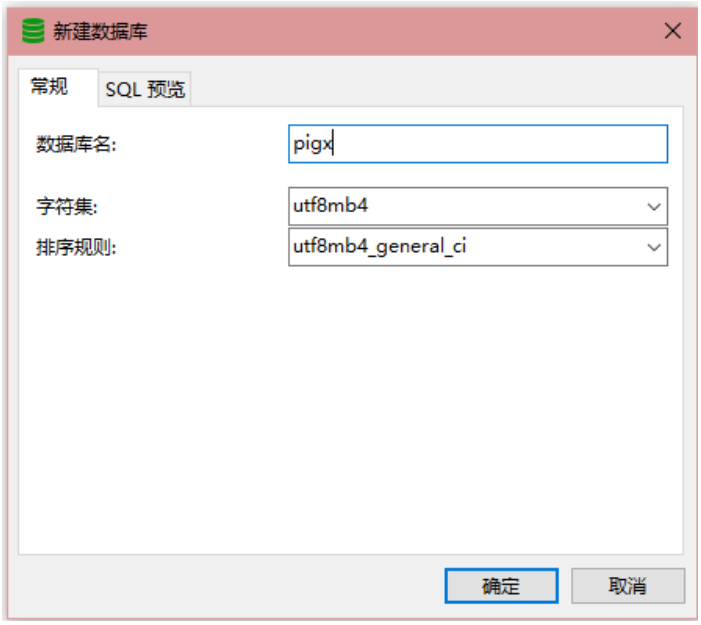
这里的主机IP是虚拟机的ip，而不能使用localhost,因为这并不是你主机上的docker容器内部的3306端口暴露给虚拟机上的CentOS,宿主机就是虚拟机上的CentOS,而跟真正的物理机没有一丁点关系。

当然有朋友表示不服，`docker for windows`虚拟的Hyper-V下的Linux怎么可以直接通过loclhost访问？那是因为除了上面的过程以外，还多了`端口转发`这一步，就是将docker已经暴露给虚拟机的端口进一步通过转发的方式转发到真正的物理机上，这样访问物理机的特定端口就相当于访问虚拟机的特定端口，而访问虚拟机的特定端口又相当于访问docker容器中的特定端口，感兴趣的朋友可以使用`NAT`的方式构建这个虚拟机，然后利用VBOX的NAT端口转发的功能就可以轻松验证这一点。

建立pigx数据库

数据库都有了，索性把数据库的东西处理完吧。

按照以下参数建立好数据库：



之后导入项目doc路径下的pigx.sql脚本。

安装redis

一行命令搞定：

```
sudo docker run  --name redis --publish 6379:6379  --restart always -d redis:latest
```

这里的`--publish`和上面简写的`-p`一样。

因为没有操作本地需要root权限才能操作的文件的需要，也就没有通过`--privileged=true`完全授权。

最后的`:lastest`不写其实也一点关系都没有。

安装zookeeper

一行命令搞定：

```
sudo docker run  --name zookeeper --publish 2181:2181 --restart always -d zookeeper:latest
```

安装nginx

拉取镜像

nginx的安装就略微有些麻烦了。

建立存放nginx文件的目录：

```
sudo mkdir -p /opt/docker/nginx|sudo mkdir -p /opt/docker/nginx/html|sudo mkdir -p /opt/docker/nginx/conf.d|sudo mkdir -p /opt/docker/nginx/log
```

先拉取镜像：

```
sudo docker pull nginx
```

一般情况下docker启动时进行配置，只要把配置文件的目录挂载出来就可以，简洁方便，但是nginx却是先加载一个主配置文件`nginx.conf`，在`nginx.conf`里再加载`conf.d`目录下的子配置

文件（一般最少一个default.conf文件）。这比单独挂载一个目录麻烦了不少。

下面是**最终**挂载好的命令：

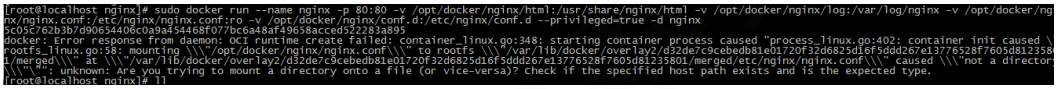
```
sudo docker run --name nginx -p 80:80  -v /opt/docker/nginx/html:/usr/share/nginx
/html -v /opt/docker/nginx/log:/var/log/nginx  -v  /opt/docker/nginx/nginx.conf:/
etc/nginx/nginx.conf:ro  -v /opt/docker/nginx/conf.d:/etc/nginx/conf.d --privileg
ed=true -d  nginx
```

注意：

1. 第一个“-v”，是项目位置，把项目放到挂载到的目录下即可；
2. 第二个“-v”，是挂载的主配置文件"nginx.conf"，注意"nginx.conf"文件内有一行"include /etc/nginx/conf.d/*.conf;"，这个include指向了子配置文件的路径，此处注意include后所跟的路径一定不要出错。
3. 第三个“-v”，把docker内子配置文件的路径也挂载了出来，注意要与（2）中include指向路径一致
4. 重点强调一下，nginx.conf是挂载了一个文件（docker是不推荐这样用的），conf.d挂载的是一个目录

目前如果尝试启动的话，会发现是有问题的，因为配置文件还没有。

你将得到以下这个界面：



是的，能看到这里，你不用怀疑，你的确已经凉了。下面我们就着手解决这个问题。

删除挂载失败的nginx

先停止之前的镜像：

```
sudo docker stop nginx
```

删除之前的镜像：

```
sudo docker rm nginx
```

删除docker生成的nginx.conf目录：

docker镜像里的nginx.conf是个文件，而当你宿主机里不存在这个文件时，docker就会将这个路径映射成宿主机的一个目录，这就是为什么会报上面这个错误的原因，所以我们还需要删除宿主机上docker错误生成的**nginx.conf文件夹**：

```
sudo rm -rf /opt/docker/nginx/nginx.conf
```

配置nginx配置文件

我们找到常规方法安装的nginx时生成的配置文件（一般以“/etc/nginx”下），对应上面启动命令中的挂载位置，把主配置文件nginx.conf放到对应位置“/opt/docker/nginx/nginx.conf”(这个是一个配置文件，而不是类似于之前挂载失败的文件夹)，把子配置文件“default.conf”放到“/opt/docker/nginx/conf.d”目录下。

重新运行启动命令，就会发现已经没有问题了，至此docker中的文件已经可以随意配置，跟原生安装是一模一样的。

建议先建立nginx.conf，，然后建立conf.d下的default.conf,然后执行docker命令建立nginx镜像，这样不会报错。

具体的操作方式如下：

使用命令建立一个标准的镜像：

```
sudo docker run -p 80:80 --name nginx  -d nginx
```

因为没有外置nginx配置文件，所以这一次启动一定会成功。执行以下命令，观察你的nginx容器的CONTAINER ID

```
sudo docker ps
```

然后用确保你当前在/opt/docker/nginx这个路径下，执行以下命令：

```
sudo docker cp 3f2596f6ace3:/etc/nginx/nginx.conf nginx.conf
```

上面的3f2596f6ace3就是我的Contaienr Id，当然，如果觉得太麻烦不用container id，直接用container name也是可以达到同样的效果的:

```
sudo docker cp nginx:/etc/nginx/nginx.conf nginx.conf
```

注意观察，这个时候在你的/opt/docker/nginx这个路径下应该已经有了nginx.conf这个文件了。

利用同样的办法把default.conf复制出来：

```
sudo docker cp 3f2596f6ace3:/etc/nginx/conf.d/default.conf conf.d/default.conf
```

当然，需要提前在/opt/docker/nginx路径下建立好conf.d这个文件夹。再强调一次，docker没有自动在宿主机上建立文件夹的能力。

然后就可以通过以下命令干掉这个镜象了：

```
sudo docker rm -f nginx
```

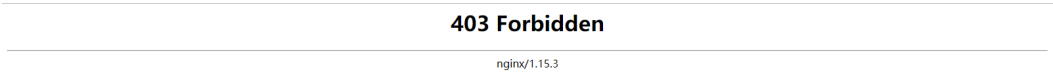
使用挂载命令，挂载我们真正想要的镜像：

```
sudo docker run --name nginx -p 80:80  -v /opt/docker/nginx/html:/usr/share/nginx
/html -v /opt/docker/nginx/log:/var/log/nginx  -v  /opt/docker/nginx/nginx.conf:/
etc/nginx/nginx.conf:ro  -v /opt/docker/nginx/conf.d:/etc/nginx/conf.d --privileg
ed=true -d  nginx
```

完事。

检查nginx是否安装成功

在物理机上直接访问虚拟机的IP,如无意外，我们将得到这个页面：



等等，403什么鬼啦。看来的确出了意外，不过我们似乎忘了做点什么。

cd到放置静态资源的html目录，果然里面空空如也，这就好办了！

直接通过:

```
sudo vim index.html
```

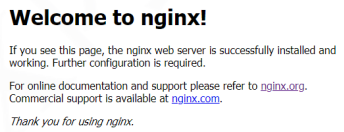
编写一个html文件：

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

现在再访问一次我们的页面试试？



自此环境全部搭建完毕。

pigx搭建实战

经过前面这么多的步骤，我们终于具备了一个最小的pigx运行环境了，现在，让我们马上开始吧。

pigx后端搭建实战

我们先来处理后端的问题。

获取源代码

对于源码的这些操作我都将在自己的家目录下进行，所以先通过：

```
cd ~
```

切换到自己的家目录，然后一条命令搞定后端源码的获取：

```
git clone https://gitee.ltd/pig/pigx.git
```

可能会要求私服的账号和密码，如实填写即可。

配置hosts文件

从1.6.3开始，pigx采用了hosts文件来管理数据库、网关、注册中心的地址，因此，请确保你的hosts中存在相关映射:

```
# 本地测试环境
你的公网或者虚拟机IP   pigx-mysql
你的公网或者虚拟机IP   pigx-zookeeper
你的公网或者虚拟机IP  pigx-redis
你的公网或者虚拟机IP pigx-gateway
你的公网或者虚拟机IP pigx-eureka
```

以开发时使用的本地环境为例，则配置如下：

```
# 本地开发环境
127.0.0.1    pigx-mysql
127.0.0.1    pigx-zookeeper
127.0.0.1    pigx-redis
127.0.0.1    pigx-gateway
127.0.0.1    pigx-eureka
```

编译打包pigx工程

通过：

```
cd pigx
```

进入pigx的工程，直接输入：

```
mvn package
```

然后等着就行了。

反正也还在打包，我们可以先扯几句:Spring Boot工程有一个特点，我们直接通过`java -jar`的方式进行运行的操作并不是后台运行模式。如果不用后台运行模式的话，当前窗口一旦关闭，那么工程也就凉了。

所以可以通过以下的方式运行，这里直接提供对应的`start.sh`脚本，内容如下：

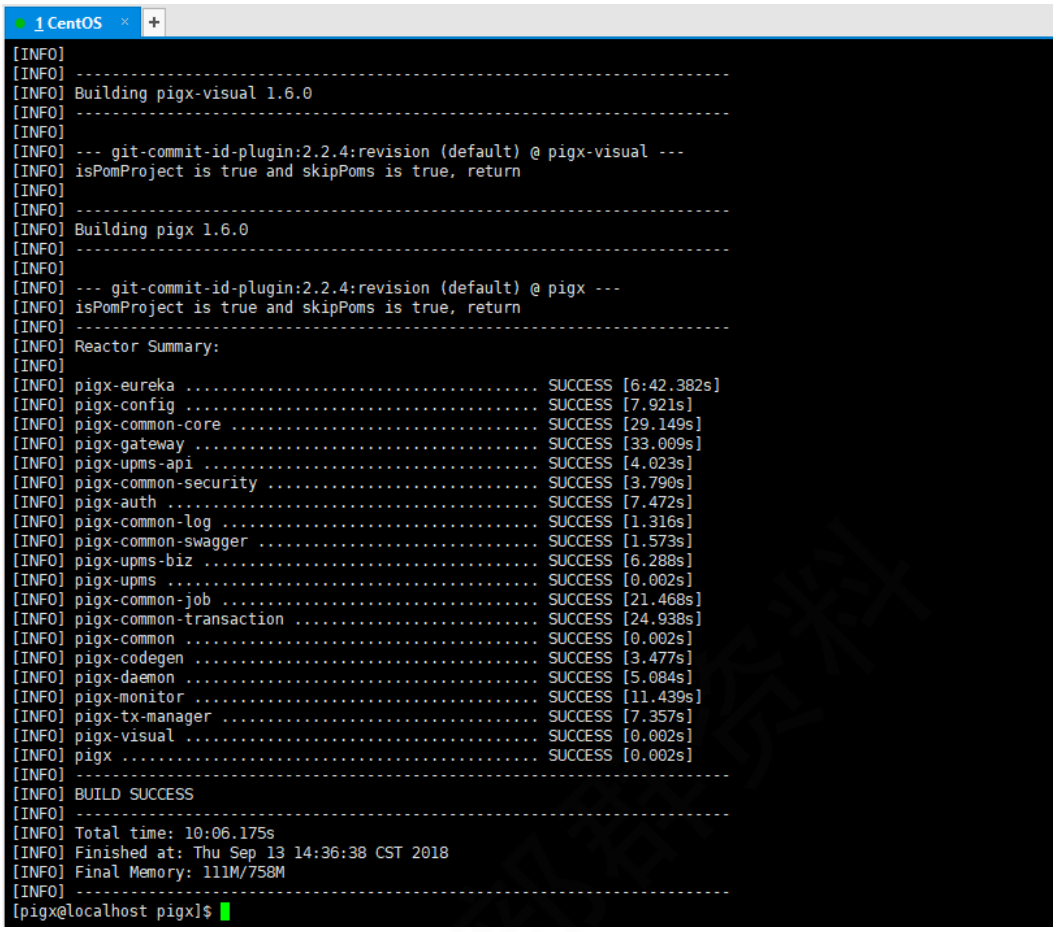
```
#!/bin/bash
nohup java -jar yourapp.jar &
```

关闭应用的脚本`stop.sh`如下：

```
#!/bin/bash
PID=$(ps -ef | grep yourapp.jar | grep -v grep | awk '{ print $2 }')
if [ -z "$PID" ]
then
    echo Application is already stopped
else
    echo kill $PID
    kill $PID
fi
```

将这里的`yourapp.jar`换成打包出来的具体工程名就行。

这个时候也文件也打包好了。



运行pigx工程

我在自己的家目录下建立一个`app`文件夹，用来存放打出来的jar包。因此在pigx的工程文件夹下，我通过执行：

```
mv pigx-eureka/target/pigx-eureka.jar ~/app| mv pigx-config/target/pigx-config.ja
r ~/app|
mv pigx-auth/target/pigx-auth.jar ~/app|
mv pigx-gateway/target/pigx-gateway.jar ~/app|
mv pigx-upms/pigx-upms-biz/target/pigx-upms-biz.jar ~/app|
mv pigx-visual/pigx-daemon/target/pigx-daemon.jar ~/app|
mv pigx-visual/pigx-monitor/target/pigx-monitor.jar ~/app|
mv pigx-visual/pigx-codegen/target/pigx-codegen.jar ~/app|
mv pigx-visual/pigx-tx-manager/target/pigx-tx-manager.jar ~/app
```

然后切换到`app`文件夹下，按照先前的启动和关闭模板，写个脚本，以eureka为例：

start-eureka.sh:

```
#!/bin/bash
nohup java -jar pigx-eureka.jar &
```

通过以下命令运行eureka:

```
./nohup: appending output to ‘nohup.out’
```

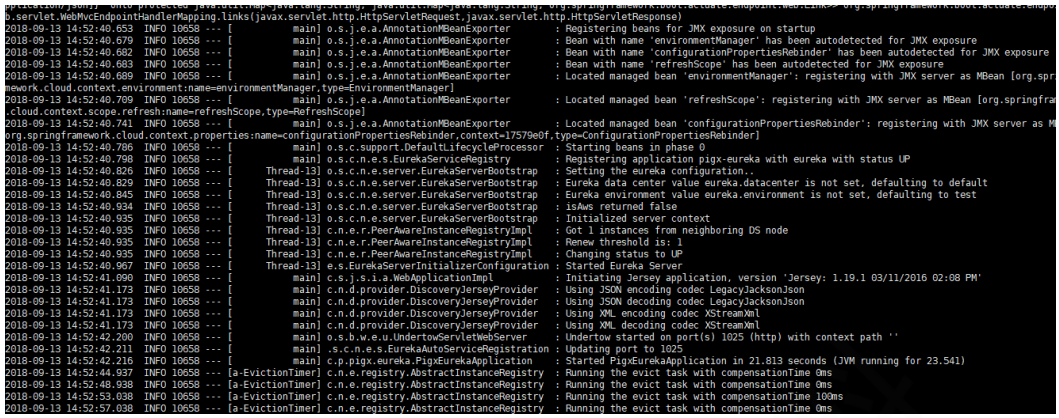
可以看到有一行输出：

```
nohup: appending output to ‘nohup.out’
```

这个nohup.out就在当前文件夹下，直接通过:

```
tail -f nohup.out
```

就可以监控到pigx-eureka的启动过程了。



能够看到这里就证明启动成功了。

我们通过宿主机访问一下看看，我目前的ip是:192.168.1.112，于是直接访问:http://192.168.1.112:1025

可以看到，其实毫无响应:

推测是防火墙问题，可以配置防火墙解析规则，阿里云可以设定安全组规则，我反正只是虚拟机，就采用暴力的彻底关闭防火墙的做法：

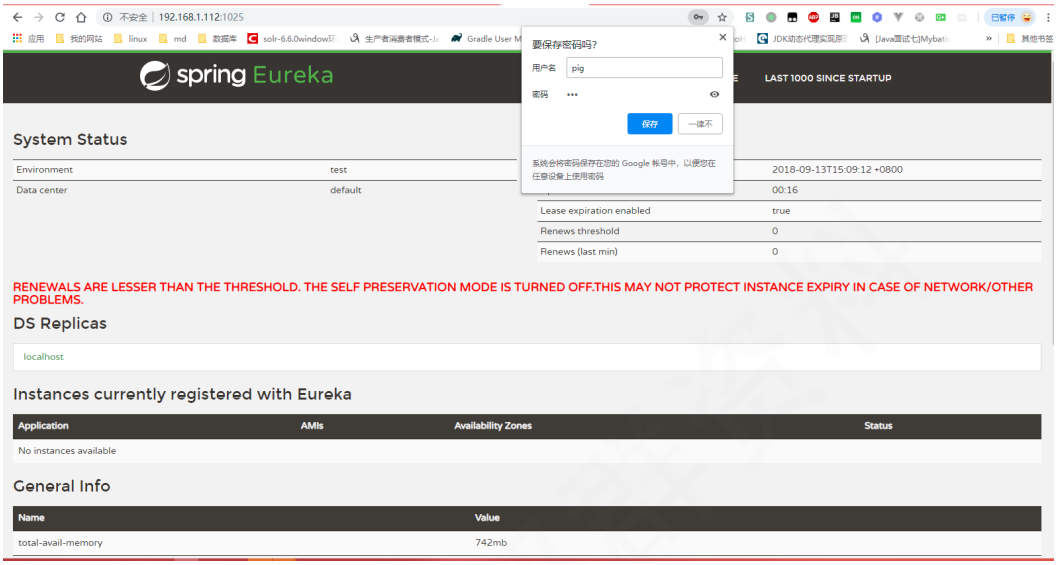
```
sudo systemctl stop firewalld
```

当然，也可以禁止它的开机自启动：

```
sudo systemctl disable firewalld
```

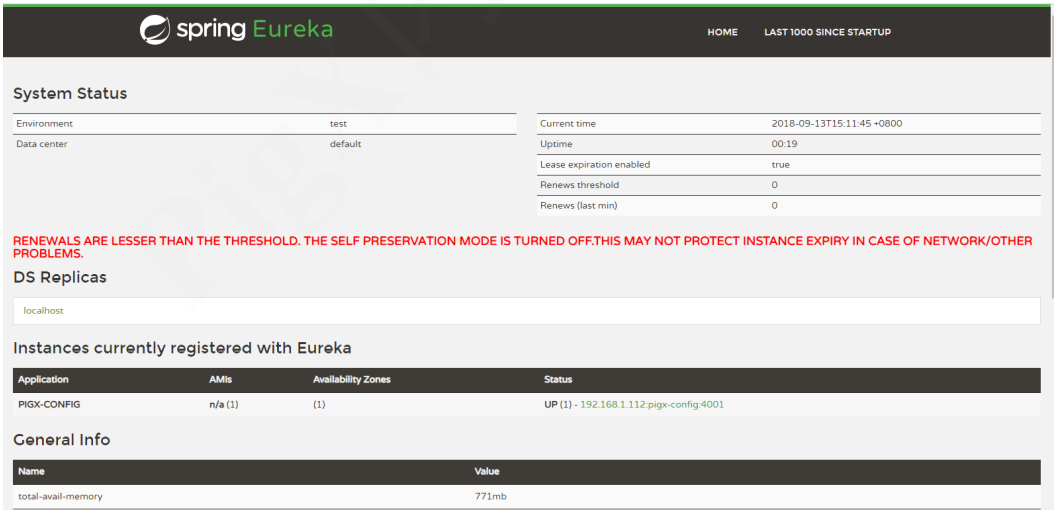
关闭防火墙后再访问:http://192.168.1.112:1025，可以看到已经妥了:

输入默认的用户名:pig,密码:pig,就可以看到这个熟悉的界面：



之后，我们可以用同样的方式启动pigx-config。

说实话，个人觉得启动命令不写成脚本，自己手动运行一下的效率也不错。



可以看到，pigx-config已经被安排上了。

然后运行pigx-auth工程。

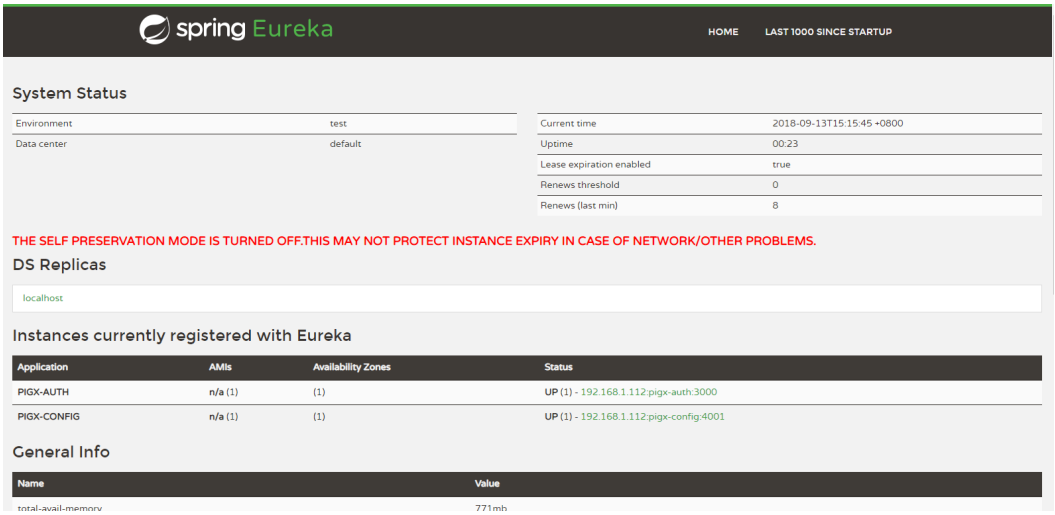
个人强烈建议在输入完

```
nohup java -jar pigx-module.jar &
```

之后立马通过

```
tail -f nohup.out
```

监控运用的执行情况。

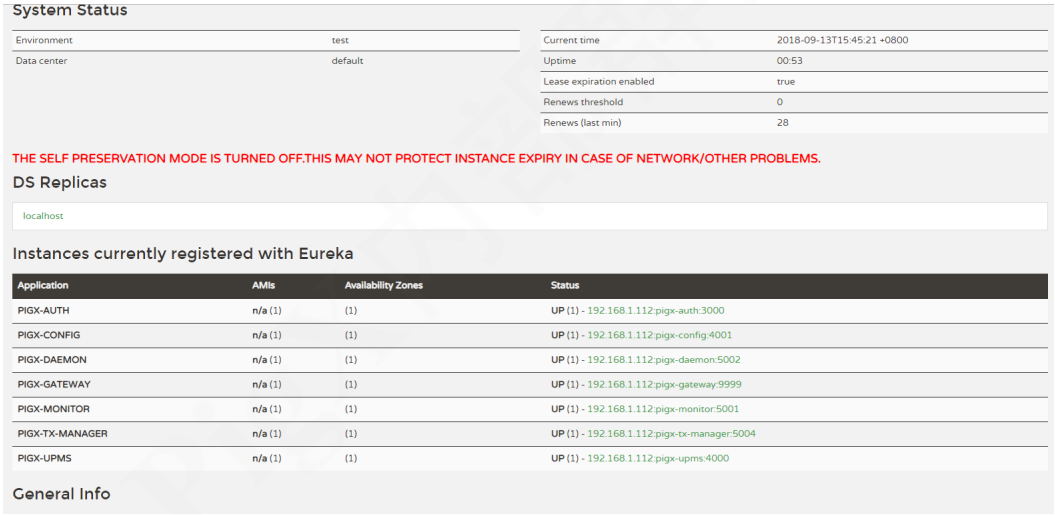


可以看到，pigx-auth也被按排上了。

之后，再用同样的方式把pigx-gateway,pigx-upms-biz,pigx-daemon,pigx-codegen,pigx-monitor,pigx-tx-manager。

当然，机器性能有限，所以我要做点限制jar包运行内存得手脚了：

```
nohup java -Xms10m -Xmx200m -jar pigx-module.jar &
```



可以看到，只要你明明白白，它们也都会被安排得明明白白。

注意：

- pigx-gateway需要redis配合
- pigx-auth,pigx-daemon-pigx-upms需要数据库配合
- pigx-daemeon需要zookeeper配合
- 所有的子模块需要pigx-eureka和pigx-config的配合
- pigx-config也需要pigx-eureka配合

所以必须要保证pigx-eureka启动成功，之后pigx-config要启动成功,其余的工程倒是没有启动顺序上的严格规定，只要保证其所依赖的服务没有凉就行。

pigx前端搭建实战

获取源代码

对于前端源码的这些操作我还是在自己的家目录下进行，一条命令搞定前端源码的获取：

```
git clone https://gitee.ltd/pig/pigx-ui.git
```

如果需要密码如实填写私服的用户名和密码即可。

编译打包pigx前端工程

进入下载的pigx-ui前端工程目录，执行：

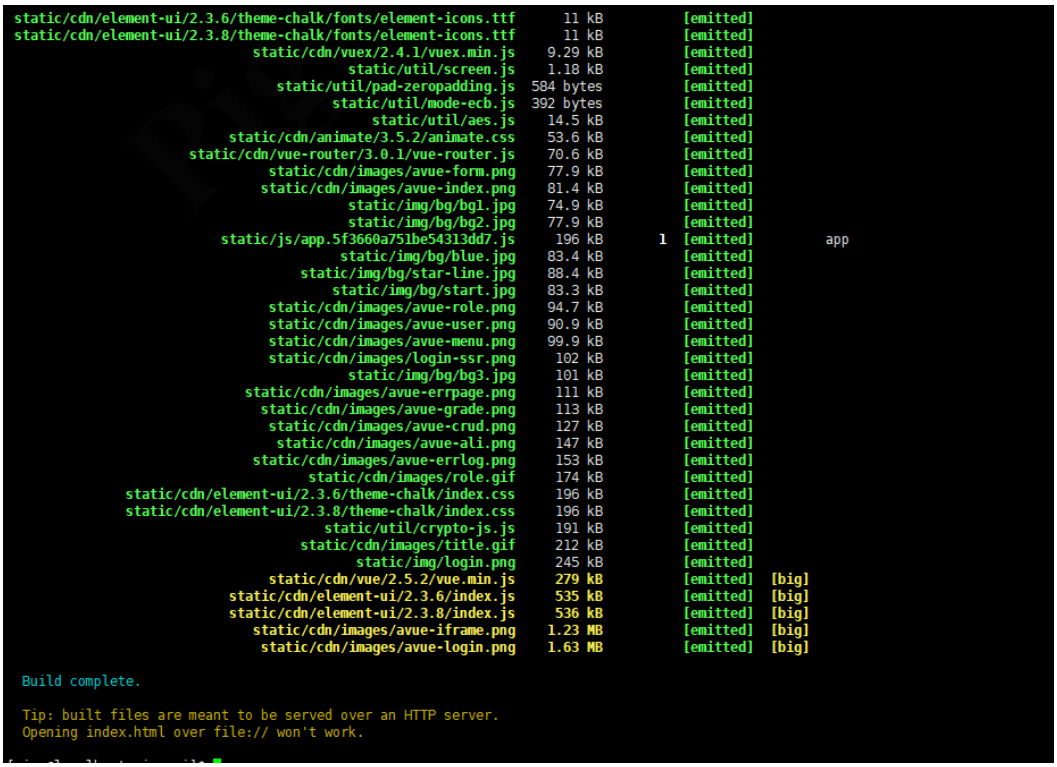
```
npm install
```

可以看到依赖已经安装完毕了，警告都是关于包过期的警告，大可直接忽视。

然后一条命令打包生产用的文件：

```
npm run build
```

看到这个界面基本上就已经成功了:



运行pigx前端工程

运行pigx前端工程很简单，就两步：

1. 配置nginx反向代理
2. 将编译后的文件移动到我们的静态资源目录中

让我们开始吧!首先处理一下nginx的配置文件。

按照开发环境下的vue-router配置nginx反向代理路径，具体规则在`pigx-ui\config\index.js`这个文件中。
所以我们`cd`到`/opt/docker/nginx/conf.d/`将default.conf复制一份后进行修改一番即可，我这里直接放出处理好的文件:

pigx-ui.conf

```
server {
    listen       80;
    server_name  localhost;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    location ^~/admin/ {
        proxy_redirect              off;
        proxy_set_header            Host $host;
        proxy_set_header            X-real-ip $remote_addr;
        proxy_set_header            X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass                  http://localhost:9999/admin/;
    }


    location ^~/auth/ {
        proxy_redirect              off;
        proxy_set_header            Host $host;
        proxy_set_header            X-real-ip $remote_addr;
        proxy_set_header            X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass                  http://localhost:9999/auth/;
    }

    location ^~/code/ {
        proxy_redirect              off;
        proxy_set_header            Host $host;
        proxy_set_header            X-real-ip $remote_addr;
        proxy_set_header            X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass                  http://localhost:9999/code/;
    }

    location ^~/gen/ {
        proxy_redirect              off;
        proxy_set_header            Host $host;
        proxy_set_header            X-real-ip $remote_addr;
        proxy_set_header            X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass                  http://localhost:9999/gen/;
    }


    location ^~/tx/ {
        proxy_redirect              off;
        proxy_set_header            Host $host;
        proxy_set_header            X-real-ip $remote_addr;
        proxy_set_header            X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass                  http://localhost:9999/tx/;
    }

    location ^~/daemon/ {
        proxy_redirect              off;
        proxy_set_header            Host $host;
        proxy_set_header            X-real-ip $remote_addr;
        proxy_set_header            X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass                  http://localhost:9999/daemon/;
    }

}
```

不用当心加载不了，主配置文件`nginx.conf`中已经包含了加载的conf.d的配置指令。

一个比较好的习惯就是修改完配置以后校验一下配置文件是否可以正常使用。

通过以下指令进入docker交互式终端：

```
sudo docker exec -it nginx bash
```

执行命令:

```
nginx -t
```

可以看到如下输出：

```
nginx: [warn] conflicting server name "localhost" on 0.0.0.0:80, ignored
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

因为是原样照抄的default.conf，所以loclhost:80这个服务器冲突了也很正常,但这只是警告，不影响什么，不放心的话可以在docker的交互式终端里执行：

```
mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.conf.bak
```

再执行:

```
nginx -t
```

可以看到那行警告已经消失了，这个修改虽然是在docker容器内部修改的，但是退出容器后依然有效。而至于为什么，可以看nginx.conf主配置文件的第31行，`include /etc/nginx/conf.d/*.conf`已经说明了一切。

尽管现在编译后的文件还没有扔到nginx中，我们也可以先执行一遍：

```
nginx -s reload
```

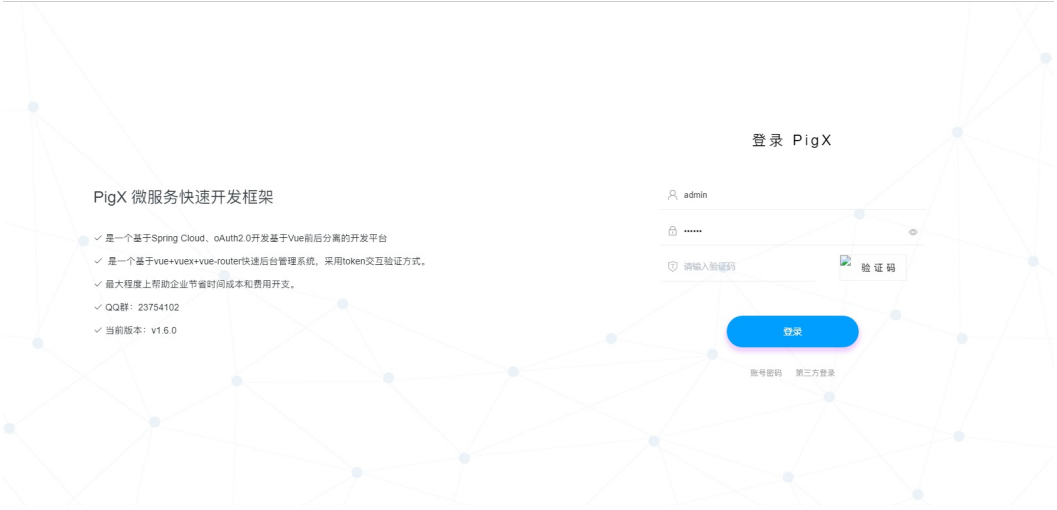
使新的配置文件生效，然后通过:

```
exit
```

退出交互式终端，通过命令:

```
sudo cp -r ~/pigx-ui/dist/* /opt/docker/nginx/html
```

然后直接访问:`http://192.168.1.112`,可以看到：



emmm，凉了。

注意：
开发环境下配置了很多的端口，一会儿8000一会儿9999，很多人就会感觉有点懵，其实抓住了问题的本质就很好理解。

一个非常简略的过程描述如下：
普通用户访问网站->实际上访问的是前端工程（前端工程有自己的端口）->前端工程按照转发规则对应路径转发到后台的网关工程（因为前端只配置了转发到网关模块的路径,见pigx-ui\config\index.js这个文件）->网关工程转发到各个微服务（转发规则）。

可以看到，前面的验证码并没有刷新出来，这个也很好解决。不要怀疑自己，通过控制台可以很清楚的看到，网关502报错，说明nginx的转发链路凉了，而直接访问后台到微服务的链路可以看到并没有问题，那是因为，这个nginx是跑在docker里的，并且docker网络默认是桥接模式，并不是和主机共享网络的host模式，因此解决办法也有两个：

- 指定docker运行时网络模式为host
- nginx反向代理你的公网IP或者虚拟机IP，不要代理localhost

更新日志

1.0.0

- 文档发布

1.0.1

- 提供docker中的nginx反向代理502的解决方案
- 提供pigx1.6.3开始的hosts文件配置说明
- 去除nginx认证模块相关的内容