Docker 部署 vue 项目

1.写在前面：

Docker 作为轻量级虚拟化技术，拥有持续集成、版本控制、可移植性、隔离性和安全性等优势。本文使用Docker来部署一个vue的前端应用，并尽可能详尽的介绍了实现思路和具体步骤，以方便有类似需要的同学参考。

> Docker 是一个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，该容器包含了应用程序的代码、运行环境、依赖库、配置文件等必需的资源，通过容器就可以实现方便快速并且与平台解耦的自动化部署方式，无论你部署时的环境如何，容器中的应用程序都会运行在同一种环境下。（更多详情请移步docker官网查看docker）

默认已经安装了 docker，@vue/cli

相关版本：

- Docker version 18.09.2, build 6247962
- vue cli --version 3.3.0
- macOS Mojave Verison 10.14.1

> 运行环境为macOS，如果与阅读者操作系统之间存在差异，请自行调整

相关镜像：

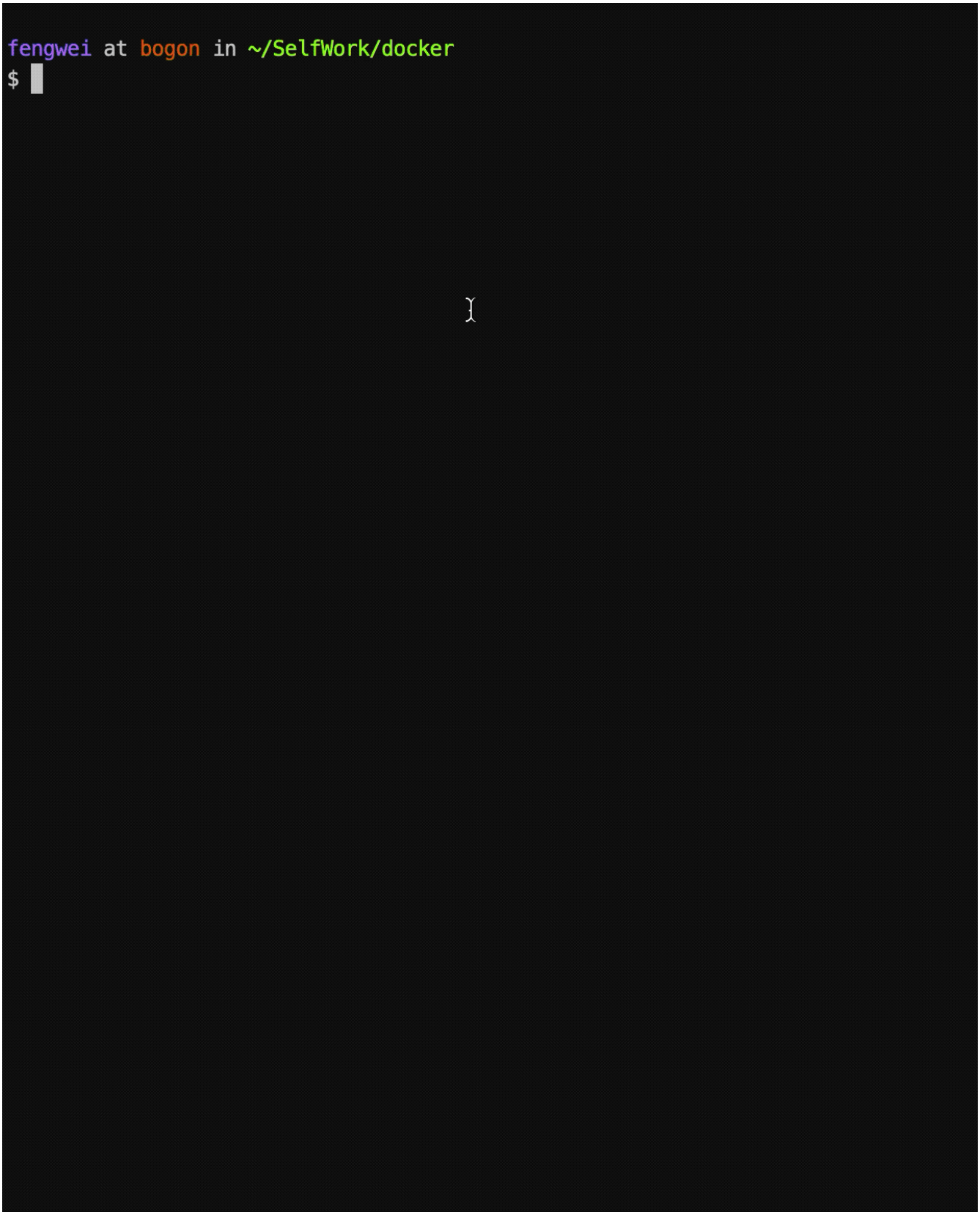
- nginx:latest
- node:latest

2.具体实现：

> 1. 用 vue cli 创建一个vue项目，修改一下创建出来的项目，在页面上写一个前端接口请求，构建一版线上资源 ，基于nginx docker镜像构建成一个前端工程镜像，然后基于这个前端工程镜像，启动一个容器 vuenginxcontainer。
> 2. 启动一个基于 node 镜像的容器 nodewebserver，提供后端接口。
> 3. 修改 vuenginxcontainer 的 nginx 配置，使前端页面的接口请求转发到 nodewebserver 上。
> 4. 稍作优化和改进。

3 创建 vue 应用

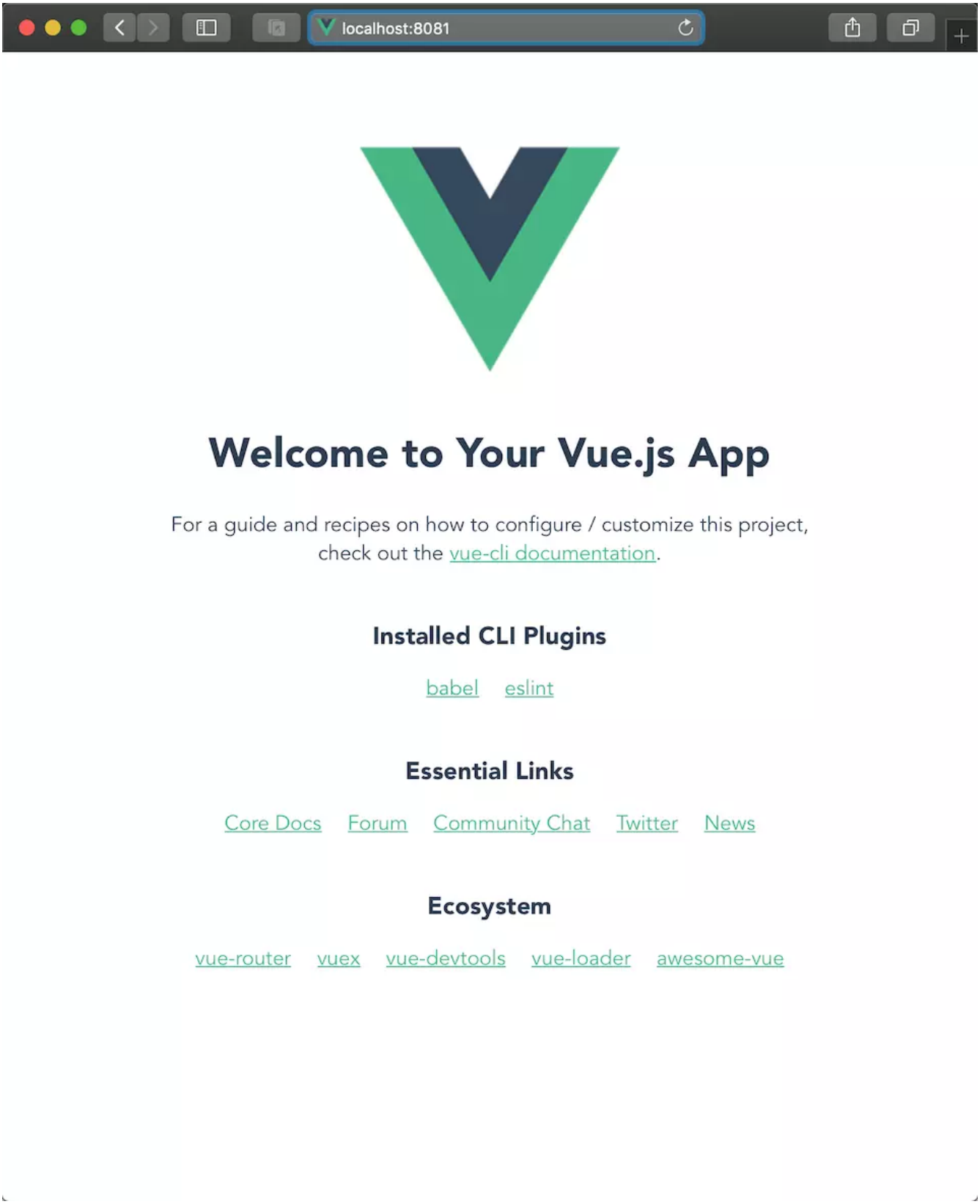
3.1 vue cli 创建一个vue项目



运行命令

```
yarn serve / npm run serve
```

访问 [http://localhost:8081



3.2 改写

稍微改写一下页面，在App.vue中 传入HelloWorld 组件中的 msg 改为Hello Docker ; created 生命周期中加入一个接口请求

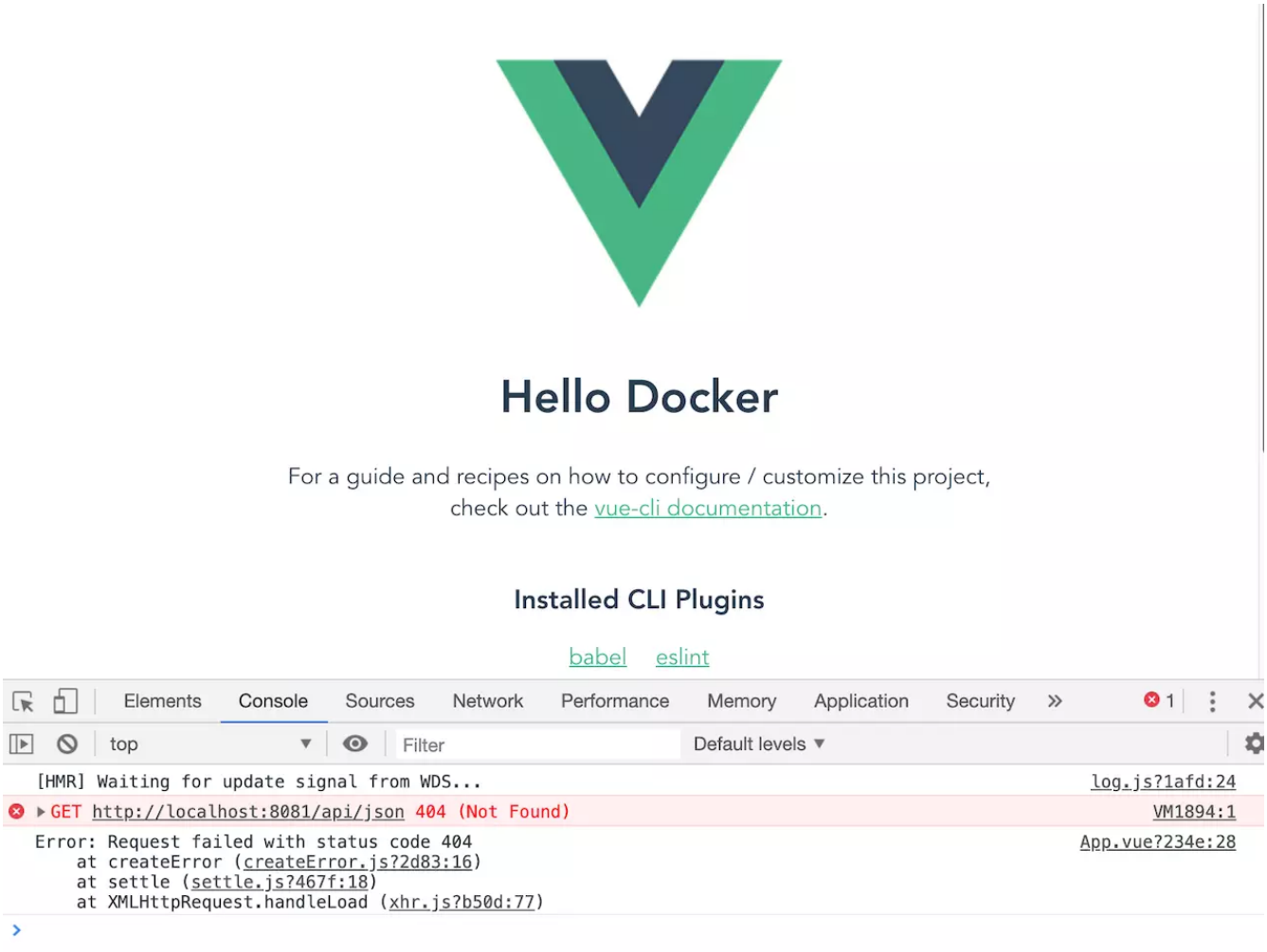
```
import axios from 'axios';
```

```
……

axios.get('/api/json', {
  params: {}
}).then(
  res => {
    console.log(res);
  }
).catch(
  error => {
      console.log(error);
  }
)

……
```

这时候会在页面控制台看到一个报错信息:



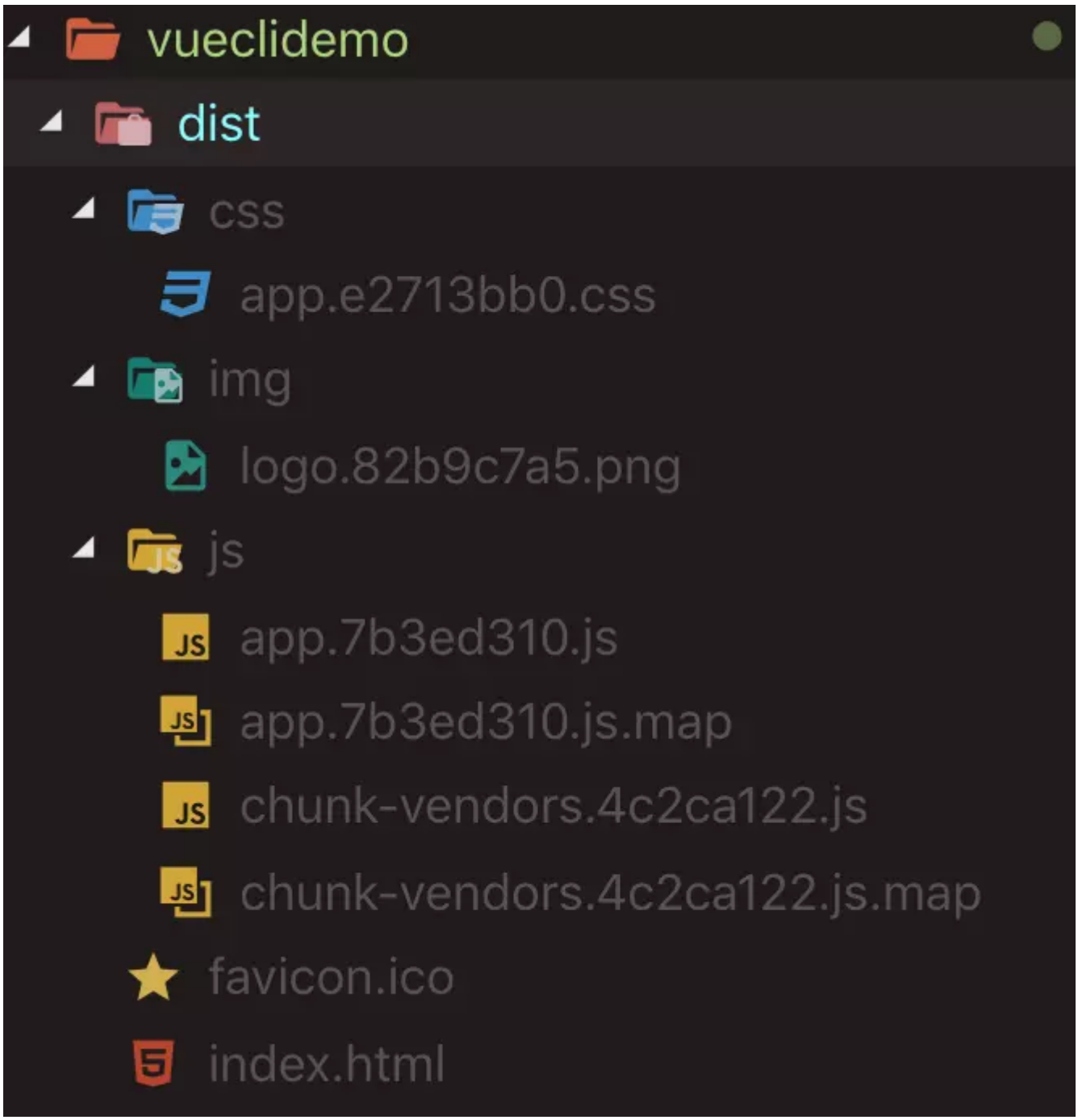
/api/json 接口 404，当然此时这个接口还不存在，暂时写到这里，一会再调这个接口。

3.3 构建vue项目

运行命令

```
yarn build / npm run build
```

此时工程根目录下多出一个`dist`文件夹



> 如果将该dist目录整个传到服务器上，部署成静态资源站点就能直接访问到该项目。

接下来就来构建一个这样的静态资源站点。

4 构建vue应用镜像

> nginx 是一个高性能的HTTP和反向代理服务器，此处我们选用 nginx 镜像作为基础来构建我们的vue应用镜像。

4.1 获取 nginx 镜像

```
docker pull nginx
```

> - `docker`镜像（Image）一个特殊的文件系统。Docker镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。 镜像不包含任何动态数据，其内容在构建之后也不会被改变。
> - docker 镜像相关操作有: 搜索镜像`docker search [REPOSITORY[:TAG]]`、拉取镜像`docker pull [REPOSITORY[:TAG]]`、查看镜像列表`docker image ls`、删除镜像: `docker image rm [REPOSITORY[:TAG]] / docker rmi [REPOSITORY[:TAG]]`等等。
> - docker 镜像名称由REPOSITORY和TAG组成`[REPOSITORY[:TAG]]`，TAG默认为latest

4.2 创建 nginx config配置文件

在项目根目录下创建`nginx`文件夹，该文件夹下新建文件`default.conf

```
server {
    listen       80;
    server_name  localhost;

    #charset koi8-r;
    access_log  /var/log/nginx/host.access.log  main;
    error_log  /var/log/nginx/error.log  error;

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

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

该配置文件定义了首页的指向为`/usr/share/nginx/html/index.html`,所以我们可以一会把构建出来的index.html文件和相关的静态资源放到`/usr/share/nginx/html`目录下。

4.3 创建 Dockerfile 文件

```
FROM nginx
COPY dist/ /usr/share/nginx/html/
COPY nginx/default.conf /etc/nginx/conf.d/default.conf
```

> - 自定义构建镜像的时候基于Dockerfile来构建。
> - `FROM nginx`命令的意思该镜像是基于 nginx:latest 镜像而构建的。
> - `COPY dist/ /usr/share/nginx/html/`命令的意思是将项目根目录下dist文件夹下的所有文件复制到镜像中 /usr/share/nginx/html/ 目录下。

> - `COPY nginx/default.conf /etc/nginx/conf.d/default.conf` 命令的意思是将nginx目录下的default.conf 复制到 etc/nginx/conf.d/default.conf，用本地的 default.conf 配置来替换nginx镜像里的默认配置。

4.4 基于该Dockerfile构建vue应用镜像

运行命令（注意不要少了最后的“.”）

```
docker build -t vuenginxcontainer .
```

> `-t` 是给镜像命名 `.` 是基于当前目录的Dockerfile来构建镜像

查看本地镜像，运行命令

```
docker image ls | grep vuenginxcontainer
```

```
$ docker image ls | grep vuenginxcontainer
vuenginxcontainer      latest              55bb1307b677        14 minutes ago      110MB
```

到此时我们的 `vue` 应用镜像 vuenginxcontainer 已经成功创建。接下来，我们基于该镜像启动一个`docker`容器。

4.5 启动 vue app 容器

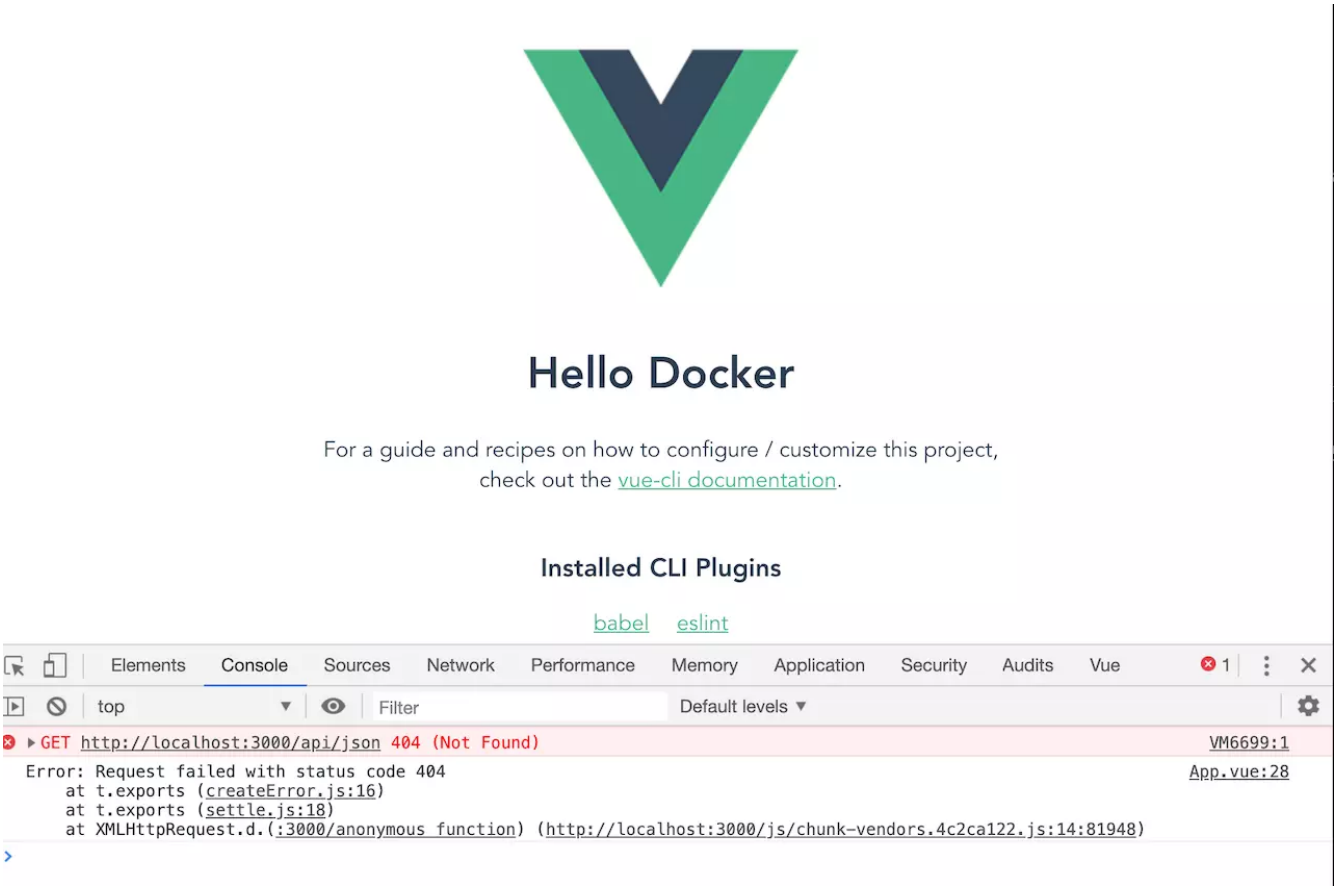
> Docker 容器Container： 镜像运行时的实体。镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等 。

基于 vuenginxcontainer 镜像启动容器，运行命令：

```
docker run \
-p 3000:80 \
-d --name vueApp \
vuenginxcontainer
```

> - `docker run` 基于镜像启动一个容器
> - `-p 3000:80` 端口映射，将宿主的3000端口映射到容器的80端口
> - `-d` 后台方式运行
> - `--name` 容器名 查看 docker 进程

可以发现名为 vueApp的容器已经运行起来。此时访问 http://localhost:3000 应该就能访问到该vue应用:



目前为止，已经通过`docker`容器部署了一个静态资源服务，可以访问到静态资源文件。还有 /api/json这个接口数据没有，接下来我们来解决一下这个问题。

5 接口服务

> 再部署一个 node 的容器来提供接口服务

5.1 express 服务

用 node web 框架 `express` 来写一个服务，注册一个返回json数据格式的路由 server.js:

```
'use strict';

const express = require('express');

const PORT = 8080;
const HOST = '0.0.0.0';

const app = express();
app.get('/', (req, res) => {
    res.send('Hello world\n');
});

app.get('/json', (req, res) => {
    res.json({
        code: 0,
        data :'This is message from node container'
    })
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

运行该 `express` 应用需要 `node` 环境，我们基于 `node` 镜像来构建一个新镜像

5.2 获取 `node` 镜像

```
docker pull node
```

5.3 编写 Dockerfile 将 `express` 应用 `docker` 化

```
FROM node

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 8080
CMD [ "npm", "start" ]
```

构建镜像的时候 node_modules 的依赖直接通过 `RUN npm install` 来安装，项目中创建一个 `.dockerignore` 文件来忽略一些直接跳过的文件：

```
node_modules
npm-debug.log
```

5.4 构建 nodewebserver 镜像

运行构建命令：

```
docker build -t nodewebserver .
```

5.5 启动 nodeserver 容器

> 基于刚刚构建的 nodewebserver 镜像 启动一个名为 nodeserver 的容器来提供接口服务8080端口，并映射宿主的5000端口

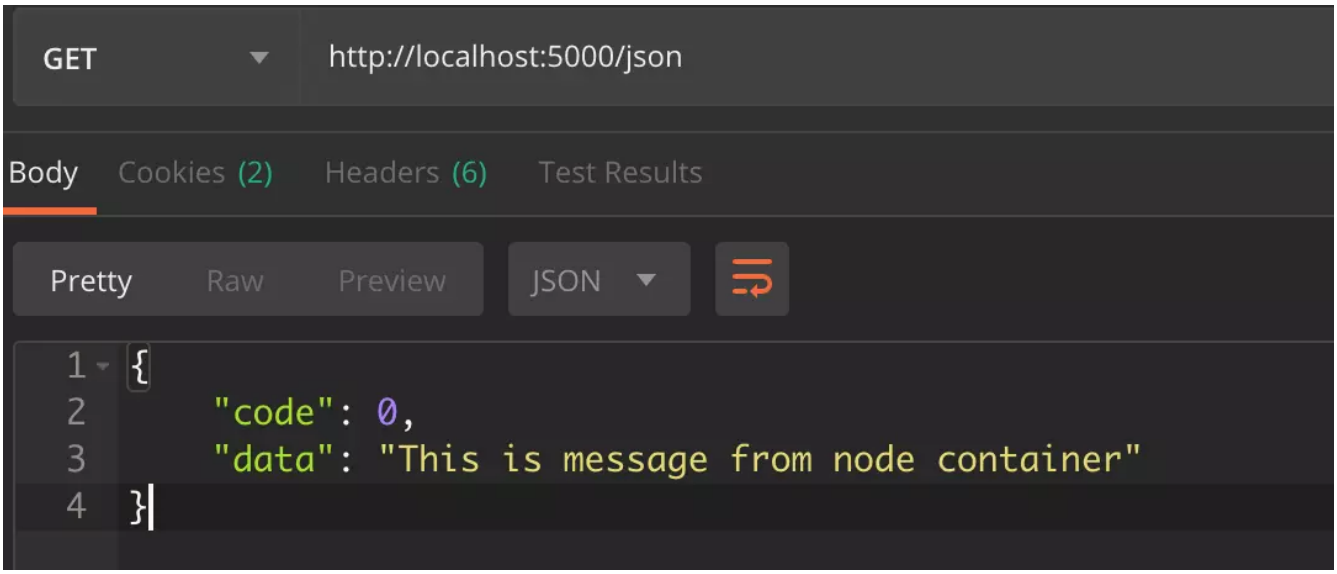
```
docker run \
-p 5000:8080 \
-d --name nodeserver \
nodewebserver
```

查看当前docker进程

```
docker ps
```

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
02277acc3efc        nodewebserver       "npm start"              3 seconds ago       Up 2 seconds        0.0.0.0:5000->8080/tcp   nodeserver
5cfe5fa8ab6a        vuenginxcontainer   "nginx -g 'daemon of…"   About an hour ago   Up About an hour    0.0.0.0:3000->80/tcp     vueApp
```

可以发现 nodeserver 的容器也正常的运行起来。访问以下 http://localhost:5000/json 能访问到前面写的json数据



到目前为止，后端接口服务也正常启动了。只需最后把页面请求的接口转发到后端接口服务就能调通接口。

6. 跨域转发

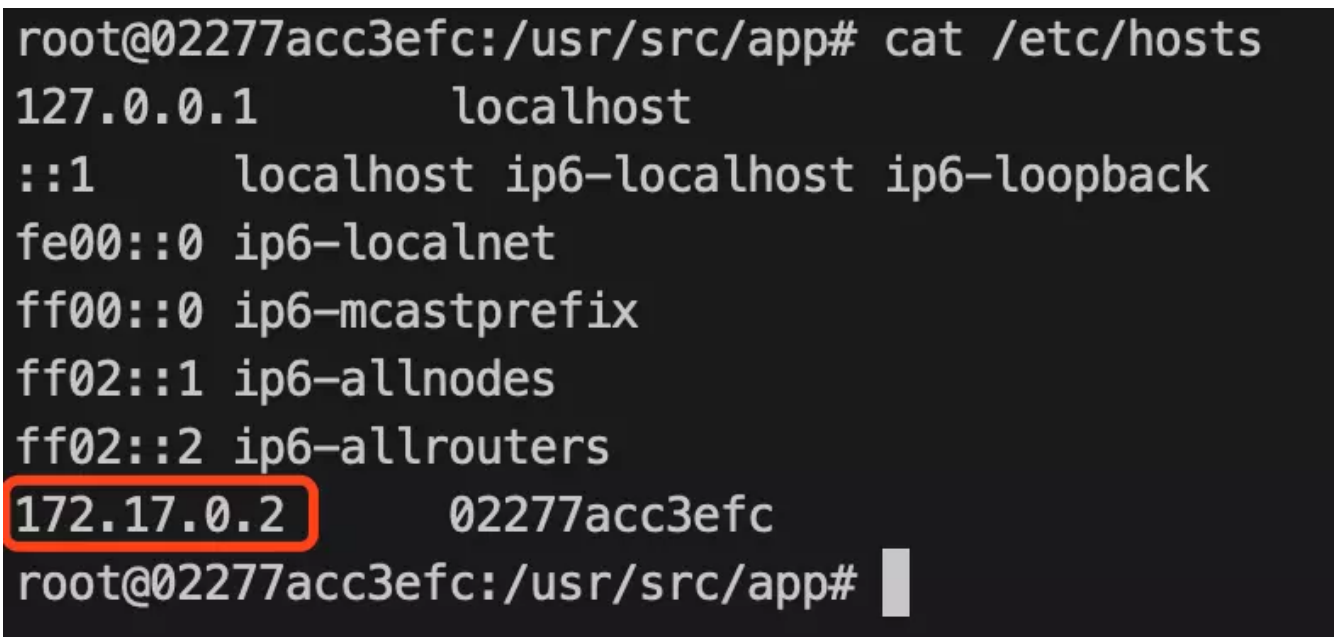
想要将 vueApp 容器 上的请求转发到 nodeserver 容器上。首先需要知道 nodeserver 容器的`ip`地址和端口，目前已知 nodeserver 容器内部服务监听在 8080 端口，还需要知道`ip`即可。

6.1 查看 nodeserver 容器的 ip 地址：

查看容器内部`ip`有多种方式，这里提供两种：

- 进入容器内部查看

```
docker exect -it 02277acc3efc bash
cat /etc/hosts
```



- docker inspect [containerId] 直接查看容器信息:

```
docker inspect 02277acc3efc
```

在其中找到 Networks 相关配置信息：

```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "d576be31109fb638720dfae0d0abb1f4ab98f3e6a5dd6ccab4971c0405741c1f",
        "EndpointID": "6d347f65ea9f9d5ee4c15e374cfe22fe89b2e3675deccf0a2b579fbe3320fb82",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.2"
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:02",
        "DriverOpts": null
    }
}
```

记录下node服务容器对应的ip，一会儿配置nginx转发的时候会用到。

6.2 修改 nginx 配置

- Nginx 配置 location 指向 node 服务 default.conf （前端想要了解的Nginx，关于Nginx的配置已经 location 的具体写法可以参考（一文弄懂Nginx的location匹配））
- 添加一条重写规则，将 /api/{path} 转到目标服务的 /{path} 接口上。 在前面的nginx/default.conf文件中加入：

```
location /api/ {
  rewrite  /api/(.*)  /$1  break;
  proxy_pass http://172.17.0.2:8080;
}
```

修改完了之后意识到一个问题：vueApp 容器是基于 vuenginxcontainer 这个镜像运行的，而在一开始构建镜像的时候是将 nginx配置 default.conf 直接构建进去了。因此如果需要修改 default.conf 还得再重新构建一个新的镜像，再基于新镜像来运行新的容器。

7. 改进

能不能每次修改配置文件后直接重启容器就能让新配置生效，答案当然是有。

> 在构建镜像的时候 不把 Nginx 配置复制到镜像中，而是直接挂载到宿主机上，每次修改配置后，直接重启容器即可。

7.1 修改 Dockerfile 文件

把 vueclidemo 项目下的 Dockerfile 修改一下

```
FROM nginx
COPY dist/  /usr/share/nginx/html/
COPY nginx/default.conf /etc/nginx/conf.d/default.conf
```

将`COPY nginx/default.conf /etc/nginx/conf.d/default.conf`命令删除，nginx配置都通过挂载命令挂载在宿主机上。再看`COPY dist/ /usr/share/nginx/html/`命令，如果每次构建的项目dist/下的内容变动都需要重新走一遍构建新镜像再启动新容器的操作，因此这条命令也可以删除，使用挂载的方式来启动容器。

7.2 重新运行vue应用容器

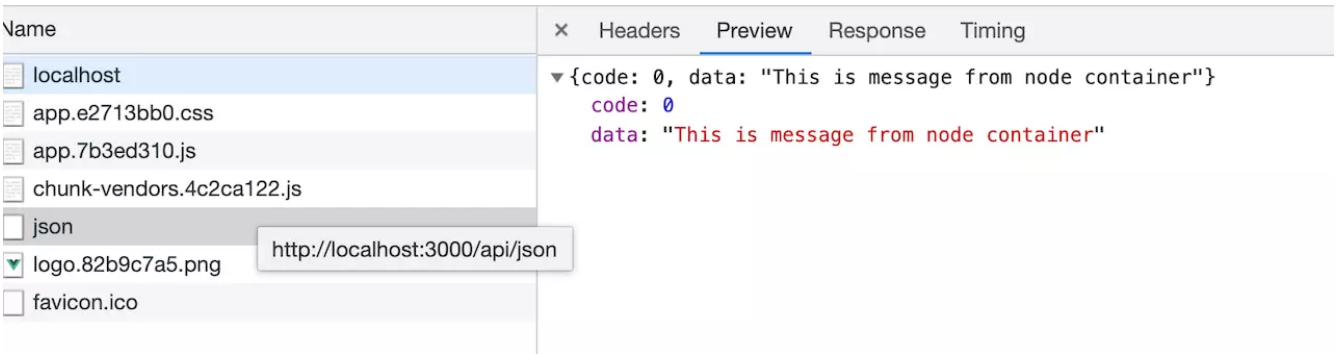
直接基于nginx镜像来启动容器 vuenginxnew ， 运行命令：

```
docker run \
-p 3000:80 \
-d --name vuenginxnew \
--mount type=bind,source=$HOME/SelfWork/docker/vueclidemo/nginx,target=/etc/nginx/conf.d \
--mount type=bind,source=$HOME/SelfWork/docker/vueclidemo/dist,target=/usr/share/nginx/html
\
nginx
```

> - `--mount type=bind,source={sourceDir},target={targetDir}` 将宿主机的sourceDir 挂载到容器的targetDir 目录上。
> - 此处运行的命令较长，如果每次重新输入难免麻烦，我们可以将完整的命令保存到一个`shell`文件`vueapp.sh` 中，然后直接执行 `sh vueapp.sh`。

这样就能每次修改了nginx配置或者 重新构建了vue应用的时候，只需重启容器就能立马生效。

此时我们再访问 http://localhost:3000/api/json 能看到接口能正常返回，说明转发生效了。



至此接口服务的转发也调通了。

7.3 配置负载均衡

> 后端服务一般都是双机或者多机以确保服务的稳定性。我们可以再启动一个后端服务容器，并修改`nginx`的配置 来优化资源利用率，最大化吞吐量，减少延迟，确保容错配置。

基于前面 4.5 节的类似操作，新启动一个容器，并基于 5.1 节类似的操作，查看到 新容器的 IP (172.17.0.3)

修改一下 `nginx/default.conf` （新增 upstream ， 修改 location /api/ 中的 proxy_pass）：

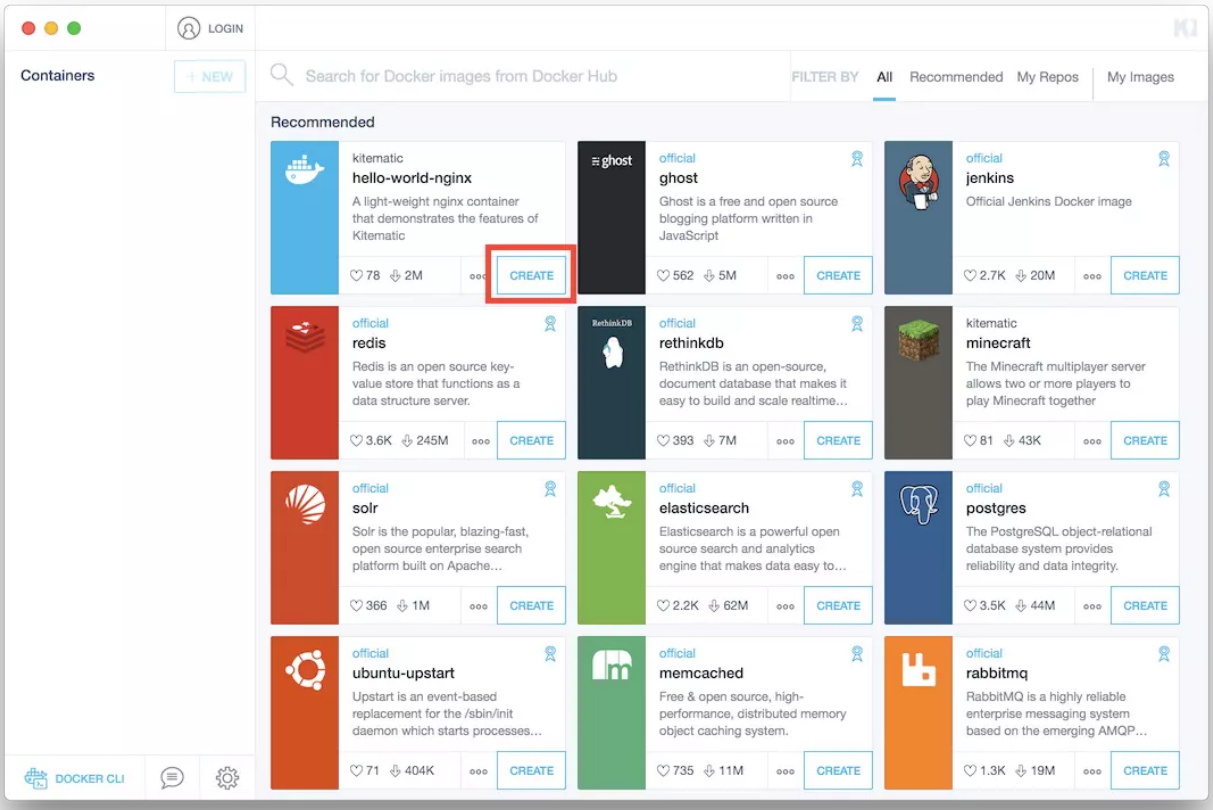
```
upstream backend {
    server 172.17.0.2:8080;
    server 172.17.0.3:8080;
}

......

location /api/ {
    rewrite  /api/(.*)  /$1  break;
    proxy_pass backend;
}
```

8. 写在后面

不习惯命令行的同学可以选用 Kitematic 来管理 `docker` 容器的状态、数据目录和网络。所有对容量的操作都可以可视化的操作，这里就不做过多介绍了，有兴趣的同学可以自行体验下。



9 总结

docker提供了非常强大的自动化部署方式与灵活性，对多个应用程序之间做到了解耦，提供了开发上的敏捷性、可控性以及可移植性。本文以vue项目为例实现一个前后分离项目使用 `docker` 部署的完整步骤，希望能给想要拥抱 docker 的同学带来一点帮助。

参考资源

docker 官网

nginx 官网

docker 从入门到实践

Kitematic user guide

前端想要了解的Nginx

一文弄懂Nginx的location匹配