

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： 实现在云服务器上的负载均衡		
实验学时：6	实验日期：2020-05-14	
<p>实验目的：在云端用 docker 部署 nginx 和 tomcat 实现负载均衡</p> <p>具体包括：</p> <ol style="list-style-type: none"><li>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：1. 轮询。</li><li>2. 权重，三台服务器的权重为 1，3，5；</li><li>3. 测试 IP Hash。（最终实现的效果是，本地电脑通过网页访问云服务器时，网页能够体现 Nginx 三种策略的结果）</li></ol>		
<p>硬件环境：</p> <p>联网计算机一台</p>		
<p>软件环境：</p> <p>Windows or Linux</p> <p>（其他工具或平台：华为云、XShell 、Docker 、Nginx 、Tomcat）</p>		
<p>实验步骤与内容：</p> <p>实验步骤概述：</p> <p>本次实验要求部署一台 Nginx 和三台 Tomcat 服务器，并利用 Nginx 实现三种策略。实验步骤可以分为：</p> <ol style="list-style-type: none"><li>1. 华为云的购买</li><li>2. Docker 环境的配置</li><li>3. 相关软件环境的配置</li><li>4. 更改配置文件测试不同的实现策略</li></ol> <p>详细实验步骤如下所示。</p>		

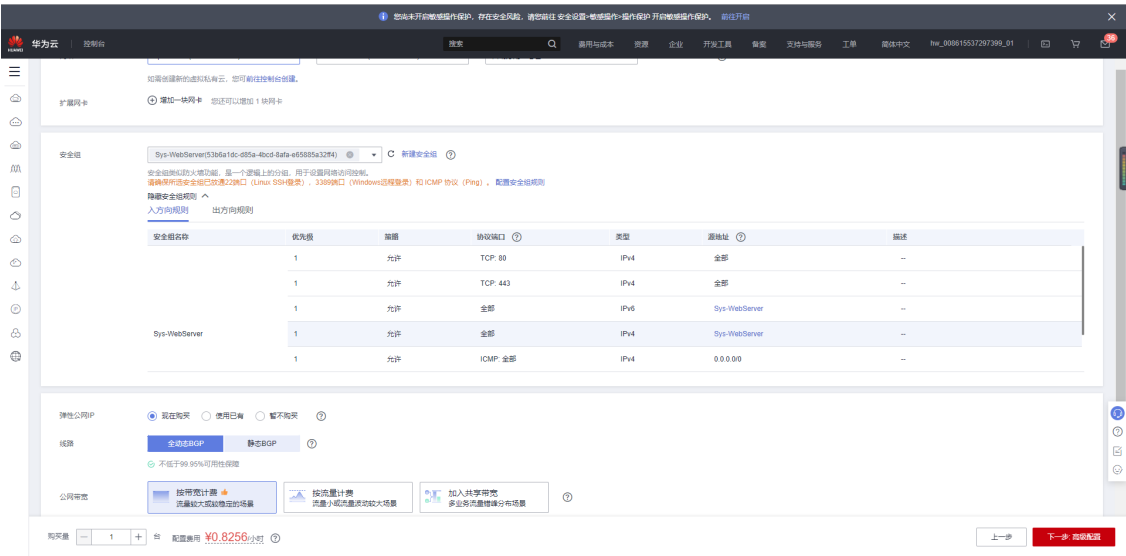
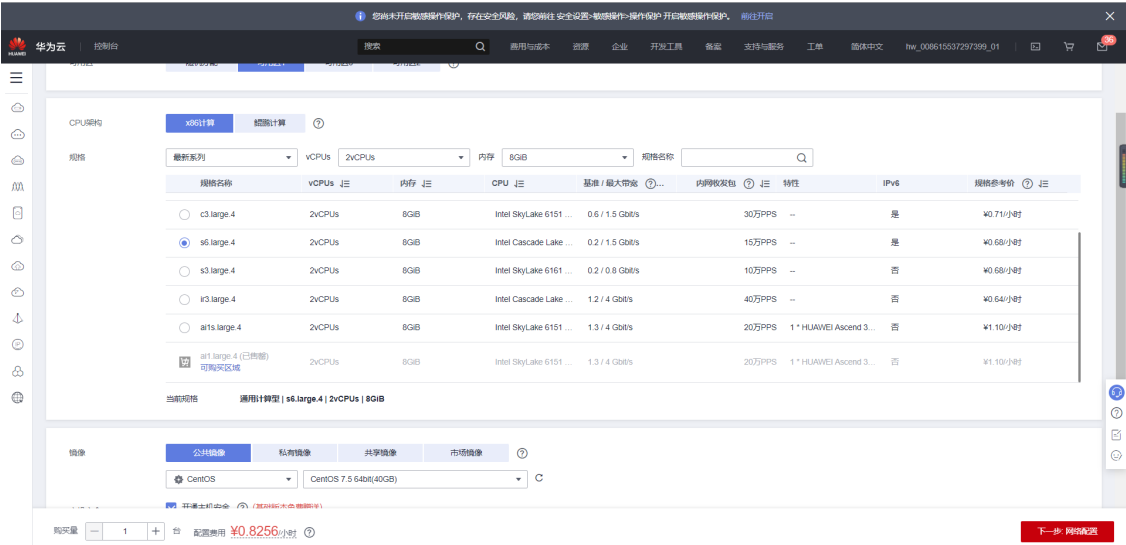
具体实验内容

1. 华为云的购买

在本次实验，需要首先购买华为云作为实验的平台。购买的具体配置如下所示：

2vCPUs | 8GiB | s6.large.4 CentOS 7.5 64bit

详细过程如下图所示





购买完成后可以返回主界面进行查看，可以发现有如下的华为云实例。



## 二、docker 环境的配置

在本实验中使用 xshell 连接云服务器，并且采用 xftp 进行文件的传输。

### 1. 添加 yml 环境

首先通过 xshell 连接云服务器，之后在终端中输入如下命令，添加 yml 源。

```
yum install epel-release -y
yum clean all
```

实验过程如下所示：

```
WARNING! The remote SSH server rejected X11 forwarding request.

Welcome to Huawei Cloud Service

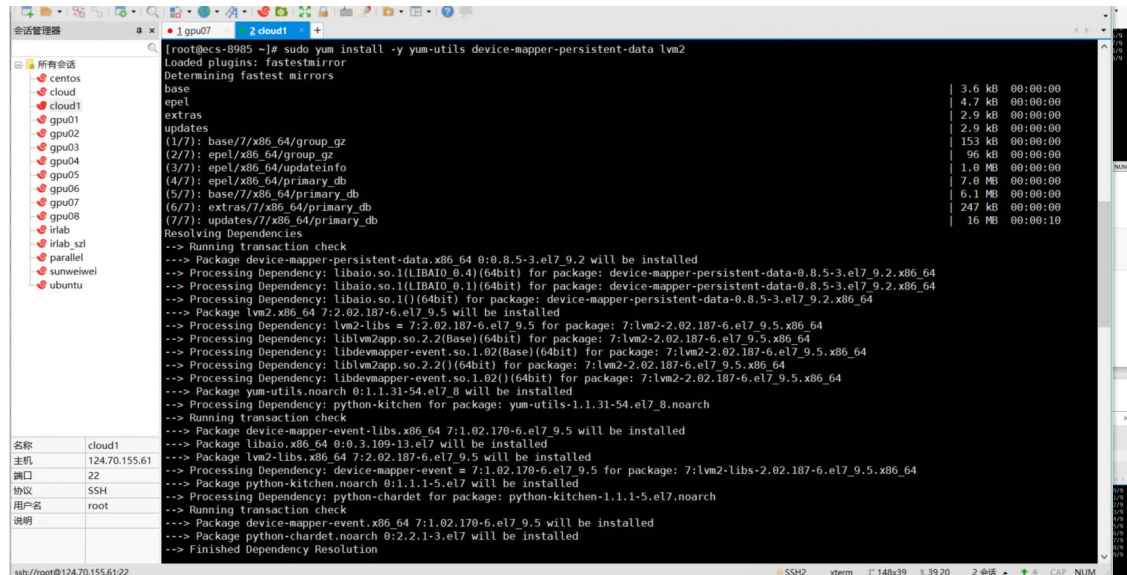
[root@ecs-8985 ~]# yum install epel-release -y
Loaded plugins: fastestmirror
Determining fastest mirrors
base                                     | 3.6 kB | 00:00:00
epel                                   | 4.7 kB | 00:00:00
extras                                | 2.9 kB | 00:00:00
updates                               | 2.9 kB | 00:00:00
(1/7): base/7/x86_64/group_gz         | 153 kB | 00:00:00
(2/7): epel/x86_64/group_gz          | 96 kB | 00:00:00
(3/7): epel/x86_64/updateinfo         | 1.0 MB | 00:00:00
(4/7): base/7/x86_64/primary_db      | 6.1 MB | 00:00:00
(5/7): epel/x86_64/primary_db        | 7.0 MB | 00:00:00
(6/7): extras/7/x86_64/primary_db    | 247 kB | 00:00:00
(7/7): updates/7/x86_64/primary_db    | 16 MB | 00:00:00
Package epel-release-7-14.noarch already installed and latest version
Nothing to do
[root@ecs-8985 ~]# yum clean all
Loaded plugins: fastestmirror
Cleaning repos: base epel extras updates
Cleaning up list of fastest mirrors
[root@ecs-8985 ~]#
```

## 2. 安装 uum-util:

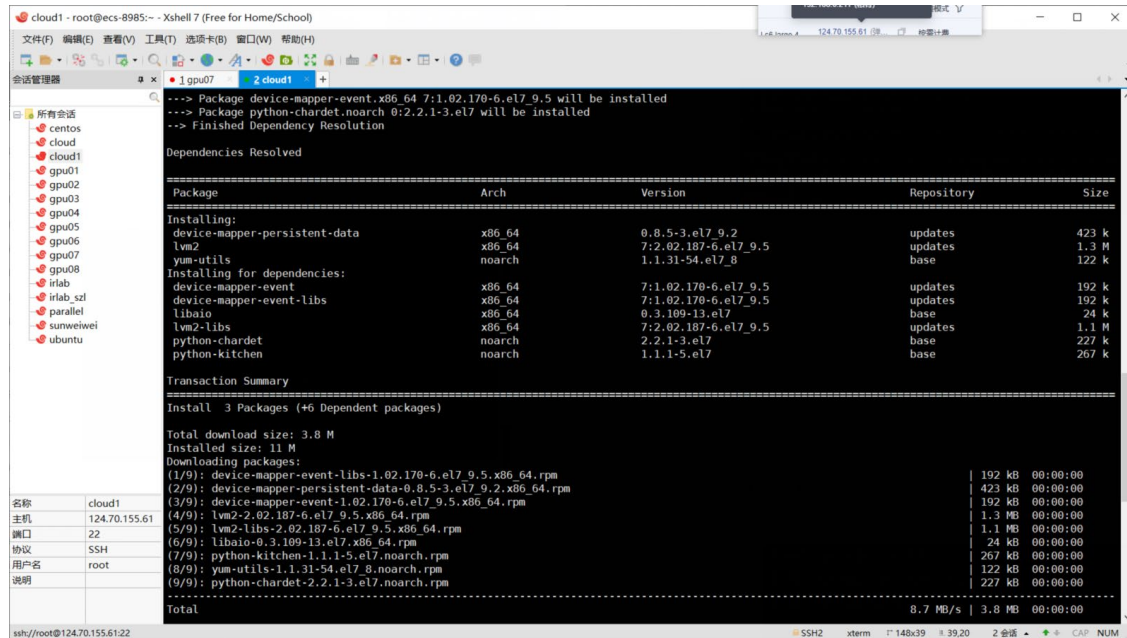
在终端中输入如下命令

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

## 3. 设置 docker: 为 docker 添加 yml 源



```
[root@ecs-8985 ~]# sudo yum install -y yum-utils device-mapper-persistent-data lvm2
Loaded plugins: fastestmirror
Determining fastest mirrors
base                                     | 3.6 kB | 00:00:00
epel                                     | 4.7 kB | 00:00:00
extras                                  | 2.9 kB | 00:00:00
updates                                 | 2.9 kB | 00:00:00
(1/7): base/7/x86_64/group_gz          | 153 kB | 00:00:00
(2/7): epel/x86_64/group_gz            | 96 kB | 00:00:00
(3/7): epel/x86_64/updateinfo           | 1.0 MB | 00:00:00
(4/7): epel/x86_64/primary_db           | 7.0 MB | 00:00:00
(5/7): base/7/x86_64/primary_db         | 6.1 MB | 00:00:00
(6/7): extras/7/x86_64/primary_db       | 247 kB | 00:00:00
(7/7): updates/7/x86_64/primary_db      | 16 MB | 00:00:10
Resolving Dependencies
--> Running transaction check
--> Package device-mapper-persistent-data.x86_64 0:0.8.5-3.el7 9.2 will be installed
--> Processing Dependency: libaio.so.1(LIBAIO 0.4)(64bit) for package: device-mapper-persistent-data-0.8.5-3.el7 9.2.x86_64
--> Processing Dependency: libaio.so.1(LIBAIO 0.1)(64bit) for package: device-mapper-persistent-data-0.8.5-3.el7 9.2.x86_64
--> Processing Dependency: libaio.so.1()(64bit) for package: device-mapper-persistent-data-0.8.5-3.el7 9.2.x86_64
--> Package lvm2.x86_64 7:2.02.187-6.el7 9.5 will be installed
--> Processing Dependency: lvm2-libs = 7:2.02.187-6.el7 9.5 for package: 7:lvm2-2.02.187-6.el7 9.5.x86_64
--> Processing Dependency: liblvm2app.so.2.2(Base)(64bit) for package: 7:lvm2-2.02.187-6.el7 9.5.x86_64
--> Processing Dependency: libdevmapper-event.so.1.02(Base)(64bit) for package: 7:lvm2-2.02.187-6.el7 9.5.x86_64
--> Processing Dependency: liblvm2app.so.2.2()(64bit) for package: 7:lvm2-2.02.187-6.el7 9.5.x86_64
--> Processing Dependency: libdevmapper-event.so.1.02()(64bit) for package: 7:lvm2-2.02.187-6.el7 9.5.x86_64
--> Package yum-utils.noarch 0:1.1.31-54.el7 8 will be installed
--> Processing Dependency: python-kitchen for package: yum-utils-1.1.31-54.el7 8.noarch
--> Running transaction check
--> Package device-mapper-event-libs.x86_64 7:1.02.170-6.el7 9.5 will be installed
--> Package libaio.x86_64 0:0.3.109-13.el7 will be installed
--> Processing Dependency: device-mapper-event = 7:1.02.170-6.el7 9.5 for package: 7:lvm2-libs-2.02.187-6.el7 9.5.x86_64
--> Package python-kitchen.noarch 0:1.1.1-5.el7 will be installed
--> Processing Dependency: python-chardet for package: python-kitchen-1.1.1-5.el7.noarch
--> Running transaction check
--> Package device-mapper-event.x86_64 7:1.02.170-6.el7 9.5 will be installed
--> Package python-chardet.noarch 0:2.2.1-3.el7 will be installed
--> Finished Dependency Resolution
```



```
--> Package device-mapper-event.x86_64 7:1.02.170-6.el7 9.5 will be installed
--> Package python-chardet.noarch 0:2.2.1-3.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

Package Arch Version Repository Size
-----
Installing:
device-mapper-persistent-data x86_64 0.8.5-3.el7 9.2 updates 423 k
lvm2 x86_64 7:2.02.187-6.el7 9.5 updates 1.3 M
yum-utils noarch 1.1.31-54.el7 8 base 122 k
Installing for dependencies:
device-mapper-event x86_64 7:1.02.170-6.el7 9.5 updates 192 k
device-mapper-event-libs x86_64 7:1.02.170-6.el7 9.5 updates 192 k
libaio x86_64 0.3.109-13.el7 base 24 k
lvm2-libs x86_64 7:2.02.187-6.el7 9.5 updates 1.1 M
python-chardet noarch 2.2.1-3.el7 base 227 k
python-kitchen noarch 1.1.1-5.el7 base 267 k

Transaction Summary
Install 3 Packages (+6 Dependent packages)
Total download size: 3.8 M
Installed size: 11 M
Downloading packages:
(1/9): device-mapper-event-libs-1.02.170-6.el7 9.5.x86_64.rpm | 192 kB | 00:00:00
(2/9): device-mapper-persistent-data-0.8.5-3.el7 9.2.x86_64.rpm | 423 kB | 00:00:00
(3/9): device-mapper-event-1.02.170-6.el7 9.5.x86_64.rpm | 192 kB | 00:00:00
(4/9): lvm2-2.02.187-6.el7 9.5.x86_64.rpm | 1.3 MB | 00:00:00
(5/9): lvm2-libs-2.02.187-6.el7 9.5.x86_64.rpm | 1.1 MB | 00:00:00
(6/9): libaio-0.3.109-13.el7.x86_64.rpm | 24 kB | 00:00:00
(7/9): python-kitchen-1.1.1-5.el7.noarch.rpm | 267 kB | 00:00:00
(8/9): yum-utils-1.1.31-54.el7 8.noarch.rpm | 122 kB | 00:00:00
(9/9): python-chardet-2.2.1-3.el7.noarch.rpm | 227 kB | 00:00:00
-----
Total 0.7 MB/s | 3.8 MB | 00:00:00
```

## 4. 利用 sudo 权限安装 docker

```
sudo yum install docker-ce
```

实验中具体安装过程如下所示：

```
[root@ecs-8985 ~]# sudo yum-config-manager --add-repo
Loaded plugins: fastestmirror
Usage: yum-config-manager [options] [section ...]

Command line error: --add-repo option requires an argument
[root@ecs-8985 ~]# https://download.docker.com/linux/centos/docker-ce.repo
-bash: https://download.docker.com/linux/centos/docker-ce.repo: No such file or directory
[root@ecs-8985 ~]# sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
Loaded plugins: fastestmirror
adding repo from: https://download.docker.com/linux/centos/docker-ce.repo
grabbing file https://download.docker.com/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
[root@ecs-8985 ~]#
```

```
---> Package audit-libs-python.x86_64 0:2.8.5-4.el7 will be installed
---> Package checkpolicy.x86_64 0:2.5-8.el7 will be installed
---> Package fuse3-libs.x86_64 0:3.6.1-4.el7 will be installed
---> Package libsemanage-python.x86_64 0:2.5-14.el7 will be installed
---> Package python-IPy.noarch 0:0.75-6.el7 will be installed
---> Package setools-libs.x86_64 0:3.3.8-4.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                                Arch              Version            Repository          Size
=====
Installing:
docker-ce                               x86_64            3:20.10.16-3.el7   docker-ce-stable    22 M
Installing for dependencies:
audit-libs-python                      x86_64            2.8.5-4.el7        base                76 k
checkpolicy                            x86_64            2.5-8.el7          base                295 k
container-selinux                      noarch            2:2.119.2-1.911c772.el7_8 extras              40 k
containerd.io                          x86_64            1.6.4-3.1.el7      docker-ce-stable    33 M
docker-ce-cli                          x86_64            1:20.10.16-3.el7   docker-ce-stable    29 M
docker-ce-rootless-extras             x86_64            20.10.16-3.el7     docker-ce-stable    8.2 M
docker-scan-plugin                     x86_64            0.17.0-3.el7       docker-ce-stable    3.7 M
fuse-overlayfs                         x86_64            0.7.2-6.el7_8      extras              54 k
fuse3-libs                             x86_64            3.6.1-4.el7        extras              82 k
libcgroup                             x86_64            0.41-21.el7        base                66 k
libsemanage-python                    x86_64            2.5-14.el7         base                113 k
policycoreutils-python                 x86_64            2.5-34.el7         base                457 k
python-IPy                             noarch            0.75-6.el7         base                32 k
setools-libs                           x86_64            3.3.8-4.el7        base                620 k
slirp4netns                            x86_64            0.4.3-4.el7_8      extras              81 k
=====

Transaction Summary
Install 1 Package (+15 Dependent packages)

Total download size: 99 M
Installed size: 399 M
Is this ok [y/d/N]:
```

```
Complete!
[root@ecs-8985 ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
[root@ecs-8985 ~]# systemctl start docker
[root@ecs-8985 ~]#
```

```
Dependency Installed:
audit-libs-python.x86_64 0:2.8.5-4.el7      checkpolicy.x86_64 0:2.5-8.el7      container-selinux.noarch 2:2.119.2-1.911c772.el7_8
containerd.io.x86_64 0:1.6.4-3.1.el7      docker-ce-cli.x86_64 1:20.10.16-3.el7   docker-ce-rootless-extras.x86_64 0:20.10.16-3.el7
docker-scan-plugin.x86_64 0:0.17.0-3.el7   fuse-overlayfs.x86_64 0:0.7.2-6.el7_8   fuse3-libs.x86_64 0:3.6.1-4.el7
libcgroup.x86_64 0:0.41-21.el7          libsemanage-python.x86_64 0:2.5-14.el7   policycoreutils-python.x86_64 0:2.5-34.el7
python-IPy.noarch 0:0.75-6.el7          setools-libs.x86_64 0:3.3.8-4.el7   slirp4netns.x86_64 0:0.4.3-4.el7_8
```

最终安装完毕之后，输出 docker 版本，检测安装是否正确，命令如下：

```
docker --version
```

```
docker-scan-plugin.x86_64 0:0.17.0-3.el7      fuse-overlayfs.x86_64 0:0.7.2-6.el7_8      fuse3-libs.x86_64 0:3.6.1-4.el7
libcgroup.x86_64 0:0.41-21.el7              libsemanage-python.x86_64 0:2.5-14.el7      policycoreutils-python.x86_64 0:2.5-34.el7
python-IPy.noarch 0:0.75-6.el7              setools-libs.x86_64 0:3.3.8-4.el7          slirp4netns.x86_64 0:0.4.3-4.el7_8

Complete!
[root@ecs-8985 ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
[root@ecs-8985 ~]# systemctl start docker
[root@ecs-8985 ~]# docker --version
Docker version 20.10.16, build aa7e414
[root@ecs-8985 ~]#
```

5. 为了后续实验的进行，在安装 docker 完毕之后还需要配置 docker 镜像仓库文件。在实验中具体路径为

`/etc/docker/daemon.json`

在 xshell 中使用 vim 编译器创建 daemon.json 文件，并键入如下内容

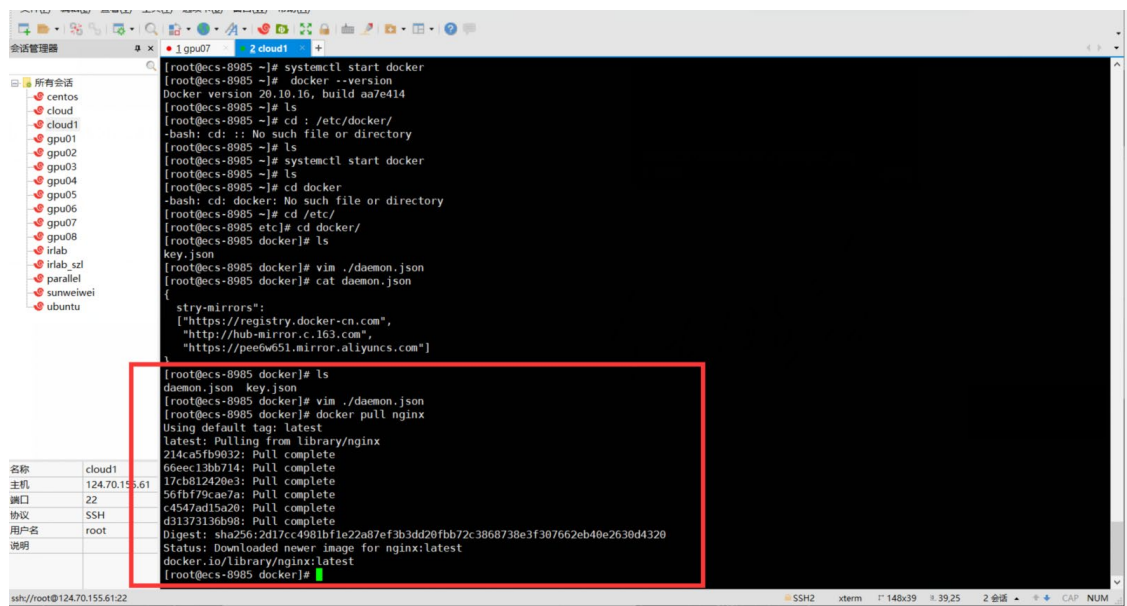
```
1. {
2.     "registry-mirrors": [
3.         "https://registry.docker-cn.com",
4.         "http://hub-mirror.c.163.com",
5.         "https://pee6w651.mirror.aliyuncs.com"
6.     ]
7. }
```



### 三、配置 nginx 与 tomcat 服务

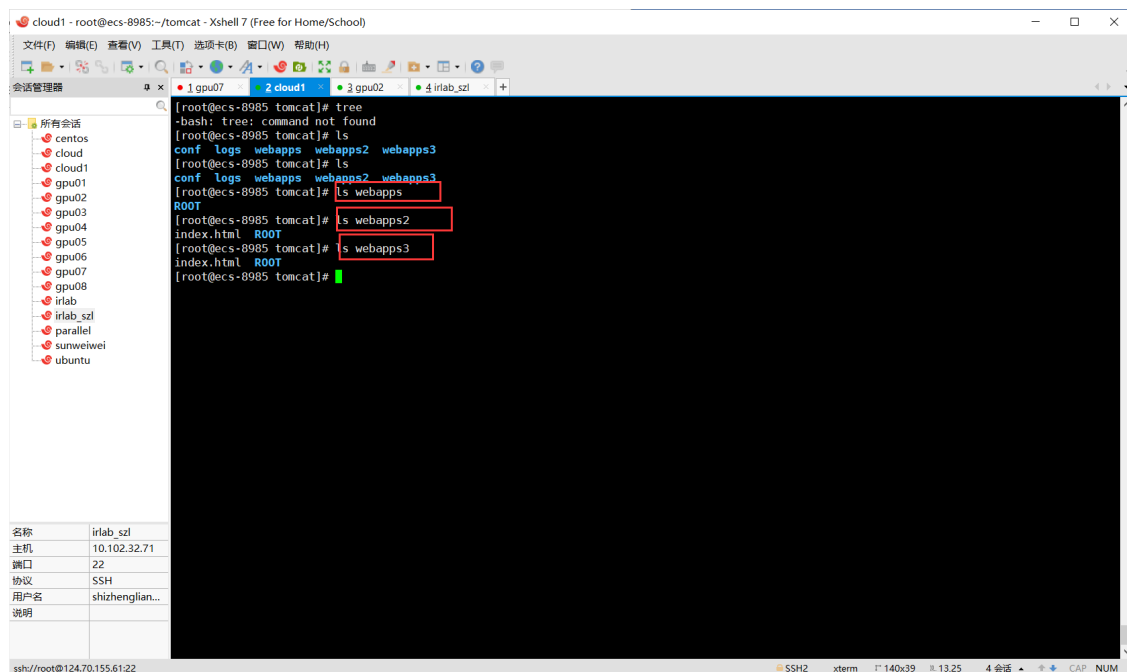
1. 在本次实验中需要配置 nginx 与 tomcat 服务，因此首先在终端中输入如下命令进行安装

1. docker pull nginx
2. docker pull tomcat



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

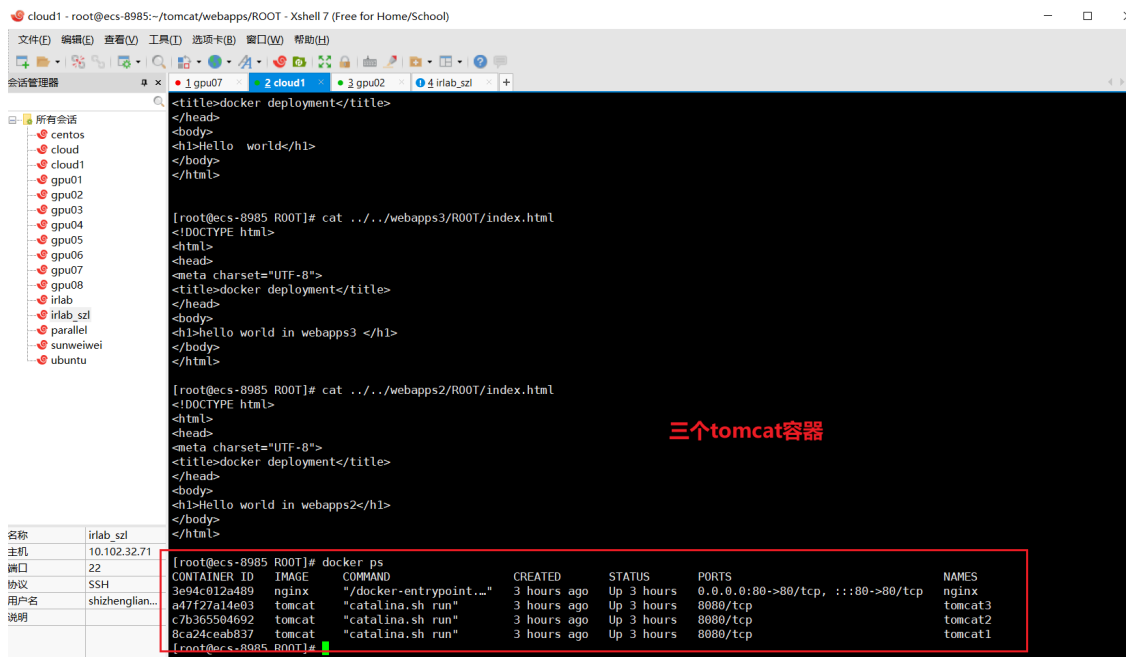
```

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 ~/ngi
nx/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 R00T]# ls
index.html
[root@ecs-8985 R00T]# vim index.html
[root@ecs-8985 R00T]# docker run -d --name tomcat1 -v ~/tomcat/webapps:/usr/local/tomcat/webapps tomcat
ebf63119066b5d0f54d2b6c0bc66fc92ca3c7ca6dc9dfdaf6a242d3b5097aab72
[root@ecs-8985 R00T]# docker run -d --name tomcat2 -v ~/tomcat/webapps:/usr/local/tomcat/webapps tomcat
2950ab60435ea0bc9016ab1c186d3bec28d385a78b5be7304029471758a10ef0
[root@ecs-8985 R00T]#

```

此时，输入 `docker ps` 命令进行查看，可以发现如下信息



```
cloud1 - root@ecs-8985:~/tomcat/webapps/ROOT - Xshell 7 (Free for Home/School)
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)
会话管理器
所有会话
centos
cloud
cloud1
gpu01
gpu02
gpu03
gpu04
gpu05
gpu06
gpu07
gpu08
irlab
irlab_szl
parallel
sunweiwei
ubuntu

[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>

[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]#
```

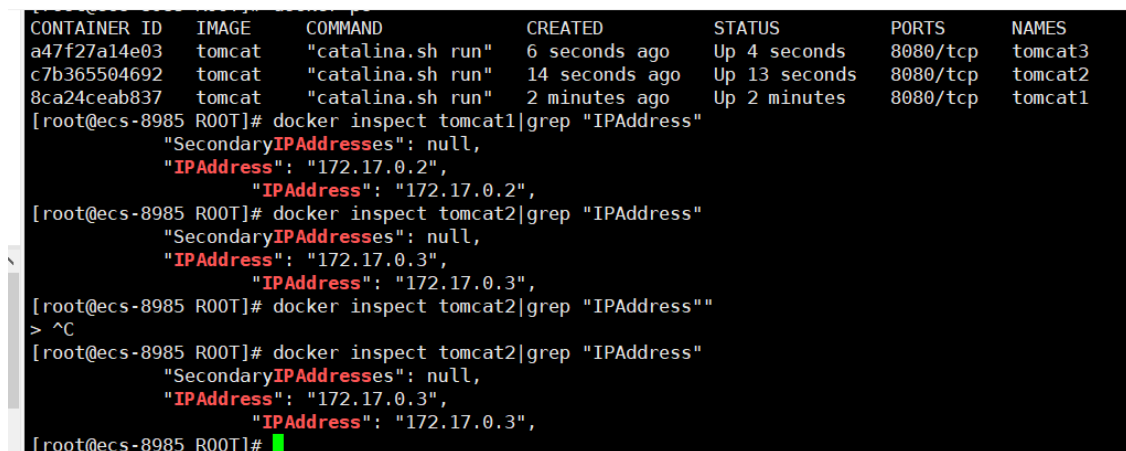
三个tomcat容器

名称	irlab_szl
主机	10.102.32.71
端口	22
协议	SSH
用户名	shizhenglian...
说明	

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 ROOT]# docker inspect tomcat1|grep "IPAddress"
"SecondaryIPAddresses": null,
"IPAddress": "172.17.0.2",
"IPAddress": "172.17.0.2",
[root@ecs-8985 ROOT]# docker inspect tomcat2|grep "IPAddress"
"SecondaryIPAddresses": null,
"IPAddress": "172.17.0.3",
"IPAddress": "172.17.0.3",
[root@ecs-8985 ROOT]# docker inspect tomcat2|grep "IPAddress"
"SecondaryIPAddresses": null,
"IPAddress": "172.17.0.3",
"IPAddress": "172.17.0.3",
[root@ecs-8985 ROOT]#
```

之后在路径: nginx/conf/配置 nginx.conf 文件, 用于后续实现不同负载均衡策略。内容如下所示:

```
1. user nginx;
2. worker_processes 1;
3. error_log /var/log/nginx/error.log warn;
4. pid /var/run/nginx.pid;
5. events {
6.     worker_connections 1024;
7. }
8.
9. http {
10.    include /etc/nginx/mime.types;
11.    default_type application/octet-stream;
12.    log_format main '$remote_addr - $remote_user [$time_local]
13.        "$request" '
14.        '$status $body_bytes_sent "$http_referer" '
15.        '"$http_user_agent" "$http_x_forwarded_for"
16.        "$upstream_addr"';
17.    access_log /var/log/nginx/access.log main;
18.    sendfile on;
19.    #tcp_nopush on;
20.    keepalive_timeout 65;
21.    #gzip on;
22.    upstream tomcat {
23.        server 172.17.0.2:8080;
24.        server 172.17.0.3:8080;
25.        server 172.17.0.4:8080;
26.    }
27.    server {
28.        listen 80;
29.        server_name localhost;
30.        location / {
31.            proxy_pass http://tomcat;
32.            proxy_redirect off;
33.            index index.html index.htm;
34.            proxy_set_header Host $host;
35.            proxy_set_header X-Real-IP $remote_addr;
```

```
36.         proxy_set_header X-Real-Port $remote_port;
37.         proxy_set_header X-Forwarded-
    For $proxy_add_x_forwarded_for;
38.     }
39.     location /static/ {
40.         alias /usr/share/nginx/html/;
41.     }
42. }
43. include /etc/nginx/conf.d/*.conf;
44. }
```


#### 四、测试不同的负载均衡策略

在本次实验中需要测试轮询，哈希以及指定权重三种负载均衡算法，需要不断更新.conf 文件实现策略的切换。具体的实现过程如下所示：

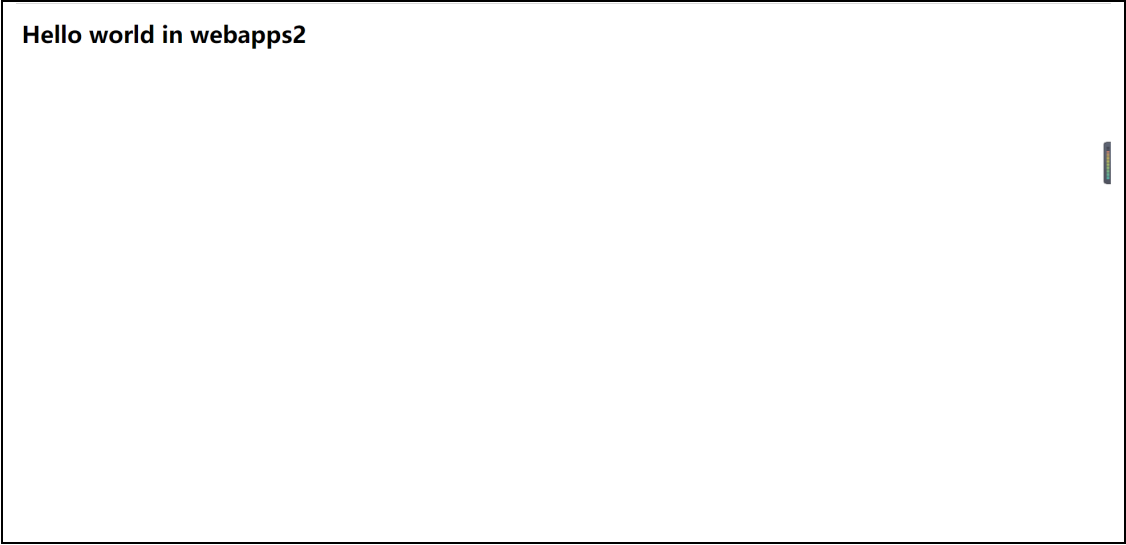
##### 1. 轮询法

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：

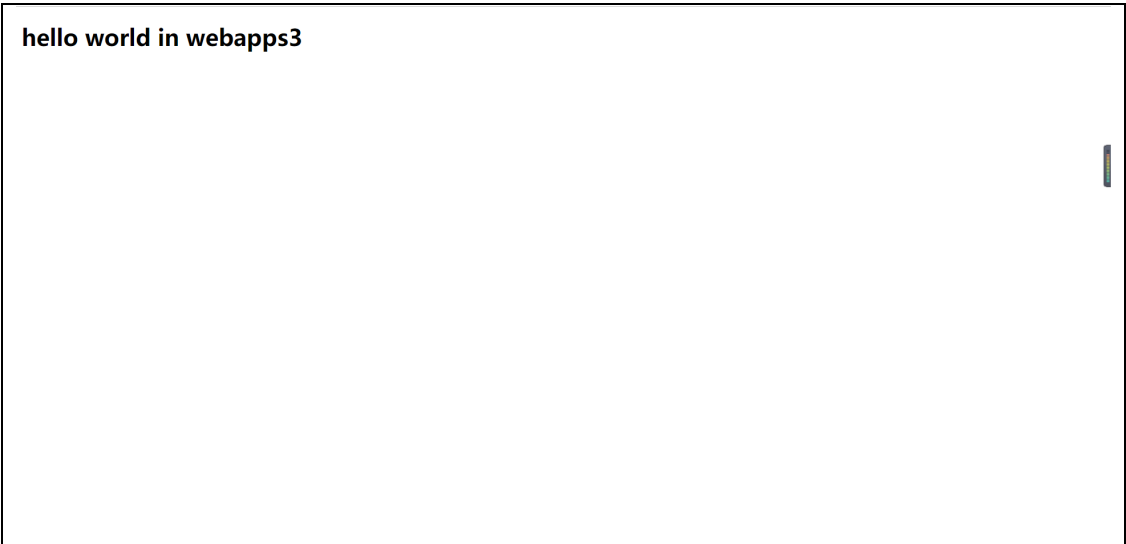


图一：容器一对应的 html 页面

A screenshot of a web browser window. The address bar is empty. The main content area displays the text "Hello world in webapps2" in a black, sans-serif font. The browser's scrollbar is visible on the right side.

Hello world in webapps2

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 scrollbar is visible on the right side.

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 1,2,3 修改完成后的 `~/nginx/conf/nginx.json` 内部代码如下所示：

```
1. user nginx;  
2. worker_processes 1;  
3. error_log /var/log/nginx/error.log warn;
```

```
4. pid /var/run/nginx.pid;
5. events {
6.     worker_connections 1024;
7. }
8.
9. http {
10.    include /etc/nginx/mime.types;
11.    default_type application/octet-stream;
12.    log_format main '$remote_addr - $remote_user [$time_local]
13.        "$request" '
14.        '$status $body_bytes_sent "$http_referer" '
15.        '"$http_user_agent" "$http_x_forwarded_for"
16.        "$upstream_addr"';
17.    access_log /var/log/nginx/access.log main;
18.    sendfile on;
19.    #tcp_nopush on;
20.    keepalive_timeout 65;
21.    #gzip on;
22.    upstream tomcat {
23.        server 172.17.0.2:8080 weight=1;
24.        server 172.17.0.3:8080 weight=3;
25.        server 172.17.0.4:8080 weight=5;
26.    }
27.    server {
28.        listen 80;
29.        server_name localhost;
30.        location / {
31.            proxy_pass http://tomcat;
32.            proxy_redirect off;
33.            index index.html index.htm;
34.            proxy_set_header Host $host;
35.            proxy_set_header X-Real-IP $remote_addr;
36.            proxy_set_header X-Real-Port $remote_port;
37.            proxy_set_header X-Forwarded-
38.                For $proxy_add_x_forwarded_for;
39.        }
40.        location /static/ {
```

```
40.         alias /usr/share/nginx/html/;
41.     }
42. }
43. include /etc/nginx/conf.d/*.conf;
44. }
```

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. upstream tomcat {
2.     ip_hash;
3.     server 172.17.0.2:8080;
4.     server 172.17.0.3:8080;
5.     server 172.17.0.4:8080;
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 172.17.0.2:8080` 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 `172.17.0.2:8080` 对应的容器，说明实验结果正确。

```
[root@ecs-8985 conf]# cd ../
[root@ecs-8985 nginx]# ls
conf  logs  www
[root@ecs-8985 nginx]# cd logs/
[root@ecs-8985 logs]# ls
access.log  error.log
[root@ecs-8985 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/2022:09:34:11 +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：

### 1. 关于负载均衡和 Nginx 的理解

负载均衡（Load Balance），它在网络现有结构之上可以提供一种廉价、有效、透明的方法来扩展网络设备和服务器的带宽，并可以在一定程度上增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性等。用官网的话说，它充当着网络流中“交通指挥官”的角色，“站在”服务器前处理所有服务器端和客户端之间的请求，从而最大程度地提高响应速率和容量利用率，同时确保任何服务器都没有超负荷工作。如果单个服务器出现故障，负载均衡的方法会将流量重定向到其余的集群服务器，以保证服务的稳定性。当新的服务器添加到服务器组后，也可通过负载均衡的方法使其开始自动处理客户端发来的请求。

Nginx 的一个常用功能便是负载均衡。Nginx 作为一个基于 C 实现的高性能 Web 服务器，可以通过系列算法解决上述的负载均衡问题。并且由于它具有高并发、高可靠性、高扩展性、开源等特点，成为开发人员常用的反向代理工具

### 2. 负载均衡常用算法

负载均衡常用算法包括轮询法、加权轮询、哈希映射以及其他算法等。

#### 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  $n$  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。

## 4、其他算法（例如 URL hash、最小连接数算法）

（1）`url_hash` 算法是根据请求的 URL 的 `hash` 值来分配服务器。该算法的特点是，相同 URL 的请求会分配给固定的服务器，当存在缓存的时候，效率一般较高。然而 Nginx 默认不支持这种负载均衡算法，需要依赖第三方库。

（2）最小连接数（Least Connections）算法假设共有 `n` 台服务器，当有新的请求出现时，遍历服务器节点列表并选取其中连接数最小的一台服务器来响应当前请求。连接数可以理解为当前处理的请求数。

## 体会

本次实验主要涉及到服务器负载均衡算法的测试，通过实际的配置体会不同算法的作用和效果。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。

Nginx 是本次实验的实现工具。作为一个基于 C 实现的高性能 Web 服务器，Nginx 可以通过系列算法解决上述的负载均衡问题。并且由于它具有高并发、高可靠性、高扩展性、开源等特点，成为开发人员常用的反向代理工具。

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。