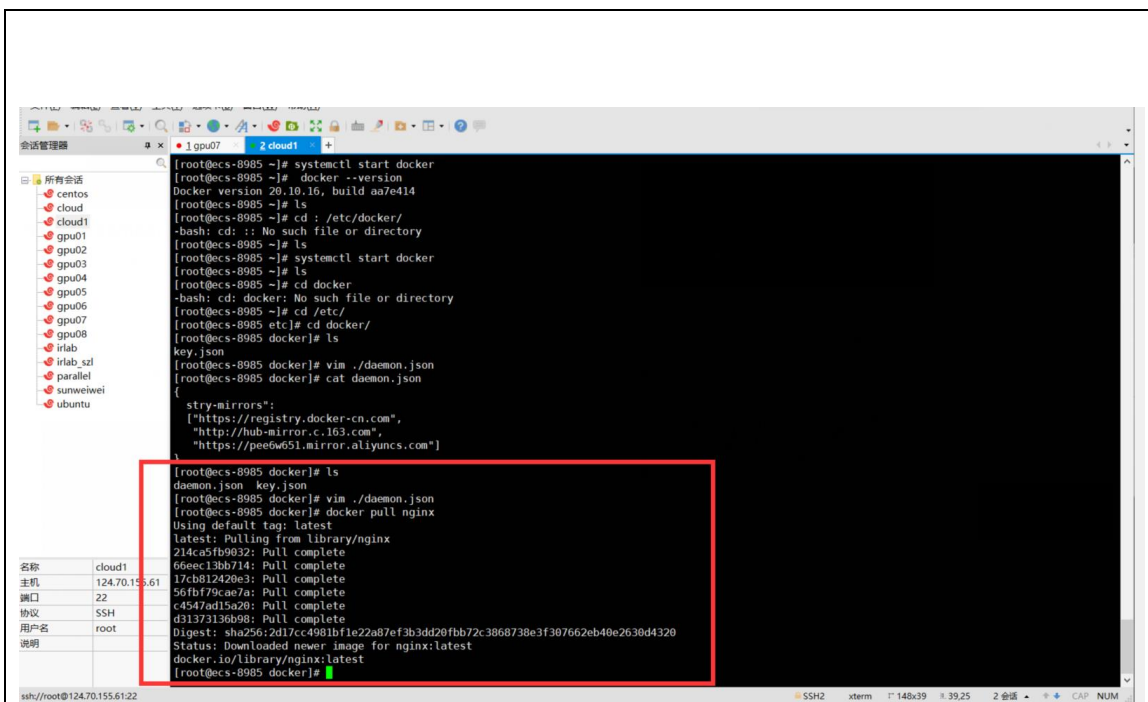
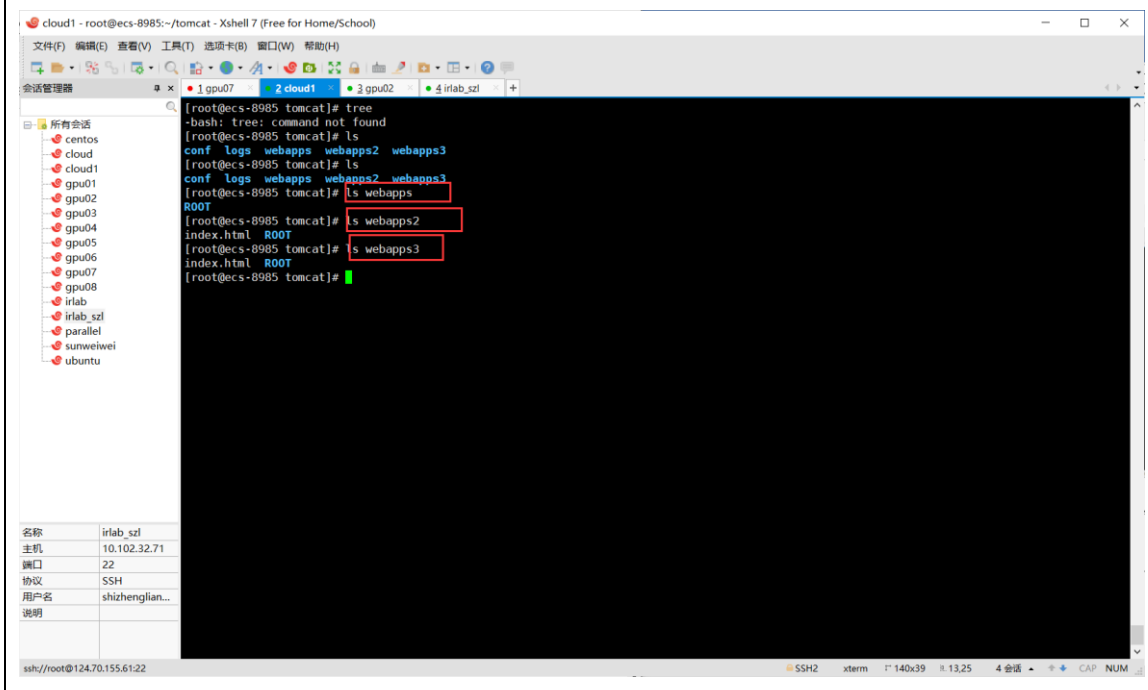


山东大学 计算机科学与技术 学院
云计算技术 课程实验报告

学号：201900130133	姓名：施政良	班级：四班
实验题目： 博客的撰写和发布		
实验学时： 2	实验日期： 2020-03-14	
实验目的： 在云端用 docker composed 的方式部署 Nginx 和 tomcat 实现负载均衡		
硬件环境： 联网计算机一台		
软件环境： Windows or Linux (其他工具或平台：华为云、XShell 、 Docker 、 Ngnix 、 Tomcat)		
实验步骤与内容： 实验步骤概述： <p>本次实验需要在云端用 docker composed 的方式部署 Nginx 和 tomcat 实现负载均衡，具体包括：</p> <ol style="list-style-type: none">1. 要求部署一台 Nginx 和三台 Tomcat 服务器2. Nginx 需要实现三种策略：<ol style="list-style-type: none">1) 轮询2) 权重，三台服务器的权重为 1，2，5；3) IP Hash。 <p>体现 docker composed 对 Docker 容器集群的快速编排</p> <p>详细实验步骤如下所示。</p> <p>与实验 11 相同，本次实验之前也需要配置 nginx 和 tomcat 环境，之后编写 docker-compose 所使用的编排文件，启动 tomcat 集群。最后测试不同的负载均衡策略。</p> <p>一、配置 nginx 和 tomcat 环境</p> <p>1.首先在终端中输入如下命令进行安装</p> <div style="border-left: 2px solid green; padding-left: 10px; margin-top: 10px;"><pre>1. docker pull nginx 2. docker pull tomcat</pre></div>		



2. 待下载完毕之后，在本地文件目录中创建相应的文件夹，将软件挂载到 docker 上。为了便于区分和后续的实验，本次实验中创建的文件目录如下所示：



```

[root@ecs-8985 etc]# cd ~
[root@ecs-8985 ~]# ls
[root@ecs-8985 ~]# mkdir -p ~/nginx/www ~/nginx/conf ~/nginx/logs
[root@ecs-8985 ~]# ls
nginx
[root@ecs-8985 ~]# mkdir -p ~/tomcat/webapps/R00T ~/tomcat/conf ~/tomcat/logs
[root@ecs-8985 ~]# ls
nginx tomcat tomcat
[root@ecs-8985 ~]# rm tomact/
rm: cannot remove 'tomact/': Is a directory
[root@ecs-8985 ~]# rm -r tomact/
rm: descend into directory 'tomact/'? n
[root@ecs-8985 ~]# mkdir -p ~/tomcat/webapps/R00T ~/tomcat/conf
[root@ecs-8985 ~]# ls
nginx tomcat tomcat
[root@ecs-8985 ~]#

```

3.编写 html 文件：为了后续在服务器上显示网页内容，需要编写相应的 html 文件。内容如下所示：

```

1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5.     <meta charset="UTF-8">
6.     <title>docker deployment</title>
7. </head>
8.
9. <body>
10.    <h1>hello, world</h1>
11. </body>
12.
13. </html>

```

```

[root@ecs-8985 R00T]# cat index.html
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>docker deployment</title>
</head>
<body>
<h1>Hello world</h1>
</body>
</html>
[root@ecs-8985 R00T]#

```

由于本次试验中需要同时开启三个服务器，因此需要分别创建三个网页。为了便于区分，每一个网页上显示的内容不同。例如，另外两个网页的源码如下：

网页 2 代码：

```

1. <!DOCTYPE html>
2. <html>

```

```
3.
4. <head>
5.     <meta charset="UTF-8">
6.     <title>docker deployment</title>
7. </head>
8.
9. <body>
10.    <h1>Hello world in webapps2</h1>
11.</body>
12.
13.</html>
```

网页 3 代码:

```
1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5.     <meta charset="UTF-8">
6.     <title>docker deployment</title>
7. </head>
8.
9. <body>
10.    <h1>Hello world in webapps3</h1>
11.</body>
12.
13.</html>
```

分别将其创建在对应的文件目录下, 并使用 cat 命令查看

```
[root@ecs-8985 R00T]# cat ../../webapps3/R00T/index.html
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>docker deployment</title>
</head>
<body>
<h1>hello world in webapps3 </h1>
</body>
</html>
[root@ecs-8985 R00T]#
```

```
[root@ecs-8985 R00T]# cat ../../webapps2/R00T/index.html
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>docker deployment</title>
</head>
<body>
<h1>Hello world in webapps2</h1>
</body>
</html>
[root@ecs-8985 R00T]#
```

4. 启动容器：在上述试验基础上，启动三个 tomcat 容器，需要在终端输入如下命令

1. `docker run -d --name tomcat1 -v ~/tomcat/webapps:/usr/local/tomcat/webapps tomcat`
2. `docker run -d --name tomcat2 -v ~/tomcat/webapps2:/usr/local/tomcat/webapps tomcat`
3. `docker run -d --name tomcat3 -v ~/tomcat/webapps3:/usr/local/tomcat/webapps tomcat`

输入之后，终端显示内容如下图所示：

```
Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]  
  
Run a command in a new container:  
[root@ecs-8985 conf]# docker run -d -p 80:80 --name nginx -v ~/nginx/www:/usr/share/nginx/html -v ~/nginx/conf/nginx.conf:/etc/nginx/nginx.conf -v ~/nginx/logs:/var/log/nginx nginx  
3e94c012a489732c4ebc4057ef0588e95b46e0f5d8e423416b5ed81b1bd40cc0  
[root@ecs-8985 conf]# docker ps  


| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS |
|--------------|-------|---------|---------|--------|-------|
|--------------|-------|---------|---------|--------|-------|

  
[root@ecs-8985 ROOT]# ls  
index.html  
[root@ecs-8985 ROOT]# vim index.html  
[root@ecs-8985 ROOT]# docker run -d --name tomcat1 -v ~/tomcat/webapps:/usr/local/tomcat/webapps tomcat  
ebf6319066bbd0f54d2b6c0bc66fc92ca3c7ca6dc9dfdf6a242d3b5897a0b72  
[root@ecs-8985 ROOT]# docker run -d --name tomcat2 -v ~/tomcat/webapps:/usr/local/tomcat/webapps tomcat  
2950eb60435ea0bc9016ab1c186d3bec28d385a78b5be7304029471758a10ef0  
[root@ecs-8985 ROOT]#
```

此时，输入 `docker ps` 命令进行查看，可以发现如下信息

所有会话

centos
cloud
cloud1
gpu01
gpu02
gpu03
gpu04
gpu05
gpu06
gpu07
gpu08
irlab
irlab_szl
parallel
sunweiwei
ubuntu

名称

主机

端口

协议

用户名

说明

irlab_szl

10.102.32.71

22

SSH

shizhenglian...

```
<title>docker deployment</title>  
</head>  
<body>  
<h1>Hello world</h1>  
</body>  
</html>  
  
[root@ecs-8985 ROOT]# cat ../../webapps3/ROOT/index.html  
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
<title>docker deployment</title>  
</head>  
<body>  
<h1>hello world in webapps3</h1>  
</body>  
</html>  
  
[root@ecs-8985 ROOT]# cat ../../webapps2/ROOT/index.html  
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
<title>docker deployment</title>  
</head>  
<body>  
<h1>hello world in webapps2</h1>  
</body>  
</html>  
  
三个tomcat容器  
  
[root@ecs-8985 ROOT]# docker ps  


| CONTAINER ID | IMAGE  | COMMAND                 | CREATED     | STATUS     | PORTS                             | NAMES   |
|--------------|--------|-------------------------|-------------|------------|-----------------------------------|---------|
| 3e94c012a489 | nginx  | "/docker-entrypoint..." | 3 hours ago | Up 3 hours | 0.0.0.0:80->80/tcp, :::80->80/tcp | nginx   |
| a47f27a14e03 | tomcat | "catalina.sh run"       | 3 hours ago | Up 3 hours | 8080/tcp                          | tomcat3 |
| c7b365504692 | tomcat | "catalina.sh run"       | 3 hours ago | Up 3 hours | 8080/tcp                          | tomcat2 |
| 8ca24ceab837 | tomcat | "catalina.sh run"       | 3 hours ago | Up 3 hours | 8080/tcp                          | tomcat1 |

  
[root@ecs-8985 ROOT]#
```

5.配置相关文件：首先使用字符串查找命令（`grep`），获取 tomcat 容器 IP，获取到的 IP 将配置到 nginx 的配置文件中。

1. `docker inspect tomcat1|grep "IPAddress"`
2. `docker inspect tomcat2|grep "IPAddress"`

3. docker inspect tomcat3|grep "IPAddress"

终端输出如下

```
CONTAINER ID   IMAGE    COMMAND                  CREATED        STATUS        PORTS          NAMES
a47f27a14e03   tomcat   "catalina.sh run"        6 seconds ago Up 4 seconds  8080/tcp       tomcat3
c7b365504692   tomcat   "catalina.sh run"        14 seconds ago Up 13 seconds  8080/tcp       tomcat2
8ca24ceab837   tomcat   "catalina.sh run"        2 minutes ago Up 2 minutes   8080/tcp       tomcat1
[root@ecs-8985 R00T]# docker inspect tomcat1|grep "IPAddress"
      "SecondaryIPAddresses": null,
      "IPAddress": "172.17.0.2",
      "IPAddress": "172.17.0.2",
[root@ecs-8985 R00T]# docker inspect tomcat2|grep "IPAddress"
      "SecondaryIPAddresses": null,
      "IPAddress": "172.17.0.3",
      "IPAddress": "172.17.0.3",
[root@ecs-8985 R00T]# docker inspect tomcat2|grep "IPAddress"
> ^C
[root@ecs-8985 R00T]# docker inspect tomcat2|grep "IPAddress"
      "SecondaryIPAddresses": null,
      "IPAddress": "172.17.0.3",
      "IPAddress": "172.17.0.3",
[root@ecs-8985 R00T]#
```

二、编写 docker-compose 所使用的编排文件

在配置好 nginx 和 tomcat 环境的基础上需要编写 docker-compose 所使用的编排文件，启动 tomcat 集群，计划 tomcat 容器端口为 8081,8082,8083，分别映射到宿主机的 8080 端口，文件内容如下（本实验使用的文件名称为 cluster.yaml）。

具体文件内容如下所示：

```
1. version: '3'
2. services:
3.   tomcat1:
4.     image: tomcat:latest
5.     container_name: tomcat1
6.     volumes:
7.       - /opt/tomcat1:/usr/local/tomcat/webapps/
8.     restart: always
9.     ports:
10.      - 8081:8080
11.  tomcat2:
12.    image: tomcat:latest
13.    container_name: tomcat2
14.    volumes:
15.      - /opt/tomcat2:/usr/local/tomcat/webapps/
16.    restart: always
17.    ports:
18.      - 8082:8080
19.  tomcat3:
20.    image: tomcat:latest
21.    container_name: tomcat3
```

```
22.     volumes:
23.         - /opt/tomcat3:/usr/local/tomcat/webapps/
24.     restart: always
25.     ports:
26.         - 8083:8080
```

编写完成之后需要实现 NGINX 的负载均衡+反向代理的配置，需要在上一个实验的基础进行改进，添加以下内容：

```
1. server {
2.     listen      80;
3.     server_name 172.18.0.1;
4.
5.     location / {
6.         root    html;
7.         index   index.html index.htm;
8.         proxy_pass http://tomcat;
9.     }
10.
11. upstream tomcat{
12.     server 172.18.0.1:8081;
13.     server 172.18.0.1:8082;
14.     server 172.18.0.1:8083;
15. }
```

最终完整的 nginx.conf 文件内容如下：

```
1. worker_processes 1;
2. events {
3.     worker_connections 1024;
4. }
5. http {
6.     include mime.types;
7.     default_type application/octet-stream;
8.     sendfile        on;
9.     keepalive_timeout 65;
10.    server {
11.        listen      80;
12.        server_name 172.18.0.1;
13.        location / {
14.            root    html;
15.            index   index.html index.htm;
16.            proxy_pass http://tomcat;
17.        }
```

```
18.         error_page   500 502 503 504   /50x.html;  
19.         location = /50x.html {  
20.             root    html;  
21.         }  
22.     }  
23.     upstream tomcat{  
24.         server 172.18.0.1:8081;  
25.         server 172.18.0.1:8082;  
26.         server 172.18.0.1:8083;  
27.     }  
28. }
```

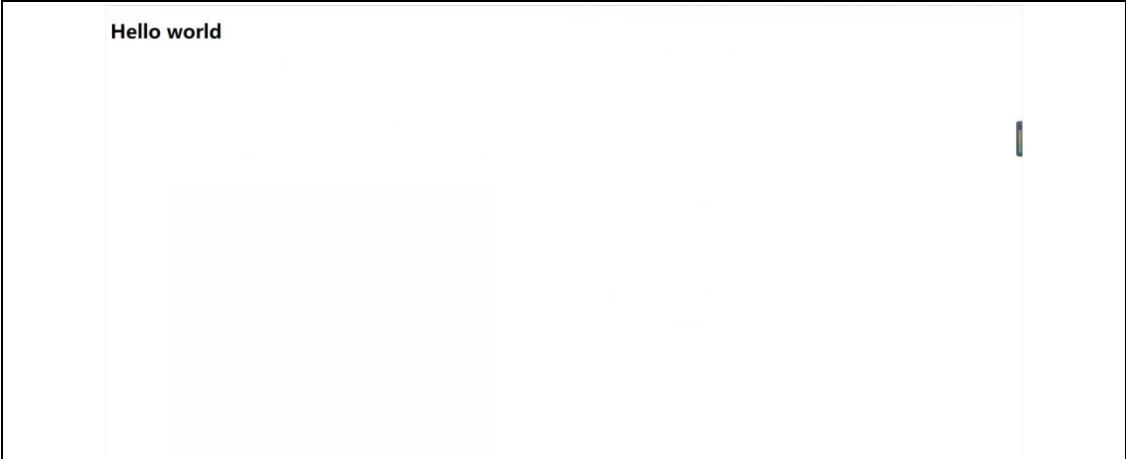
修改文件之后重启 nginx 服务，启动 tomcat 集群。具体命令为

nginx -s reload

同时利用 docker-compose 命令执行之前编写的.yaml 文件。

docker-compose -f cluster.yaml up -d

这个命令启动成功集群后，打开任意浏览器对服务器进行访问。可以显示如下页面

A screenshot of a web browser window. The browser has a light gray header bar. The main content area is white and displays the text "Hello world" in a bold, black, sans-serif font. The text is positioned in the upper left corner of the content area. The browser window has a thin gray border.

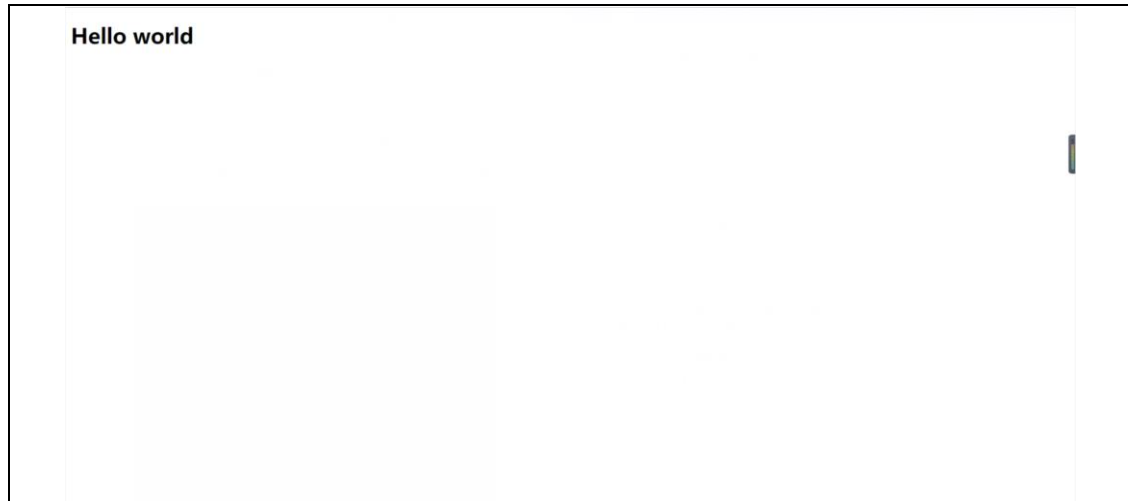
三、测试不同的负载均衡策略

1. 轮询法

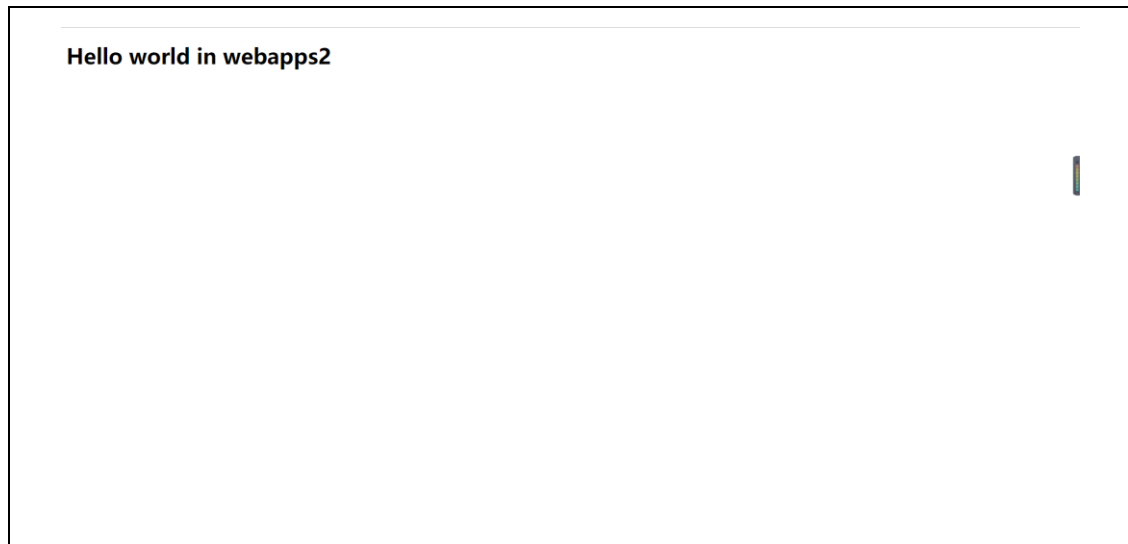
轮询法实现负载均衡即对于三个 ip 地址按照一定的次序轮番进行访问。在

nginx 的配置文件汇总默认即为轮训法，因此在上述实验的基础上可以直接进行测试。

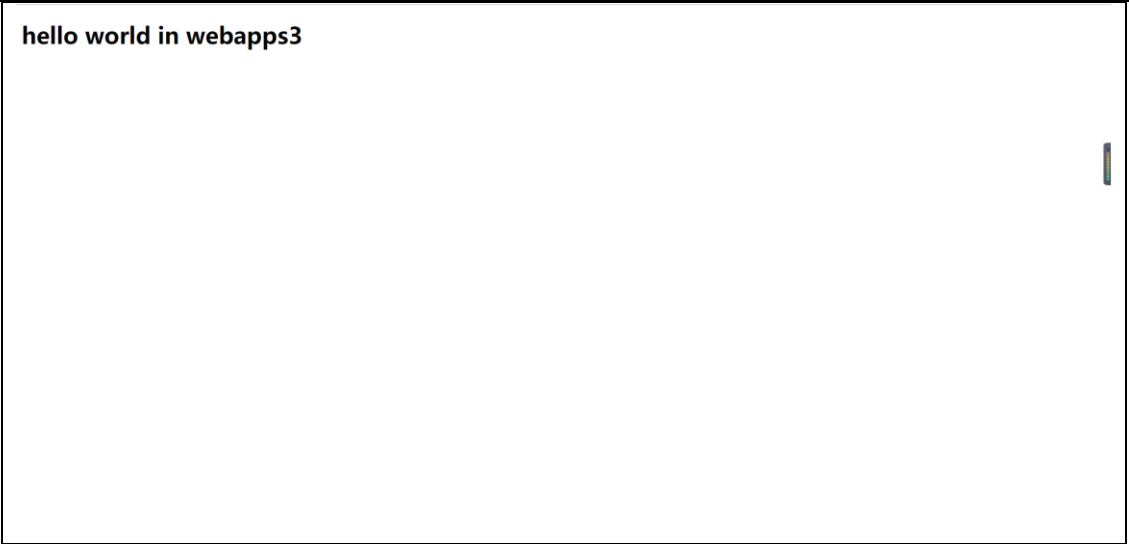
在测试时首先通过本地访问服务器的公网 ip，之后不断进行刷新以模拟不同次序的访问。通过具体实验可以看到，由于采用了轮询法，三个 tomcat 容器按照 1->2->3->1...的顺序依次被访问。可以通过观察网页中 html 的显示内容进行验证，如下图所示：



图一：容器一对应的 html 页面



图二：容器二对应的 html 页面

A screenshot of a web browser window. The address bar is empty. The main content area displays the text "hello world in webapps3" in a black, sans-serif font. The browser's interface, including the address bar and some UI elements, is visible at the top and right edges of the window.

hello world in webapps3

图三：容器三对应的 html 页面

2. 指定权重

指定权重的负载均衡策略通过为每一个容器分配不同的权重使每一个容器承担的流量不同，且权重越大，负载越大。

按照实验指导书说明,设定三台服务器对应的权重信息为 1 , 2 ,5 对 yaml 文件中添加如下的 weight 说明

```
1.    upstream tomcat
2.        server 172.18.0.1:8080 weight=1;
3.        server 172.18.0.1:8080 weight=2;
4.        server 172.18.0.1:8080 weight=5;
5.    }
```

相对于之前的代码仅仅修改了权重的分配。之后在本地不不断刷新浏览器模拟对服务器的多次访问。

可以发现，与轮询法不同，此时三个容器被不等概率的访问，且由于第三个容器的权重最大因此被访问的频率最大。

三、哈希实现负载均衡

按照实验指导书说明,需要测试哈希法实现负载均衡,只需修改配置文件的 upstream 字段即可。修改后内容如下:

```
1. 1. upstream tomcat {  
2. 2.     ip_hash;  
3. 3.     server 172.18.0.1:8080;  
4. 4.     server 172.18.0.1:8080;  
5. 5.     server 172.18.0.1:8080;  
6. 6. }
```

修改之后,使用如下命令重启服务器

docker restart nginx

之后在本地不不断刷新浏览器模拟对服务器的多次访问。与之前不同,由于使用了哈希对 ip 地址进行了哈希映射,因此不论如何刷新页面访问的均为 server 172.17.0.1:8080 对应的页面,与实验指导书的结果一致。

进一步的,可以打印日志文件进行查看,如下图所示。可以发现,在刷新页面的过程中,文件中记录的均为 172.17.0.1:8080 对应的容器,说明实验结果正确。

```
(root@ecs-69d5: /logs) # cat access.log  
223.99.13.199 - - [23/May/2022:09:34:08 +0000]  
"GET / HTTP/1.1" 200 141 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36 Edg/101.0.1210.53" "-"  
"172.17.0.2:8080"  
223.99.13.199 - - [23/May/2022:09:34:09 +0000]  
"GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36 Edg/101.0.1210.53" "-"  
"172.17.0.2:8080"  
223.99.13.199 - - [23/May/2022:09:34:09 +0000]  
"GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36 Edg/101.0.1210.53" "-"  
"172.17.0.2:8080"  
223.99.13.199 - - [23/May/2022:09:34:10 +0000]  
"GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36 Edg/101.0.1210.53" "-"  
"172.17.0.2:8080"  
223.99.13.199 - - [23/May/2022:09:34:10 +0000]  
"GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36 Edg/101.0.1210.53" "-"  
"172.17.0.2:8080"  
223.99.13.199 - - [23/May/2022:09:34:10 +0000]  
"GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36 Edg/101.0.1210.53" "-"  
"172.17.0.2:8080"  
223.99.13.199 - - [23/May/2022:09:34:10 +0000]  
"GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.64 Safari/537.36 Edg/101.0.1210.53" "-"  
"172.17.0.2:8080"
```

结论分析与体会：

一、结论分析

1. 对 docker-compose 的理解

分析：

docker-compose 是基于 docker 的编排工具，使容器的操作能够批量的，可视的执行，是一个管理多个容器的工具，比如可以解决容器之间的依赖关系，当在宿主机启动较多的容器时候，

如果都是手动操作会觉得比较麻烦而且容器出错，这个时候推荐使用 dockerd 的单机编排工具 docker-compose。

docker-compose 是基于 docker 的开源项目，托管于 github 上，由 python 实现，调用 docker 服务的 API 负责实现对 docker 容器集群的快速编排，即通过一个单独的 yaml 文件，来定义一组相关的容器来为一个项目服务。因此，docker-compose 默认的管理对象是项目，通过子命令的方式对项目中的一组容器进行生命周期的管理。

2. Docker 和 docker-compose 的对比

分析：二者的对比如下所示：

- 1.docker 是自动化构建镜像，并启动镜像。 docker compose 是自动化编排容器。
2. docker 是基于 Dockerfile 得到 images,启动的时候是一个单独的 container
3. docker-compose 是基于 docker-compose.yml,通常启动的时候是一个服务，这个服务通常由多个 container 共同组成，并且端口，配置等由 docker-compose 定义好。
- 4.两者都需要安装，但是要使用 docker-compose，必须已经安装 docker

3. 对比不同的负载均衡策略

分析：

本次实验中基于 docker-compose 测试了三种负载均衡策略的效果，这三种策略的对比如下所示。

1. **轮询（round-robin）**：轮询为负载均衡中较为基础也较为简单的算法，它不需要配置额外参数。假设配置文件中共有 台服务器，该算法遍历服务器节点列表，并按节点次序每轮选择一台服务器处理请求。当所有节点均被调用过一次后，该算法将从第一个节点开始重新新一轮遍历。

特点：由于该算法中每个请求按时间顺序逐一分配到不同的服务器处理，因此适用于服务器性能相近的集群情况，其中每个服务器承载相同的负载。但对于服务器性能不同的集群而言，该算法容易引发资源分配不合理等问题。

2、加权轮询：为了避免普通轮询带来的弊端，加权轮询应运而生。在加权轮询中，每个服务器会有各自的 `weight`。一般情况下，`weight` 的值越大意味着该服务器的性能越好，可以承载更多的请求。该算法中，客户端的请求按权值比例分配，当一个请求到达时，优先为其分配权值最大的服务器。

特点：加权轮询可以应用于服务器性能不等的集群中，使资源分配更加合理化。

3、哈希映射：`ip_hash` 依据发出请求的客户端 IP 的 `hash` 值来分配服务器，该算法可以保证同 IP 发出的请求映射到同一服务器，或者具有相同 `hash` 值的不同 IP 映射到同一服务器。

二、体会：

在本次实验中我了解了 `docker-compose` 的基本知识，`docker-compose` 是基于 `docker` 的编排工具，使容器的操作能够批量的，可视的执行，是一个管理多个容器的工具。并且在使用 `docker-compose` 之前必须确保已经安装了 `docker`。

与之前的实验 11 相同，本次实验童谣需要测试不同负载均衡策略的效果。通过实验测试可知，轮训算法会轮流对服务器进行访问，每一个容器承担的流量相同；加权轮询法则是提供了更大的灵活性，权重也大访问的频率越大；而 `ip hash` 算法则会锁定唯一的容器 `ip` 地址进行访问。

作为实验的总结，本次实验将课堂上的理论知识同实际的配置过程相结合，虽然配置过程略微枯燥，但是通过实际的操作，加深了我对知识的掌握和理解。