**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 | |
| **实验目的：**在云端用docker部署nginx和tomcat实现负载均衡  **具体包括：**  1. 要求部署一台Nginx和三台Tomcat服务器 2. Ngnix需要实现三种策略： 1. 轮询。  2. 权重，三台服务器的权重为1，2，5；  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Ngnix三种策略的结果） | | | |
| **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：**  本次实验要求部署一台Nginx和三台Tomcat服务器，并利用Ngnix实现三种策略。实验步骤可以分为：   1. 华为云的购买 2. Docker环境的配置 3. 相关软件环境的配置 4. 更改配置文件测试不同的实现策略   详细实验步骤如下所示。  **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：     1. **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 |  1. **设置docker：**为docker添加yml源        1. 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：        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 |      1. 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. ]     **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：      **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>** 4. **<head>** 5. **<meta** charset="UTF-8"**>** 6. **<title>**docker deployment**</title>** 7. **</head>** 9. **<body>** 10. **<h1>**hello, world**</h1>** 11. **</body>** 13. **</html>**     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>** 4. **<head>** 5. **<meta** charset="UTF-8"**>** 6. **<title>**docker deployment**</title>** 7. **</head>** 9. **<body>** 10. **<h1>**Hello world in webapps2**</h1>** 11. **</body>** 13. **</html>**   网页3代码：   1. <!DOCTYPE html**>** 2. **<html>** 4. **<head>** 5. **<meta** charset="UTF-8"**>** 6. **<title>**docker deployment**</title>** 7. **</head>** 9. **<body>** 10. **<h1>**Hello world in webapps3**</h1>** 11. **</body>** 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      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 tomca   输入之后，终端显示内容如下图所示：      此时，输入docker ps命令进行查看，可以发现如下信息    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    之后在路径./data/nginx/conf下配置nginx.conf文件，用于后续实现不同负载均衡策略。内容如下所示（默认配置三个服务器的权重是1,2,5）   1. # For more information on configuration, see: 2. #   \* Official English Documentation: http://nginx.org/en/docs/ 3. #   \* Official Russian Documentation: http://nginx.org/ru/docs/ 5. user nginx; 6. worker\_processes auto; 7. error\_log /var/log/nginx/error.log; 8. pid /run/nginx.pid; 10. # Load dynamic modules. See /usr/share/nginx/README.dynamic. 11. include /usr/share/nginx/modules/\*.conf; 13. events { 14. worker\_connections 1024; 15. } 17. http { 18. log\_format  main  '$remote\_addr - $remote\_user [$time\_local] "$request" ' 19. '$status $body\_bytes\_sent "$http\_referer" ' 20. '"$http\_user\_agent" "$http\_x\_forwarded\_for"'; 22. access\_log  /var/log/nginx/access.log  main; 24. sendfile            on; 25. tcp\_nopush          on; 26. tcp\_nodelay         on; 27. keepalive\_timeout   65; 28. types\_hash\_max\_size 2048; 30. include             /etc/nginx/mime.types; 31. default\_type        application/octet-stream; 33. # Load modular configuration files from the /etc/nginx/conf.d directory. 34. # See http://nginx.org/en/docs/ngx\_core\_module.html#include 35. # for more information. 36. include /etc/nginx/conf.d/\*.conf; 38. server { 39. listen       80 default\_server; 40. listen       [::]:80 default\_server; 41. server\_name  172.18.0.1; 42. root         /usr/share/nginx/html; 44. # Load configuration files for the default server block. 45. include /etc/nginx/**default**.d/\*.conf; 47. location / { 48. proxy\_pass http://pic; 49. } 51. error\_page 404 /404.html; 52. location = /40x.html { 53. } 55. error\_page 500 502 503 504 /50x.html; 56. location = /50x.html { 57. } 58. } 60. upstream pic{ 61. server 172.18.0.1:8081 weight=1; 62. server 172.18.0.1:8082 weight=2; 63. server 172.18.0.1:8083 weight=5; 64. } 66. }   **之后使用如下命令创建nginx容器**   1. docker run --name mynginx -d -p 80:80  -v /data/nginx/conf/nginx.conf:/etc/nginx/nginx.conf  -v /data/nginx/logs:/var/log/nginx -d docker.io/nginx   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，并且根据实验要求，  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**   |  | | --- | |  |   **图三：容器三对应的html页面**   1. 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. } 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-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. }   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。 | | | |
| **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   1. **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 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 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。 | | | |