

本科毕业论文（设计）

题 目 后台集群服务设计

学 院 计算机与网络科学学院

专 业 网络工程

年 级 2015级

学 号 222015321210122

姓 名 李浩

指 导 教 师 于显平

成 绩

年 月 日

后台集群服务设计

李浩

西南大学计算机与网络科学学院软件学院学院，重庆 400715

**摘要：**由于C/B模型的简单性、易管理性和易维护性，C/B计算模式在互联网中被大量采用。自万维网出现以来，以其简单的方式将图文并茂的信息传递给大众，受到全世界用户的欢迎。而Web也逐渐从一种内容发送机制成为一种服务平台，大量的服务和应用(比如支付宝、淘宝、抖音)都围绕着Web进行。这促进Internet用户剧烈增长和Internet流量爆炸式地增长。以往的单机服务在这种工作场景下，会产生响应超时、信息丢失等一些列影响信息安全、用户体验的问题，带来极大的运营维护成本。因此，对硬件和软件实现可伸缩性、高可用性、可管理性、价格有效性的需求不断增长，促进了服务的发展。

本文主要就是通过Docker虚拟容器技术实现一个简单的小型集群服务。

**关键词**：Web；服务集群；容器技术

**Design of background service cluster**

LI Hao

School of Computer and Information Science, Southwest University, Chongqing 400715, PR China

**Abstract:** Due to the simplicity, manageability and ease of maintenance of the C/B model, the C/B computing model is widely used in the Internet. Since the advent of the World Wide Web, the message has been conveyed to the public in a simple way, and has been welcomed by users all over the world. The Web has gradually become a service platform from a content delivery mechanism, and a large number of services and applications (such as Alipay, Taobao, and Vibrato) are all around the Web. This has contributed to the dramatic growth of Internet users and the explosive growth of Internet traffic. In the past, this kind of single-machine service will cause problems such as response timeout and information loss, which will affect information security and user experience, and bring great operation and maintenance costs. As a result, the need for hardware and software to achieve scalability, high availability, manageability, and price effectiveness continues to grow, driving the growth of services.

**Key words:** Web; Service cluster; Vitrtual Container

第1章 导论

**1.1 当代互联网Web的发展**

Internet经过近半个世纪的发展，规模已经空前宏大。智能手机的出现，使得人们不用电脑也能体验Internet带来的便利生活。种种新技术和硬件的应用，让Internet用户以及Internet流量呈现爆炸式的增长。

**1.2 建立服务集群的必要性**

随着Internet用户剧烈增长和Internet流量爆炸式地增长，以往的单机服务在这种工作场景下，会产生响应超时、信息丢失等一些列影响信息安全、用户体验的问题，带来极大运营维护成本。因此，对硬件和软件实现可伸缩性、高可用性、可管理性、价格有效性的需求不断增长，建立服务集群来解决这些问题，也就是必要的。

**1.3 课题主要工作**

**1.3.1 课题探讨**

服务集群系统设计考虑到方方面面，比如硬件选择、服务机房分布、规模大小等等。一个人实现完整系统不仅经济成本高，也相当困难。因此，通过容器技术可以在短时间内实现较小规模服务集群，同时由于容器技术特性，可以通过配置文件迅速地在其它地方进行相同服务集群部署。

**1.3.2 主要工作**

通过Docker容器技术，实现小型的集群网络，将Nginx、PHP、redis、MySQL服务部署在上面，实现一个简单的集群服务功能

第2章 服务集群设计

**2.1 相关技术**

**2.1.1 Docker容器技术**

容器是轻量级的操作系统级虚拟化，可以让我们在一个资源隔离的进程中运行应用及其依赖项。运行应用程序所必需的组件都将打包成一个镜像并可以复用。执行镜像时，它运行在一个隔离环境中，并且不会共享宿主机的内存、CPU 以及磁盘，这就保证了容器内进程不能监控容器外的任何进程。

容器与虚拟机有一定差别。虚拟机通常包含整个操作系统及其应用程序，同时也需要运行一个 hypervisor 来控制虚拟机。因为虚拟机包括操作系统，所有大小一般是几个 G。使用虚拟机的缺点之一是启动操作系统以及初始化托管应用会花费几分钟的时间，但是，容器是轻量级的而且大小在 M 以内。性能方面，与 VM 相比，容器表现更加出色，并且几乎可以秒启动。其对比如图2.1.1所示：

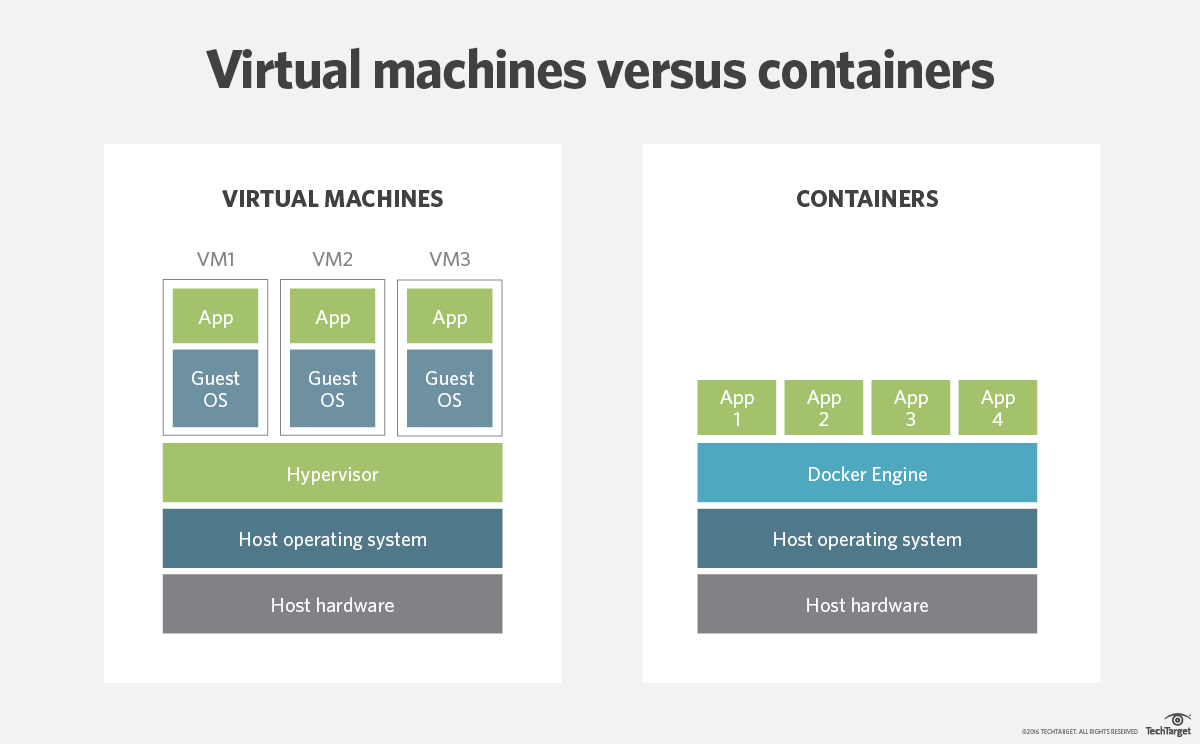


图 2.1.1 虚拟机与容器对比

Virtual machines versus containers

(1)Docker容器优点

①敏捷环境：

容器技术最大的优点是创建容器实例比创建虚拟机示例快得多，容器轻量级的脚本可以从性能和大小方面减少开销。

②提高生产力：

容器通过移除跨服务依赖和冲突提高了开发者的生产力。每个容器都可以看作是一个不同的微服务，因此可以独立升级，而不用担心同步。

③版本控制：每一个容器的镜像都有版本控制，这样就可以追踪不同版本的容器，监控版本之间的差异等等。

④运行环境可移植：

容器封装了所有运行应用程序所必需的相关的细节比如应用依赖以及操作系统。这就使得镜像从一个环境移植到另外一个环境更加灵活。比如，同一个镜像可以在 Windows 或 Linux 或者 开发、测试或 stage 环境中运行。

⑤标准化：

大多数容器基于开放标准，可以运行在所有主流 Linux 发行版、Microsoft 平台等等。

⑥安全：

容器之间的进程是相互隔离的，其中的基础设施亦是如此。这样其中一个容器的升级或者变化不会影响其他容器。

(2)Docker容器生命周期

Docker在本地通过镜像实例化产生容器，以支持各服务运行。提交容器，能够将容器服务的运行状态保存下来。将本地镜像push到远程仓库，其他服务器上的Docker将需要的镜像pull到本地，然后在原有基础上对镜像进行修改，实现镜像部署以及扩展

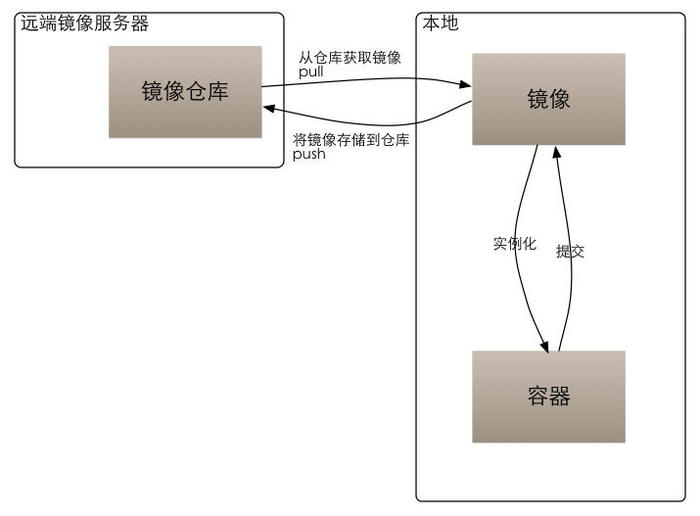


图3.2.1 Docker生命周期

Docker life cycle

(2)Docker容器应用

正是Docker容器相关优点，可以简单快捷地在一台宿主机下实现模拟服务集群，并能迅速移植到其他宿主机上。

**2.1.2 KeepAlived与Nginx**

(1)KeepAlived

keepalived是以VRRP协议为实现基础的，VRRP全称Virtual Router Redundancy Protocol，即虚拟路由冗余协议。

虚拟路由冗余协议，可以认为是实现路由器高可用的协议，即将N台提供相同功能的路由器组成一个路由器组，这个组里面有一个master和多个backup，master上面有一个对外提供服务的vip（该路由器所在局域网内其他机器的默认路由为该vip），master会发组播，当backup收不到vrrp包时就认为master宕掉了，这时就需要根据VRRP的优先级来选举一个backup当master。这样的话就可以保证路由器的高可用了。

(2)Nginx

Nginx是一款自由的、开源的、高性能的HTTP服务器和反向代理服务器；同时也是一个IMAP、POP3、SMTP代理服务器；Nginx可以作为一个HTTP服务器进行网站的发布处理，另外Nginx可以作为反向代理进行负载均衡的实现。

①正向代理: 把局域网外的Internet想象成一个巨大的资源库，则局域网中的客户端要访问Internet，则需要通过代理服务器来访问，这种代理服务就称为正向代理。

②反向代理: 局域网向Internet提供资源服务，让Internet上的其他客户端来访问局域网内的资源，使它们必须通过一个代理服务器来进行访问，这种服务就称为反向代理。正向代理和反向代理逻辑正好相反。

③负载均衡: nginx可以处理大量并发连接，请求到来后，nginx可将其转发给任意数量的后台服务器进行处理，这等于将负载均衡分散到整个集群。

**2.1.3 PHP**

(1)PHP介绍

PHP（外文名:PHP: Hypertext Preprocessor，中文名：“超文本预处理器”）是一种通用开源脚本语言。语法吸收了C语言、Java和Perl的特点，利于学习，使用广泛，主要适用于Web开发领域。PHP 独特的语法混合了C、Java、Perl以及PHP自创的语法。

(2)PHP特点

①它可以比CGI或者Perl更快速地执行动态网页

②用PHP做出的动态页面与其他的编程语言相比，PHP是将程序嵌入到HTML（标准通用标记语言下的一个应用）文档中去执行，执行效率比完全生成HTML标记的CGI要高许多

③PHP还可以执行编译后代码，编译可以达到加密和优化代码运行，使代码运行更快

(3)PHP-FPM

PHP-FPM(PHP FastCGI Process Manager)意：PHP FastCGI 进程管理器，用于管理PHP进程池的软件，用于接受web服务器的请求。它提供了更好的PHP进程管理方式，可以有效控制内存和进程、可以平滑重载PHP配置。

**2.1.4 redis**

(1)redis介绍

redis是Nosql数据库，是一个key-value存储系统。虽然redis是key-value的存储系统，但是redis支持的value存储类型是非常的多，比如字符串、链表、集合、有序集合和哈希。

(2)redis特点

①分布式 读写分离模式

②可用于缓存消息，按key设置过期时间，过期后自动删除

③支持丰富数据类型

④支持事务，操作都是原子性，所谓原子性就是对数据的更改要么全部执行，要不全部不执行。

⑤ redis具有快速和持久化的特征，速度快，因为数据存在内存中。

⑥单线程，利用redis队列技术并将访问变为串行访问，消除了传统数据库串行控制的开销

(3)redis-sentinel(redis哨兵)

Redis-Sentinel是Redis官方推荐的高可用性(HA)解决方案，当用Redis做Master-slave的高可用方案时，假如master宕机了，Redis本身(包括它的很多客户端)都没有实现自动进行主备切换，而Redis-sentinel本身也是一个独立运行的进程，它能监控多个master-slave集群，发现master宕机后能进行自动切换

特点:

①不时地监控redis是否按照预期良好地运行;

②如果发现某个redis节点运行出现状况，能够通知另外一个进程(例如它的客户端);

③能够进行自动切换。当一个master节点不可用时，能够选举出master的多个slave(如果有超过一个slave的话)中的一个来作为新的master,其它的slave节点会将它所追随的master的地址改为被提升为master的slave的新地址。

**2.1.5 MySQL**

(1)MySQL介绍

MySQL是一个关系型数据库管理系统，由瑞典MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件。

(2)MySQL特点

①MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

②MySQL所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

(3)MySQL主从备份

MySQL数据库支持单向、双向、链式级联、环状等不同业务场景的复制。在复制过程中，一台服务器充当主服务器（Master），接收来自用户的内容更新，而一个或多个其他的服务器充当从服务器（Slave），接收来自主服务器binlog文件的日志内容，解析出SQL重新更新到从服务器，使得主从服务器数据达到一致。

**2.2 开发环境及工具**

**2.2.1 Linux操作系统(Ubuntu)**

Linux是一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务、支持多线程和多CPU的操作系统。它能运行主要的UNIX工具软件、应用程序和网络协议。它支持32位和64位硬件。Linux继承了Unix以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统

**2.2.2 JetBrain工具**

Jetbrin开发工具集成各种插件以及命令行终端，支持多种开发语言，其智能提示以及自动引入极大地提高开发效率

第3章 系统架构

**3.1 整体架构**

以实现用户注册、登录功能来体现出服务集群效果

客户端

VIP(虚拟IP)

KeepAlived从（Nginx）

Keepalived主(Nginx)

PHP\_2

PHP\_3

PHP\_1

MySQL\_Backup

MySQL\_Master

Redis Sentinel

Redis\_backup

Redis\_master

图2.2.1 集群服务架构图

Cluster service architecture diagram

如图2.2.1集群服务架构图所示，整个服务集群分为以下部分：

①客户端与服务端：服务端通过KeepAlived的VIP，向客户端提供唯一的IP地址

②KeepAlived与Nginx：KeepAlived与Nginx搭配使用，保证服务的可用性

③PHP：通过Nginx进行负载均衡，将请求分发到不同的PHP服务器上进行处理，减小单机请求处理压力

④redis:由于有多台PHP服务器，使用redis进行数据的共享，比如讲session数据保存到redis。redis采用主从备份，同时运用redis的sentinel功能对主从服务器进行监视

⑤MySQL:主要数据库，实现主从备份，防止数据丢失。

**3.2 Docker设计**

在Docker里面可以部署各种服务，而容器之间运行的服务又被隔离开，互不影响，因此可以使用Docker来实现物理服务器模拟

**3.2.1 Docker部署**

网易云的镜像仓库([https://c.163yun.com/dashboard#/m/mirrorRepo/](https://c.163yun.com/dashboard" \l "/m/mirrorRepo/))作为Docker远程镜像仓库，同时使用docker-compose工具对Dcoker容器进行编排管理。

**3.2.2 Docker启动**

启动Docker，需要多次切换目录以及运行docker-compose up 命令，所以使用Golang脚本，对docker-compose up命令统一执行和终止，提高效率

**3.3 KeepAlived与Nginx设计**

Nginx与KeepAlived搭配使用

客户端

VIP(虚拟IP)

KeepAlived从（Nginx）

Keepalived主(Nginx)

Web2

Web3

Web1

图2.1.1 Nginx与Keepalived组合

Combination of Nginx and Keepalived

Nginx与Keepalived组合的服务架构如图2.1.1所示，Nginx与KeepAlived运行在同一服务器上。通过设置虚拟IP提供给客户端进行访问。KeepAlived应用服务，主从之间以心跳探测机制来判断主服务器是否正常访问，正常的话，虚拟IP绑定在主服务器上，由主服务通过Nginx来转发请求至后台集群。如果访问不可达，则虚拟IP会自动绑定到从服务器上，由从服务器通过Nginx来转发请求至后台集群。

**3.4 PHP设计**

**3.4.1 PHP架构设计**

(1)单台服务架构

单台服务架构上，使用Nginx接受请求，当Nginx收到请求后，根据路由匹配规则，运用反向代理将请求从内部转发到满足路由规则的PHP服务器上，PHP处理完后，将结果返回给Nginx，Nginx再返回给客户端。流程如图3.4.1所示:



图3.4.1 Nginx转发请求至PHP服务器

Nginx forwarding request to PHP server

(2)集群服务架构

这里的集群服务是指多台运行着PHP的服务器形成一个Web集群，接受并处理来自上游服务器的请求。配置Nginx的负载均衡模块采取权重策略，进行请求分发

**3.4.2 PHP Web服务设计**

Web服务设计是用来体现出整个集群系统在正常运行状态下的效果，主要实现三个功能：

1. 注册

添加新的账号到数据库

1. 登录

打开首页判断是否登录，如未登录则重定向到登录注册聚合页

1. 首页展示

首页上展示用户名，为更好地展示出效果，在首页上打印出各个服务器的状态以及IP地址

**3.5 redis设计**

**3.5.1 redis主从设置**

设置两台redis服务器，标记为主服务器、从服务器。

(1)主服务器

进行读写操作，提供redis的session功能，同时开启主从备份功能

(2)从服务器

主服务器正常工作的状态下，从服务器复制主服务器产生的所有数据。主服务器出现异常不能继续提供服务时，从服务器升级为主服务器提供服务

**3.5.2 redis-sentine设置**

对redis的主从架构来说，redis提供了主从架构的管理模块，也就是sentine模块。它用来监视redis服务集群的异常，并及时切换集群中各个服务器的角色以保证服务的正常进行

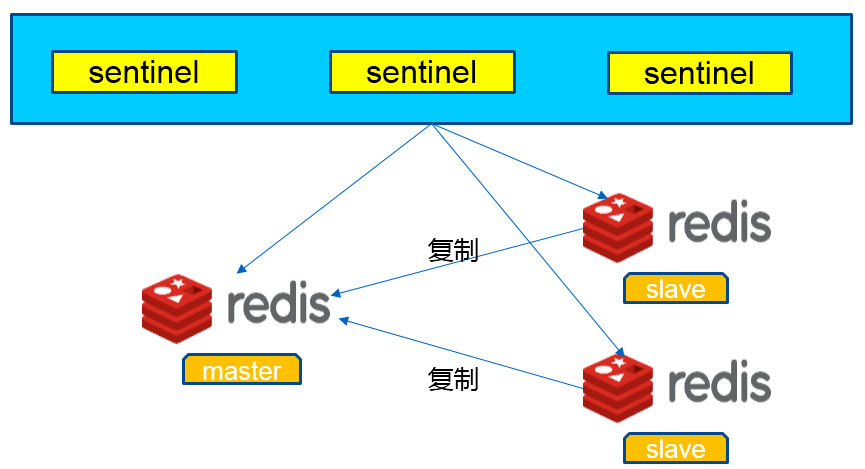


图3.4.1 redis集群与哨兵

redis-cluster and sentinel

**3.6 MySQL设计**

**3.6.1 MySQL架构设计**

设置两台MySQL服务器，标记为主服务器、从服务器。

(1)主服务器

进行写操作，打开主从备份功能

(2)从服务器

同步主服务器数据，进行读操作，减轻主数据库的压力

**3.6.2 MySQL业务设计**

MySQL的业务设计非常简单，总共就涉及一张User

表1 用户信息表

Table1 UserInformation table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 数据类型 | 是否允许空 | 主键 | 功能说明 |
| Id | 自动编号 | 否 | 主键 | 主键，递增 |
| Username | Varchar(32) | 否 |  | 用户名 |
| Password | Varchar(32) | 是 |  | 用户登录密码 |

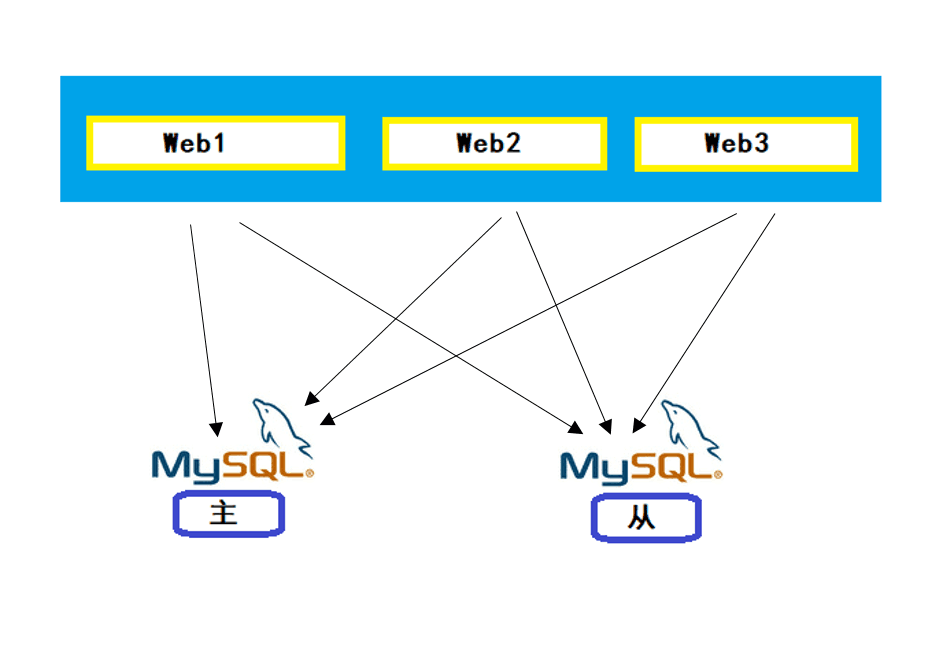
备注：此服务集群系统中，侧重点不在业务上，所以也就实现了一个简单的用户注册登录功能。具体就包括:用户名、用户登录密码

图3.6.1 MySQL架构图

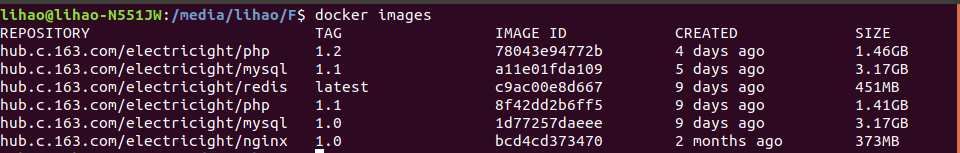
MySQL Architecture Diagram

第4章 系统具体实现

**4.1 Docker**

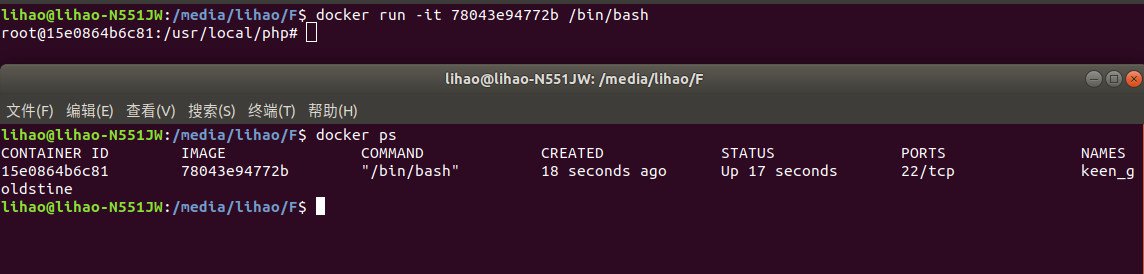
**4.1.1 Docker pull与push**

(1)**列出本地镜像**

docker images

(2)**后台启动容器，验证配置是否正确**

docker run -d {镜像名或ID}

docker ps

(3)**登录网易云镜像仓库**

docker login -u {登录账号} -p {你的网易云密码} hub.c.163.com

Attention:登录账号可前往网易云控台，点击右上角用户名- 基本信息里查看。

返回「**Login Succeded**」即为登录成功。

(4)**标记本地镜像**

docker tag {镜像名或ID} hub.c.163.com/{你的用户名}/{标签名}

你的网易云镜像仓库推送地址为 hub.c.163.com/{你的用户名}/{标签名}

Attention: 此处为你的用户名，不是你的邮箱帐号或者手机号码 登录网易云控制台，页面右上角头像右侧即为「用户名」

推送至不存在的镜像仓库时，自动创建镜像仓库并保存新推送的镜像版本；

推送至已存在的镜像仓库时，在该镜像仓库中保存新推送的版本，当版本号相同时覆盖原有镜像。

(5)**推送至网易云镜像仓库**

docker push hub.c.163.com/{你的用户名}/{标签名}

默认为私有镜像仓库，推送成功后即可在控制台的「镜像仓库」查看。

**4.1.2 Docker启动实现**

Golang调用shell脚本命令实现:

package main

import (

"bytes"

"fmt"

"log"

"os"

"os/exec"

"os/signal"

)

func main() {

count := 0

n := make(chan int, 1)

c := make(chan os.Signal, 1)

signal.Notify(c, os.Interrupt, os.Kill)

startCmds := []string{"cd nginx && docker-compose up",

"cd redis && docker-compose up",

"cd mysql && docker-compose up",

"cd php && docker-compose up"}

endCmds := []string{"docker stop $(docker ps -aq)"}

for i := 0; i <= len(startCmds); i++ {

if i == len(startCmds) {

go listener(endCmds, c, n)

break

}

go exec\_shell(startCmds[i], n)

}

for {

select {

case <-n:

count++

}

if count == len(startCmds)+1 {

break

}

}

for i := 0; i < len(endCmds); i++ {

fmt.Println("命令:"+endCmds[i]+"执行完毕")

}

fmt.Println("执行完毕")

}

func listener(cmd []string, c chan os.Signal, n chan int) {

select {

case <-c:

for i := 0; i < len(cmd); i++ {

fmt.Println("接受到关闭指令,开始关闭容器")

end\_shell(cmd[i])

}

break

}

n <- 1

}

func end\_shell(s string) {

fmt.Println("执行关闭命令:" + s)

cmd := exec.Command("/bin/bash", "-c", s)

var out bytes.Buffer

cmd.Stdout = &out

err := cmd.Start()

if err != nil {

log.Fatal(err)

}

err = cmd.Wait()

if err != nil {

log.Fatal(err)

}

}

func exec\_shell(s string, n chan int) {

fmt.Println("执行命令:" + s)

cmd := exec.Command("/bin/bash", "-c", s)

var out bytes.Buffer

cmd.Stdout = &out

err := cmd.Start()

if err != nil {

log.Fatal(err)

}

fmt.Printf("%s", out.String())

err = cmd.Wait()

if err != nil {

log.Fatal(err)

}

fmt.Println("命令:" + s + "被终止")

n <- 1

}

**4.2 KeepAlived+Nginx**

**4.2.1 KeepAlived**

(1)KeepAlived主服务器配置:

vrrp\_instance VI\_1 {

state MASTER #声明为主服务器

interface eth0 #虚拟IP绑定在eth0网卡上

virtual\_router\_id 51 #虚拟路由标识

priority 100 #优先级

advert\_int 1 #心跳探测频率

authentication {

auth\_type PASS #认证方式

auth\_pass MrUse #认证密码

}

virtual\_ipaddress {

172.18.0.99 #虚拟IP

}

}

(2)KeepAlived从服务器配置:

vrrp\_instance VI\_1 {

state BACKUP #声明为从服务器

interface eth0 #虚拟IP绑定在eth0网卡上

virtual\_router\_id 51 #虚拟路由标识

priority 90 #优先级

advert\_int 1 #心跳探测频率

authentication {

auth\_type PASS #认证方式

auth\_pass MrUse #认证密码

}

virtual\_ipaddress {

172.18.0.99 #虚拟IP

}

}

4.2.2 Nginx转发服务器

(1)负载均衡配置

upstream phpcluster{

server 172.18.0.4:80 weight=1; #第一台权重为1

server 172.18.0.5:80 weight=2; #第二台权重为2

server 172.18.0.6:80 weight=3; #第三台权重为3

}

给3台PHP服务配置负载均衡

(2)监听地址端口以及转发

server {

listen 80; #监听80端口

root /usr/share/nginx/www; #设置web根目录

index index.html index.htm; #默认文件

server\_name localhost; #监听域名

location / {

proxy\_pass <http://phpcluster>; #转发的负载服务名

}

}

4.3 PHP

4.3.1 PHP单机服务实现

Nginx搭配PHP-FPM实现单机服务

server {

listen 80; #监听80端口

root /usr/share/nginx/www; #设置web根目录

index index.html index.htm; #默认文件

server\_name localhost; #监听域名

location \ {

root /usr/share/nginx/www;

index index.html index.htm;

}

location ~ \.php$ { #匹配后缀是.php的请求，转发至PHP-FPM

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

include fastcgi\_params;

}

}

**4.3.2 PHP Web服务构建**

(1)MySQL操作

<?php

include\_once "common.php";

class MySql

{

public static $instance;

private $pdo;

//私有化构造方法

private function \_\_construct()

{

$json = getJson("mysql");

$dsn = sprintf("mysql:dbname=%s;host=%s:%d", $json["dbname"], $json["host"], $json["port"]);

try {

$pdo = new PDO($dsn, $json["user"], $json["password"]);

$this->pdo = $pdo;

return $this;

}

catch (PDOException $exception) {

echo "数据库连接失败:" . $exception->getMessage();

exit();

}

}

//单利模式中获取单一对象

public static function getInstance()

{

if (self::$instance == null) {

self::$instance = new MySql();

}

return self::$instance;

}

//添加用户

public function addUser($username, $password)

{

$checkUser = $this->findUserByUsernameAndPassword($username);

if ($checkUser != null) {

return null;

}

$insertSql = "insert into user (`username`,`password`) values(:username,:password)";

$stmt = self::$instance->pdo->prepare($insertSql);

$stmt->bindParam(":username", $username);

$stmt->bindParam(":password", $password);

if ($stmt->execute()) {

return new User($username, $password);

} else {

return null;

}

}

//查找用户

public function findUserByUsernameAndPassword($username)

{

$sql = "select \* from user where username= :username";

$stmt = self::$instance->pdo->prepare($sql);

$stmt->bindParam(":username", $username);

if (!$stmt->execute()) {

return null;

} else {

$userInfoArray = $stmt->fetchAll(PDO::FETCH\_ASSOC);

if ($userInfoArray[0]!=null) {

return new User($userInfoArray[0]["username"], $userInfoArray[0]["password"]);

} else {

return null;

}

}

}

}

(2)redis

<?php

include\_once "common.php";

//sentinel操作类

class Sentinel

{

public static $sentinelInstance;

private $redis;

//私有化构造方法

private function \_\_construct()

{

$json = getJson("sentinel");

$redis = new Redis();

if ($redis->connect($json["host"], $json["port"])) {

$this->redis = $redis;

}

return $this;

}

//单利模式中获取单一对象

public static function getSentinelInstance()

{

if (self::$sentinelInstance == null) {

self::$sentinelInstance = new Sentinel();

}

return self::$sentinelInstance;

}

//获得主服务器IP及端口

public function getHostAndPort()

{

$result = $this->redis->rawCommand('sentinel', 'get-master-addr-by-name', 'mymaster');

return ['host' => $result[0], 'port' => $result[1]];

}

}

(4)注册功能

<?php

include\_once "database/mysql.php";

$instance = MySql::getInstance();

$username = $\_POST["username"];

$password = $\_POST["password"];

$user = $instance->addUser($username, $password);

if ($user == null) {

echo "注册失败";

} else {

$\_SESSION["user"] = $user;

header("Location:index.php");

exit;

}

(5)登录功能

<?php

include\_once "database/mysql.php";

$instance = MySql::getInstance();

@$username = $\_POST["username"];

@$password = $\_POST["password"];

var\_dump($\_POST);

if ($username == null || $password == null) {

echo "请填写账户名或者密码";

exit();

}

$user = $instance->findUserByUsernameAndPassword($username);

if ($user == null || $user["password"] != $password) {

echo "用户名或者密码错误";

} else {

$\_SESSION["user"] = $user;

header("Location:index.php");

exit;

}

**4.4 redis**

**4.4.1 redis主从备份**

redis主从配置很简单，只需要在从服务器配置文件中加上一下配置，其余使用默认配置即可

replicaof 172.18.0.7 6379

**4.4.2 redis-sentinel**

在哨兵配置中,需要指定监视的主服务器IP及端口，然后哨兵会根据主服务器，找出所有的从服务器，其余使用默认配置即可

sentinel monitor mymaster 172.18.0.7 6379 1

**4.5 MySQL**

**4.5.1 MySQL主从备份**

主服务器配置：

[mysqld]

log-bin=mysql-bin #二进制日志名

binlog\_format=mixed #使用混合复制模式

server-id =100 #MySQL服务器ID

skip\_name\_resolve=ON #跳过域名解析IP，基于域名授权提高效率

从服务器配置:

[mysqld]

relay-log=relay-log #relay-log文件名

relay-log-index=relay-log.index #relay-log-index文件名

server-id=2 #MySQL服务器ID

skip\_name\_resolve=ON #跳过域名解析IP，基于域名授权提高效率

以上是主服务器与从服务器的配置文件，当数据库启动后，还需要在数据库中进行如下设置。

1. 首先进行数据库初始化

mysqld --initialize-insecure

修改data目录及目录下所有文件的权限

chmod 0777 -R data

启动数据库

mysqld\_safe --user=mysql &

(2)主服务器上创建有复制权限的账户

grant replication slave,replication client on \*.\* to ‘repluser’@’172.18.0.11’ identified by ‘replpass’

然后

flush privileges

账户名为:repluser 密码为:replpass 地址设为从服务器地址

(3)从服务器配置

首先在主服务器上运行命令

show master status

查看主服务器的bin-log 以及pos偏移量

然后在从服务器上

change master to master\_host=’172.18.0.10’,master\_user=’repluser’,

master\_password=’replpass’,master\_log\_file=’mysq-bin.000042’,master\_log\_pos=3;

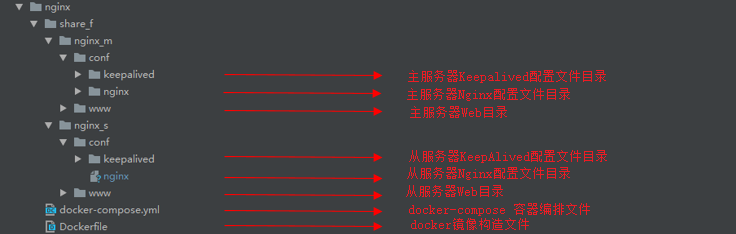
最后运行一下命令启动复制线程

start salve

**4.6 Docker-compose.yml编写**

**4.6.1 Nginx**

(1)目录结构

****

1. **docker-compose.yml**

version: "2"

services:

nginx\_m:

image: hub.c.163.com/electricight/nginx:1.0

container\_name: nginx\_m

privileged: true

command:

- /bin/bash

- -c

- service keepalived start && service nginx start && tail -f /var/log/nginx/error.log

volumes:

- /media/lihao/F/mywork/nginx/share\_f/nginx\_m/conf/nginx:/etc/nginx

- /media/lihao/F/mywork/nginx/share\_f/nginx\_m/conf/keepalived:/etc/keepalived

- /media/lihao/F/mywork/nginx/share\_f/nginx\_m/www:/usr/share/nginx/www

networks:

default:

ipv4\_address: 172.18.0.2

nginx\_s:

image: hub.c.163.com/electricight/nginx:1.0

container\_name: nginx\_s

privileged: true

depends\_on:

- nginx\_m

command:

- /bin/bash

- -c

- service keepalived start && service nginx start && tail -f /var/log/nginx/error.log

volumes:

- /media/lihao/F/mywork/nginx/share\_f/nginx\_s/conf/nginx:/etc/nginx

- /media/lihao/F/mywork/nginx/share\_f/nginx\_s/conf/keepalived:/etc/keepalived

- /media/lihao/F/mywork/nginx/share\_f/nginx\_s/www:/usr/share/nginx/www

networks:

default:

ipv4\_address: 172.18.0.3

networks:

default:

external:

name: backend

**4.6.2 PHP**

(1)目录结构

**(2)docker-compose.yml**

version: "2"

services:

php\_1:

image: hub.c.163.com/electricight/php:1.2

container\_name: php\_1

privileged: true

command:

- /bin/bash

- -c

- php-fpm && nginx && tail -f /usr/local/php/var/log/php-fpm.log

volumes:

- /media/lihao/F/mywork/php/share\_f/conf/etc:/usr/local/php/etc

- /media/lihao/F/mywork/php/share\_f/www:/usr/share/nginx/www

- /media/lihao/F/mywork/php/share\_f/conf/nginxconf:/etc/nginx

- /media/lihao/F/mywork/php/share\_f/conf/lib:/usr/local/php/lib

networks:

default:

ipv4\_address: 172.18.0.4

php\_2:

image: hub.c.163.com/electricight/php:1.2

container\_name: php\_2

privileged: true

command:

- /bin/bash

- -c

- php-fpm && nginx && tail -f /usr/local/php/var/log/php-fpm.log

volumes:

- /media/lihao/F/mywork/php/share\_f/conf/etc:/usr/local/php/etc

- /media/lihao/F/mywork/php/share\_f/www:/usr/share/nginx/www

- /media/lihao/F/mywork/php/share\_f/conf/nginxconf:/etc/nginx

- /media/lihao/F/mywork/php/share\_f/conf/lib:/usr/local/php/lib

networks:

default:

ipv4\_address: 172.18.0.5

php\_3:

image: hub.c.163.com/electricight/php:1.2

container\_name: php\_3

privileged: true

command:

- /bin/bash

- -c

- php-fpm && nginx && tail -f /usr/local/php/var/log/php-fpm.log

volumes:

- /media/lihao/F/mywork/php/share\_f/conf/etc:/usr/local/php/etc

- /media/lihao/F/mywork/php/share\_f/www:/usr/share/nginx/www

- /media/lihao/F/mywork/php/share\_f/conf/nginxconf:/etc/nginx

- /media/lihao/F/mywork/php/share\_f/conf/lib:/usr/local/php/lib

networks:

default:

ipv4\_address: 172.18.0.6

networks:

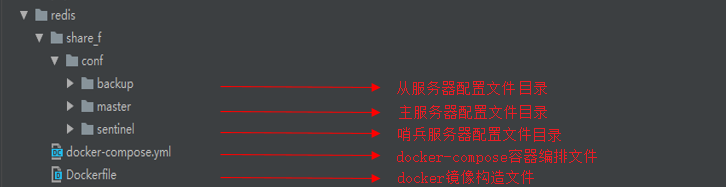
default:

external:

name: backend

**4.6.3 redis**

(1)目录结构



**(2)docker-compose.yml**

version: "2"

services:

redis\_master:

image: hub.c.163.com/electricight/redis:latest

container\_name: redis\_master

privileged: true

command:

- /bin/bash

- -c

- cp /usr/local/redis/etc/redis.conf.bak /usr/local/redis/etc/redis.conf && service keepalived start && redis-server /usr/local/redis/etc/redis.conf && tail -f /usr/local/redis/log/redis.log

volumes:

- /media/lihao/F/mywork/redis/share\_f/conf/master:/usr/local/redis/etc

- /media/lihao/F/mywork/redis/share\_f/sh:/usr/local/redis/sh

- /media/lihao/F/mywork/redis/share\_f/conf/master\_keepalived:/etc/keepalived

networks:

default:

ipv4\_address: 172.18.0.7

redis\_backup:

image: hub.c.163.com/electricight/redis:latest

container\_name: redis\_backup

privileged: true

command:

- /bin/bash

- -c

- cp /usr/local/redis/etc/redis.conf.bak /usr/local/redis/etc/redis.conf && service keepalived start && redis-server /usr/local/redis/etc/redis.conf && tail -f /usr/local/redis/log/redis.log

volumes:

- /media/lihao/F/mywork/redis/share\_f/conf/backup:/usr/local/redis/etc

- /media/lihao/F/mywork/redis/share\_f/sh:/usr/local/redis/sh

- /media/lihao/F/mywork/redis/share\_f/conf/backup\_keepalived:/etc/keepalived

networks:

default:

ipv4\_address: 172.18.0.8

redis\_sentinel:

image: hub.c.163.com/electricight/redis:latest

container\_name: redis\_sentinel

privileged: true

command:

- /bin/bash

- -c

- cp /usr/local/redis/etc/sentinel.conf.bak /usr/local/redis/etc/sentinel.conf && redis-sentinel /usr/local/redis/etc/sentinel.conf && tail -f /usr/local/redis/log/redis.log

volumes:

- /media/lihao/F/mywork/redis/share\_f/conf/sentinel:/usr/local/redis/etc

networks:

default:

ipv4\_address: 172.18.0.9

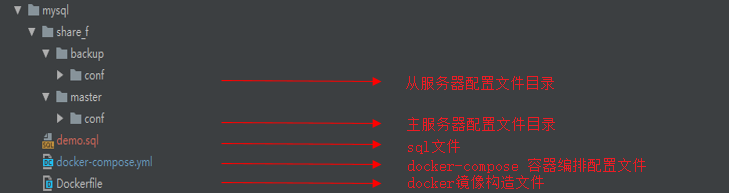
networks:

default:

external:

name: backend

**4.6.4 MySQL**

(1)目录结构

**(2)docker-compose.yml**

version: "2"

services:

mysql\_master:

image: hub.c.163.com/electricight/mysql:1.1

container\_name: mysql\_master

privileged: true

command:

- /bin/bash

- -c

- cp /etc/mysqlcnf/my.cnf /etc/my.cnf && chmod 644 /etc/my.cnf && tail -f /etc/my.cnf

volumes:

- /media/lihao/F/mywork/mysql/share\_f/master/conf:/etc/mysqlcnf

networks:

default:

ipv4\_address: 172.18.0.10

mysql\_backup:

image: hub.c.163.com/electricight/mysql:1.1

container\_name: mysql\_backup

privileged: true

command:

- /bin/bash

- -c

- cp /etc/mysqlcnf/my.cnf /etc/my.cnf && chmod 644 /etc/my.cnf && tail -f /etc/my.cnf

volumes:

- /media/lihao/F/mywork/mysql/share\_f/backup/conf:/etc/mysqlcnf

networks:

default:

ipv4\_address: 172.18.0.11

networks:

default:

external:

name: backend