NginX入门到精通实战

个人介绍

讲师：李振良

资深运维工程师，51CTO知名博主。曾就职在IDC，大数据，金融行业，现任职360公司，经重重磨练，具备丰富的运维实战经验。

技术博客：http://blog.51cto.com/lizhenliang

DevOps技术栈

专注于分享DevOps工具链及经验总结。

微信扫一扫

Linux运维学员群：545214087

课程目录

Nginx初探

- Nginx是什么
- Nginx特性与基本功能
- Nginx架构
- Nginx与Apache对比
- 网络IO模型
- HTTP协议

NginX初探

Nginx是什么

Nginx(engine X)是一个开源、轻量级、高性能的HTTP和反向代理服务器，可以代理HTTP、IMAP/POP3/SMTP和TCP/UDP协议；
其特点是占用内存少，并发能力强，采用C语言编写，所以在性能方面有一定保证。
Nginx是一个俄罗斯人伊戈尔·赛索耶夫开发的，第一个公开版本0.1.0发布于2004年10月；截止到2017年7月11日，最新稳定版是1.12。
与Nginx同类的Web服务有IIS、Apache、Tomcat等。

NginX初探

Nginx特性

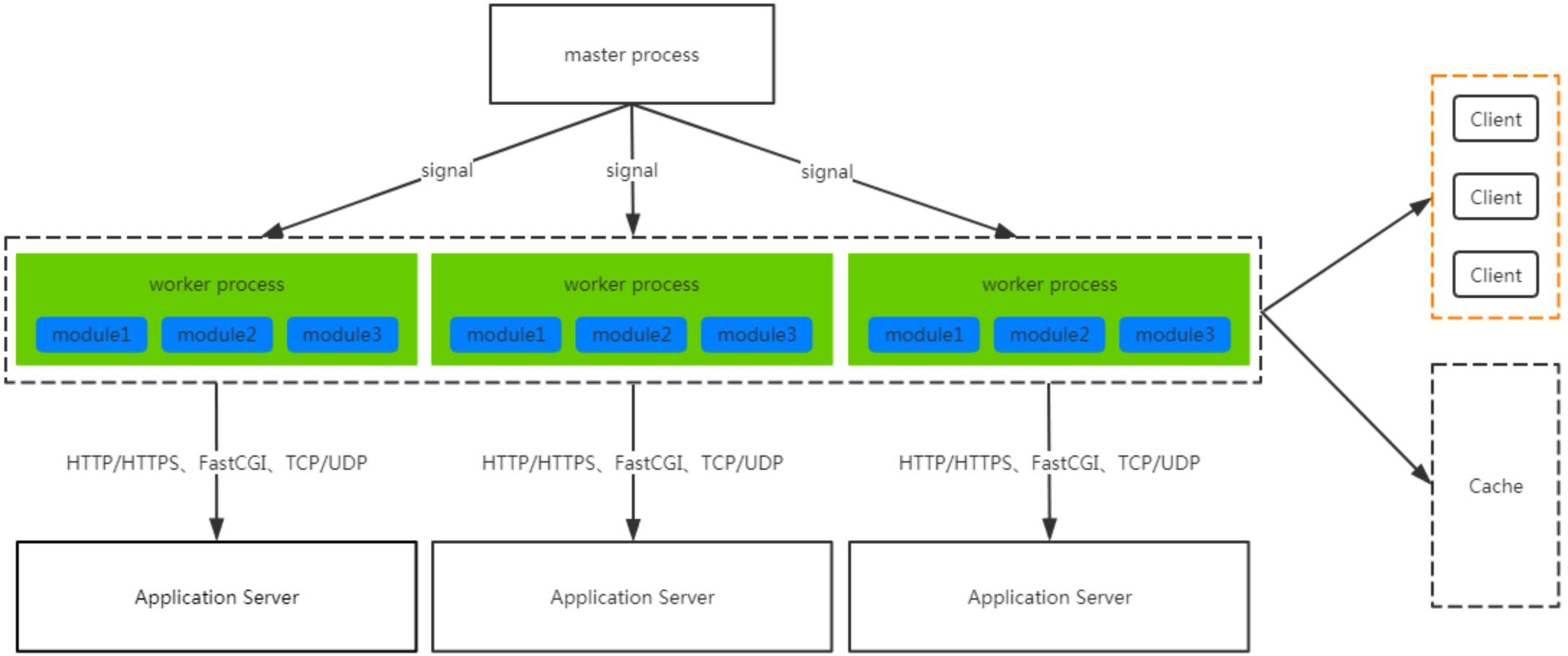
- 模块化设计
- 低内存消耗，高并发
- 事件驱动，AIO
- 高可靠性，master与worker架构
- 支持热更新配置、日志文件滚动、平滑升级
- 丰富的扩展模块

NginX初探

Nginx基本功能

- 静态资源Web服务器
- 基于域名/IP/端口的虚拟主机
- HTTP/HTTPS、SMTP、POP3和TCP/UDP反向代理
- 负载均衡
- 页面缓存
- 支持代理FastCGI、uWSGI等应用服务器
- 支持gzip、expires
- URL Rewrite
- 路径别名
- 基于IP、用户的访问控制
- 支持访问速率、并发限制
- 等…

Nginx架构


master process
signal
signal
signal
worker process
module1
module2
module3
worker process
module1
module2
module3
worker process
module1
module2
module3
Client
Client
Client
Cache
HTTP/HTTPS、FastCGI、TCP/UDP
HTTP/HTTPS、FastCGI、TCP/UDP
HTTP/HTTPS、FastCGI、TCP/UDP
Application Server
Application Server
Application Server

NginX初探

Nginx软件架构

主进程（master process）：

1. 主要与外界通信和工作进程管理；
2. 读取nginx配置文件并验证有效性；
3. 建立、绑定和关闭socket；
4. 按照配置文件生成、管理和结束工作进程；
5. nginx重启、停止、重载配置文件、平滑升级、管理日志文件等。

工作进程（worker process）：

1. 接收客户端请求，将请求交给各个功能模块处理；
2. 系统IO调用，获取响应的数据，发送响应给客户端；
3. 数据缓存管理；
4. 接收主进程指令，比如重启、关闭等。

缓存索引重建及管理进程（cache loader & cache manager）：

cache模块，主要由缓存索引重建（cache loader）和缓存索引管理（cache manager）两个进程完成，缓存索引重建进程是在进程在nginx服务启动一段时间之后（默认是1分钟）由主进程生成，对本地磁盘的索引文件在内存中建立元数据库，包括扫描、过期更新等操作，完成后退出。

NginX初探

Nginx与Apache对比

- Nginx轻量级，比Apache占用内存更少，尤其是prefork模型；
- Nginx更抗并发，单机支持10万+QPS，Nginx处理请求是异步非阻塞的，而Aapche是阻塞的；
- Nginx采用多进程工作模式，而Apache有多进程和多进程多线程两种工作模式；
- Nginx高度模块化设计，有很多丰富的模块，更好的扩展性；
- Apache历史悠久（在九几年就已经流行了），稳定性要比Nginx高；
- 采用网络I/O模型不同，Apache采用select，Nginx在Linux2.6+上采用epoll。

NginX初探

网络IO模型

简单来说，网络I/O是用户态和内核态之间的数据交换。
一次网络数据读取操作，大概是这样的：
应用进程通过系统调用（read）->由用户态转到内核态->内核将请求的数据发送到内核缓冲区->应用进程查看内核缓冲区是否有数据->如果有则把数据拷贝到用户态->完成I/O操作

- 阻塞
- 非阻塞
- 同步
- 异步
- IO多路复用

NginX初探

网络IO模型

五个 I/O 模型的比较

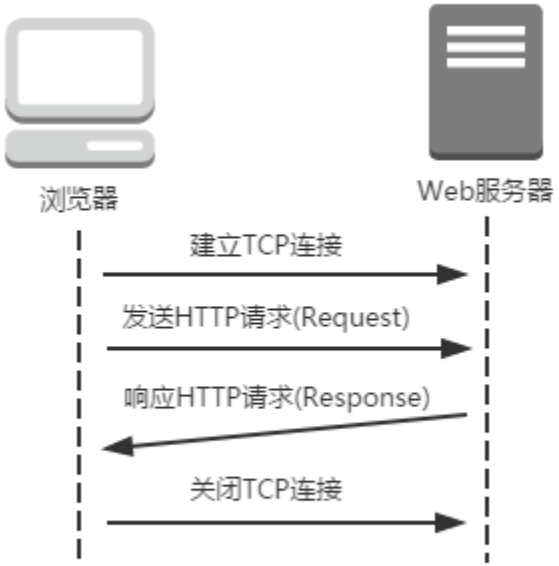
IO多路复用技术

IO多路复用技术是一种消息通知模式，基于事件驱动，常用有select、poll、epoll几种系统函数实现。

	select/poll	epoll
性能	随着连接数增加，性能急剧下降	随着连接数增加，性能很小下降
连接数	连接数有限制，处理的最大连接不超过1024，如果要修改FD_SETSIZE需要重新编译	连接数无限制
处理机制	轮训	回调

NginX初探

HTTP协议



HTTP工作流程图

1. 浏览器地址栏输入网址(URL,统一资源定位符)；
2. 浏览器DNS服务器解析到域名的真实IP；
3. 向真实IP 80端口发起TCP连接，完成三次握手；
4. 封装HTTP数据，发送HTTP请求，包括HTTP方法、资源地址、浏览器本身信息等；
5. Web服务器处理HTTP请求，返回请求资源的数据给客户端。浏览器接收到数据后渲染展示；
6. 关闭TCP连接。

Nginx安装

- CentOS与Ubuntu网络源安装
- 编译安装及编译参数
- 命令行参数

Nginx安装

CentOS网络源安装Nginx

```
# vi /etc/yum.repos.d/nginx.repo
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/OS/OSRELEASE/$basearch/
gpgcheck=0
enabled=1
# yum install nginx
# nginx -v
# nginx -V
# rpm -ql nginx  # 查看已安装包在系统安装了哪些文件
# systemctl start nginx
# ps -ef |grep nginx
```

Nginx安装

Ubuntu网络源安装Nginx

```
# curl http://nginx.org/keys/nginx_signing.key |apt-key add
# echo "deb http://nginx.org/packages/ubuntu $(lsb_release -cs) nginx
deb-src http://nginx.org/packages/ubuntu $(lsb_release -cs) nginx"
# apt-get install nginx
# nginx  -v
# nginx  -V
# dpkg -L nginx  # 查看已安装包在系统安装了哪些文件
```

Nginx编译安装及编译参数

CentOS编译安装Nginx

```
# yum install gcc pcre-devel openssl-devel
# curl -o nginx-1.12.1.tar.gz http://nginx.org/download/nginx-1.12.1.tar.gz
# tar zxvf nginx-1.12.1.tar.gz
# cd nginx-1.12.1
# ./configure --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http_ssl_module --
with-http_stub_status_module --with-stream=dynamic
# make && make install
```

Nginx编译安装及编译参数

Ubuntu编译安装Nginx

```
# apt-get install libpcre3-dev libssl-dev
# curl -o nginx-1.12.1.tar.gz http://nginx.org/download/nginx-1.12.1.tar.gz
# tar zxvf nginx-1.12.1.tar.gz
# cd nginx-1.12.1
# ./configure --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http_ssl_module --
with-http_stub_status_module --with-stream=dynamic
# make && make install
```

Nginx编译安装及编译参数

Nginx常用编译参数

参数	描述
--prefix=PATH	安装目录
--sbin-path=PATH	nginx可执行文件目录
--modules-path=PATH	模块路径
--conf-path=PATH	配置文件路径
--error-log-path=PATH	错误日志路径
--http-log-path=PATH	访问日志路径
--pid-path=PATH	pid路径
--lock-path=PATH	lock文件路径
--user=USER	运行用户
--group=GROUP	运行组
--with-threads	启用多线程
--with-http_ssl_module	提供HTTPS支持
--with-http_v2_module	HTTP2.0协议
--with-http_realip_module	获取真实客户端IP
--with-http_image_filter_module	图片过滤模块，比如缩略图、旋转等
--with-http_geoip_module	基于客户端IP获取地理位置
--with-http_sub_module	在应答数据中可替换静态页面源码内容
--with-http_dav_module	为文件和目录指定权限，限制用户对页面有不同的访问权限
--with-http_flv_module --with-http_mp4_module	支持flv、mp4流媒体播放
--with-http_gzip_static_module	针对静态文件，允许发送.gz文件扩展名的预压缩文件给客户端，使用是gzip_static on
--with-http_gunzip_static_module	gzip 用于对不支持gzip压缩的客户端使用，先解压缩后再响应。
--with-http_secure_link_module	检查链接，比如实现防盗链
--with-http_stub_status_module	获取nginx工作状态模块
--with-mail_ssl_module	启用邮件SSL模块
--with-stream	启用TCP/UDP代理模块
--add-module=PATH	启用扩展模块
--with-stream_realip_module	流形式，获取真实客户端IP
--with-stream_geoip_module	流形式，获取客户端IP地理位置
--with-pcre	启用PCRE库，rewrite需要的正则库
--with-pcre=DIR	指定PCRE库路径
--with-zlib=DIR	指定zlib库路径，gzip模块依赖
--with-openssl=DIR	指定openssl库路径，ssl模块依赖

Nginx命令行参数

Nginx命令行参数

```
# /usr/local/nginx/sbin/nginx  -h
-c file 指定配置文件
-g directives 设置全局配置指令，例如nginx -g“pid /var/run/nginx.pid”
-t 检查配置文件语法
-v 打印nginx版本
-V 打印nginx版本，编译器版本和配置
-s 发送信号
信号：
stop 快速关闭
quit 正常关闭，等待工作进程完成当前请求后停止nginx进程
reload 重新加载配置文件
reopen 重新打开日志文件
```

配置文件详解与调优

- 配置文件结构
- 配置文件指令详解
- 配置基本调优
- 常用内置变量

配置文件详解与调优

配置文件结构

nginx.conf配置文件由指令控制的模块组成，指令分为简单指令和块指令，一个简单指令由名称和参数组成，空格分隔，分号结尾。块指令与简单指令相同的结构，但不是以分号结尾，而是以大括号包围的一组附加指令结束。

块指令如下：

全局块：配置nginx全局的指令
events块：配置nginx与用户连接相关指令
http块：提供HTTP服务
server块：配置虚拟主机，一个http可以有多个server
location块：匹配URL后做什么动作

配置文件详解与调优

配置文件指令详解

以默认nginx.conf配置文件参考讲解...

配置文件详解与调优

配置基本调优

- 增加工作进程数、连接数
- 工作进程CPU绑定
- 增大打开最大文件数
- sendfile提升文件传输
- 启用文件压缩（gzip）
- 启用客户端缓存（expires）

  其他客户端缓存策略：
  - Last-Modified+If-Modified-Since
  - ETag+If-None-Match
  - Cache-Control add_header Cache-Contorl max-age=60
- 错误页面优雅显示
- 屏蔽输出版本

配置文件详解与调优

常用内置变量

变量名	描述
$remote_addr	客户端IP地址
$remote_user	客户端用户名，auth_basic认证的用户
$time_local	访问时间和时区
$request	请求的HTTP方法、URI和协议版本
$status	响应的HTTP状态码
$body_bytes_sent	响应客户端文件内容大小
$http_referer	从哪个页面链接过来的
$http_user_agent	客户端浏览器信息
$http_x_forwarded_for	客户端IP地址，只有在配置代理或负载均衡下请求头才添加该项
$host	请求头中Host字段的值
$request_filename	请求路径及文件名
$request_uri	完整的URI，包含请求参数
$uri	不包含请求参数
$http_cookie	客户端cookie
$query_string和$args	请求中的参数
$server_name	Nginx的服务器名称
$request_time	从接收用户请求的第一个字节到发送响应数据时间。
$upstream_addr	后端节点IP
$upstream_response_time	与后端建立连接到接收完响应数据时间
$upsrtream_cache_status	缓存后端节点响应数据状态，有MISS、HIT、EXPIRED、UPDATING和STALE

虚拟主机与请求处理流程

- 虚拟主机
  - 基于域名
  - 基于端口
  - 基于IP
- 请求处理流程
  - server_name指令

虚拟主机与请求处理流程

基于域名

```
server {
       listen 80;
       server_name nextdevops.cn www.nextdevops.cn;
}
server {
       listen 80;
       server_name nextdevops.com www.nextdevops.com;
}
server {
       listen 80;
       server_name nextdevops.net www.nextdevops.net;
}
```

虚拟主机与请求处理流程

基于端口

```
server {
       listen 81;
       server_name localhost;
}
server {
       listen 82;
       server_name localhost;
}
server {
       listen 83;
       server_name localhost;
}
```

虚拟主机与请求处理流程

基于IP地址

```
server {
       listen 192.168.1.101:80;
       server_name localhost;
}
server {
       listen 192.168.1.102:80;
       server_name localhost;
}
server {
       listen 192.168.1.103:80;
       server_name localhost;
}
```

虚拟主机与请求处理流程

基于名称与IP地址

```
server {
       listen 192.168.1.100:80;
       server_name nextdevops.cn www.nextdevops.cn;
}
server {
       listen 192.168.1.100:80;
       server_name nextdevops.com www.nextdevops.com;
}
server {
       listen 192.168.1.101:80;
       server_name nextdevops.cn www.nextdevops.cn;
}
```

虚拟主机与请求处理流程

server_name指令

1、精确名称

例如：server_name nextdevops.cn www.nextdevops.cn;

2、通配符

例如：

server_name *.nextdevops.cn www.nextdevops.cn;

server_name mail.*;

3、正则表达式

例如：

server_name "~^www\d+\.nextdevops\.cn$";

server_name "~^\d{1,3}\.nextdevops\.cn$";

server_name "~^www\.(?P<domain>.+)$";

匹配优先级：精确名称->开头通配符名称->结尾通配符名称->正则表达式

虚拟主机与请求处理流程

请求处理流程小结

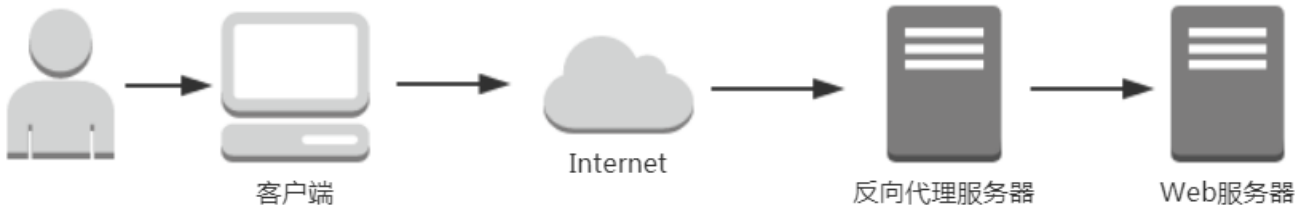
监听指令判断请求的IP地址和端口->server_name指令->默认虚拟主机

反向代理

- 正向代理与反向代理概念
- FastCGI代理（php-fpm）
- FastCGI代理配置优化（fastcgi_cache）
- HTTP代理（Tomcat）
- HTTP代理配置优化（proxy_cache）
- TCP与UDP代理

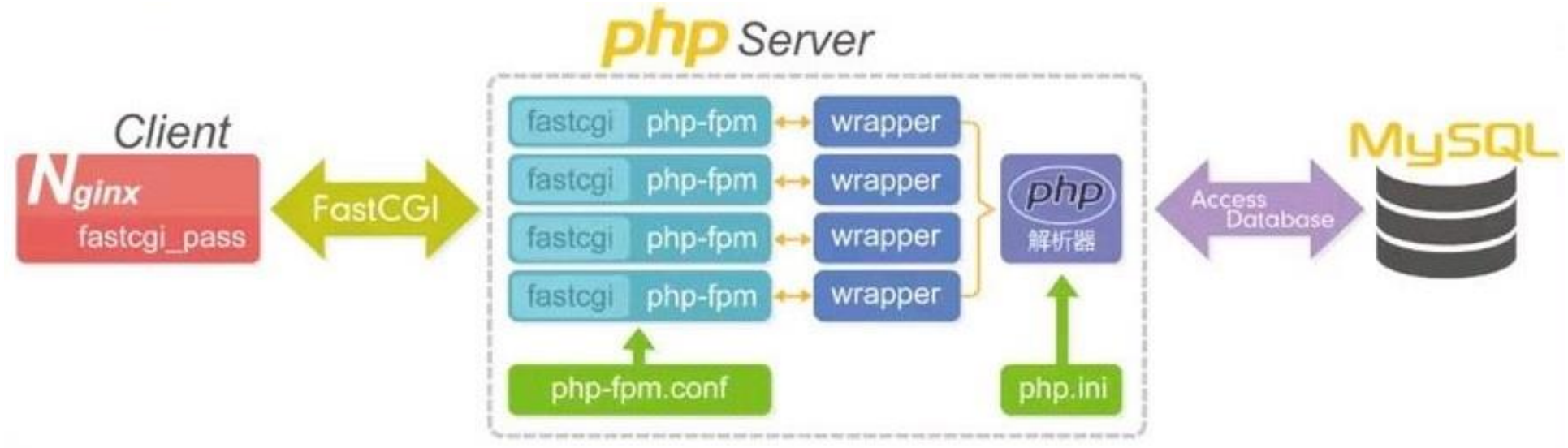
反向代理

正向代理与反向代理概念



反向代理

FastCGI代理（php-fpm）



Nginx+FastCGI+PHP工作流程图

反向代理

FastCGI代理（php-fpm）

1、安装php依赖的第三方库

```
yum install gd-devel libxml2-devel libcurl-devel libjpeg-devel libpng-devel
```

2、编译安装php

```
# 编译
tar zxvf php-5.6.31.tar.gz
./configure --prefix=/usr/local/php \
--with-config-file-path=/usr/local/php/etc \
--with-mysql --with-mysqli \
--with-openssl --with-zlib --with-curl --with-gd \
--with-jpeg-dir --with-png-dir --with-iconv \
--enable-fpm --enable-zip --enable-mbstring
make -j 4 && make install
# 配置php
cp php.ini-production /usr/local/php/etc/php.ini
vi /usr/local/php/etc/php.ini
date.timezone = Asia/Shanghai
```

3、配置php-fpm

```
cp /usr/local/php/etc/php-fpm.conf.default /usr/local/php/etc/php-fpm.conf
vi /usr/local/php/etc/php-fpm.conf
user = nginx
group = nginx
pid = run/php-fpm.pid
cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm
chmod +x /etc/rc.d/init.d/php-fpm
service php-fpm start
```

4、Nginx配置php程序通过FastCGI转发给php-fpm

```
server {
    listen       80;
    server_name  localhost;
    access_log  logs/wp.access.log  main;
    location / {
        root   html/wordpress;
        index  index.php index.html index.htm;
    }
    location ~ \.php$ {
        root           html/wordpress;
        fastcgi_pass   127.0.0.1:9000;
        fastcgi_index  index.php;
        fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
        include        fastcgi_params;
    }
}
```

反向代理

FastCGI代理配置优化

```
http {
 ......
    fastcgi_cache_path /usr/local/nginx/fastcgi_cache levels=1:2 keys_zone=fastcgi_cache_zone:128m inactive=5m max_size=10g;
    # 缓存目录 目录层级 缓存区名称和大小 移除多长时间未访问的缓存数据 最大占用磁盘空间

    fastcgi_buffering on;                # 默认on，是否缓存FastCGI响应
    fastcgi_buffer_size 64k;             # 缓存区大小
    fastcgi_buffers 8 32k;               # 指定多少与多大缓存区来缓存FastCGI响应
    fastcgi_temp_path fastcgi_temp 1 2;  # 默认目录fastcgi_temp
    fastcgi_max_temp_file_size 1024m;    # 默认1024m，单个临时文件最大大小
    fastcgi_temp_file_write_size 128k;   # 一次写入临时文件数据大小
    fastcgi_request_buffering on;        # 默认on，是否先缓存整个客户端请求正文再发送FastCGI服务器
    fastcgi_connect_timeout 60s;         # 默认60s，与FastCGI服务器建立连接超时时间
    fastcgi_read_timeout 300s;           # 默认60s，读取FastCGI服务器响应超时时间
    fastcgi_send_timeout 300s;           # 默认60s，发送请求到FastCGI服务器超时时间

    server {
        listen 80;
        server_name localhost;
        location / {
            root html/wordpress;
            index index.php index.html;
        }
        location ~ \.php$ {
            root           html/wordpress;
            fastcgi_pass   127.0.0.1:9000;
            fastcgi_index  index.php;
            fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
            include        fastcgi_params;

            fastcgi_cache fastcgi_cache_zone;     # 指定缓存区名称
            fastcgi_cache_key $host$request_uri;  # 定义缓存的key，根据md5值为缓存文件名
            fastcgi_cache_valid 200 302 10m;      # 为不同状态码设置缓存时间
            fastcgi_cache_valid 301 1d;
            fastcgi_cache_valid any 1m;
        }
    }
}
```

反向代理

第三方清理缓存模块

```
# 重新编译Nginx，添加清理缓存模块
git clone https://github.com/FRiCKLE/ngx_cache_purge
/usr/local/nginx/sbin/nginx  -V
cd nginx-1.12.1
./configure --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http_ssl_module --with-
http_stub_status_module --with-stream=dynamic --add-module=/usr/local/nginx/ngx_cache_purge
pkill nginx
cp objs/nginx /usr/local/nginx/sbin/nginx
/usr/local/nginx/sbin/nginx

# 添加清理缓存功能
location ~ /purge(.*) {
allow 127.0.0.1;
deny all;
    proxy_cache_purge proxy_cache $host$1;
    access_log logs/cache.log main;
}
```

反向代理

FastCGI代理配置优化

```
[global]
pid = run/php-fpm.pid
[www]
user = nginx
group = nginx
listen = 127.0.0.1:9000
pm = dynamic
pm.max_children = 5             # 子进程最大数量
pm.start_servers = 2            # 在启动时创建的子进程数量
pm.min_spare_servers = 1        # 空闲子进程的最小数量
pm.max_spare_servers = 3        # 空闲子进程的最大数量
pm.process_idle_timeout = 10s  # 一个空闲进程空闲多少秒后被杀死
pm.max_requests = 500  每个子进程处理完多少请求之后自动重启，为防止PHP第三方库内存泄露问题。设置0则不自动重启，会一直处理请求。
```

反向代理

FastCGI代理配置优化

```
# vi /usr/local/php/etc/php-fpm.conf
listen = /usr/local/php/php-fcgi.sock;
listen.owner = nginx
listen.group = nginx

# Nginx配置
location ~ \.php$ {
    root           html/wordpress;
    fastcgi_pass   unix:/usr/local/php/php-fcgi.sock;
    fastcgi_index  index.php;
    fastcgi_param  SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include        fastcgi_params;
}
```

反向代理

HTTP代理（Tomcat）

```
server {
    listen 80;
    server_name  localhost;
    access_log  logs/test_access.log;

    location / {
        proxy_pass  http://127.0.0.1:8080;
        proxy_set_header   Host             $host;
        proxy_set_header   X-Real-IP        $remote_addr;
        proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
    }
}
```

反向代理

HTTP代理配置优化（proxy_cache）

```
http {
    ……
    proxy_cache_path /usr/local/nginx/proxy_cache levels=1:2 keys_zone=proxy_cache_zone:128m inactive=5m max_size=10g;
    proxy_buffering on;                 # 默认on，是否缓存后端服务器响应
    proxy_buffer_size 64k;              # 缓存区大小
    proxy_buffers 8 32k;                # 指定多少与多大缓存区来缓存后端服务器响应
    proxy_temp_path proxy_temp 1 2;     # 默认目录proxy_temp
    proxy_max_temp_file_size 1024m;     # 默认1024m，单个临时文件最大大小
    proxy_temp_file_write_size 128k;    # 默认16k，一次写入临时文件数据大小
    proxy_request_buffering on;         # 默认on，是否先缓存整个客户端请求正文再发送后端服务器
    proxy_ignore_headers Set-Cookie;    # 忽略缓存cookie

    proxy_set_header Host $host;  # 添加请求头Host字段值为本机IP地址
    proxy_set_header X-Real-IP $remote_addr;  # 添加请求头X-Real-IP值为客户端IP
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  # 原始客户端IP和代理IP地址

    proxy_connect_timeout 60s;          # 默认60s，与后端服务器建立连接超时时间
    proxy_read_timeout 300s;            # 默认60s，读取后端服务器响应超时时间
    proxy_send_timeout 300s;            # 默认60s，发送请求到后端服务器超时时间
    server {
        listen 88;
        server_name localhost;
        # 动态资源不做缓存
        location / {
            proxy_pass http://192.168.1.100:8080;
        }
        # 只缓存静态文件
        location ~ \.(html|css|js|jpg|png|gif)$ {
            root /usr/local/apache-tomcat-8.0.45/webapps/ROOT;
            proxy_cache proxy_cache_zone;        # 指定缓存区名称
            proxy_cache_key $host$request_uri;   # 定义缓存的key，根据md5值为缓存文件名
            proxy_cache_valid 200 302 10m;       # 为不同状态码设置缓存时间
            proxy_cache_valid 301 1d;
            proxy_cache_valid any 1m;
            # add_header X-Cache $upstream_cache_status;  # 添加响应头，测试是否命中;代理服务器才有的变量
        }
    }
}
```

反向代理

TCP与UDP代理

```
stream {
    server {
        listen  88;
        proxy_connect_timeout 1s;
        proxy_timeout 1m;
        proxy_pass 192.168.1.100:8080;
    }
    server {
        listen  89;
        proxy_connect_timeout 1s;
        proxy_timeout 1m;
        proxy_pass 192.168.1.100:80;
    }
    server {
        listen 53 udp;
        proxy_responses 1;
        proxy_timeout 20s;
        proxy_pass 192.168.1.100:53;
    }
    # UNIX域套接字
    server {
        listen 12345;
        proxy_pass unix:/tmp/stream.socket;
    }
}
```

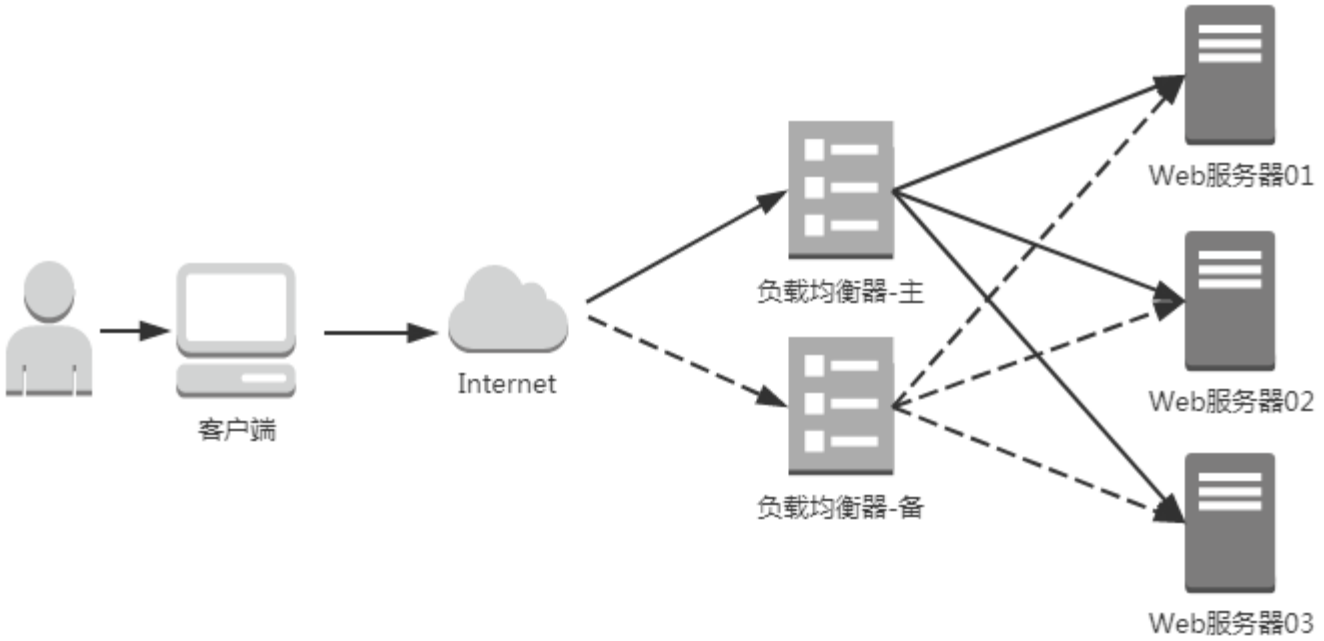
负载均衡

- 负载均衡是什么
- upstream块
- upstream调度算法
- upstream内置变量

负载均衡

负载均衡是什么

是一个把网络请求转发到一组服务器中可用的服务器上的设备。
负载均衡器实现分为两种：
硬件：F5、NetScaler
软件：LVS、Nginx、HAProxy
负载均衡器技术原理上又分为：
四层：数据包解封装到传输层，获取到IP和端口，然后转发。
七层：数据包解封装到应用层，获取到应用层数据，然后分析转发。



负载均衡

upstream块

语法：

```
upstream name {
    server address [parameters];
    ...
}
```

HTTP示例：

```
upstream http_backend {
    server backend1.example.com weight=5;
    server backend2.example.com:8080 max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;
    server backup1.example.com:8080 backup;
}
server {
    listen 80;
    server_name localhost;
    location / {
        proxy_pass http://http_backend;
    }
}
```

负载均衡

upstream块

参数:

```
weight=number          # 设置服务器权重，默认为1
max_conns=number       # 限制代理服务器同时活动的最大连接数（1.11.5），默认为0，没限制
max_fails=number       # 在fail_timeout参数持续时间内尝试与服务器通信的失败最大次数，默认为1,0为禁用
fail_timeout=time      # 在这个时间内max_fails尝试连接最大失败次数，以及服务器不可用时间，默认10s
backup                 # 将服务器标记为备份服务器，当主服务器不可用时，它将接受请求
down                   # 将服务器标记为永久不可用
```

负载均衡

upstream块

TCP/UDP示例：

```
upstream tcp_backend {
    server backend1.example.com weight=5;
    server backend2.example.com:8080 max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;
    server backup1.example.com:8080 backup;
}
server {
    listen 12345;
    location / {
        proxy_pass tcp_backend;
    }
}
```

负载均衡

upstream块

指令：

```
keepalive connections  # 激活连接缓存池，每个工作进程与上游服务器保持的最大空闲连接数
```

HTTP示例：

```
upstream http_backend {
    server 127.0.0.1:8080;
    keepalive 16;
}
server {
    ...
    location / {
        proxy_pass http://http_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        ...
    }
}
```

FastCGI示例：

```
upstream fastcgi_backend {
    server 127.0.0.1:9000;
    keepalive 8;
}
server {
    ...
    location / {
        fastcgi_pass fastcgi_backend;
        fastcgi_keep_conn on;
        ...
    }
}
```

负载均衡

upstream块

指令：

proxy_next_upstream　# 指定以下哪种情况下，将请求转发到下一个服务器，默认error和timeout

error　　# 与服务器建立连接，向其发送请求或读取响应头时发生错误

timeout　　# 与服务器建立连接，向其发送请求或读取响应头时发生超时

invalid_header　# 服务器返回空或无效的响应

http_500、http_502、http_504、http_403、http_404 # 服务器响应状态码

off　# 禁止向下一个服务器转发请求

负载均衡

upstream调度算法

ngx_http_upstream_module支持以下调度算法：

轮训　　# 默认，以循环方式向上游服务器分发请求

least_conn　# 最少连接，下一个请求被分配到最小活动连接的上游服务器，同时考虑权重

ip_hash　　# 基于客户端IP地址做哈希，将同一哈希值的请求分配到上一次分配的上游服务器

hash key [consistent] # 自定义key做哈希，该key可以是文本、变量以及它们组合；consistent是更改使用ketama一致散列算法。

ngx_stream_upstream_module模块只支持hash key和least_conn;

upstream内置变量

ngx_http_upstream_module模块内置变量：

```
$upstream_addr     # 上游服务器IP地址和端口或UNIX域套接字的路径。如果在请求处理期间连接了多个服务器，则他们地址用逗号分隔，例如“192.168.1.2:12345, 192.168.1.3:12345, unix:/tmp/sock” 。
$upstream_bytes_received # 从上游服务器接收的字节数（1.11.4），多个连接的值用逗号分隔。
$upstream_cache_status   # 访问响应缓存的状态。有MISS、BYPASS、EXPIRED、STALE、UPDATING、REVALIDATED和HIT。
$upstream_connect_time   # 与上游服务器建立连接所需时间（1.9.1），在SSL情况下，包括握手上花费的时间。
$upstream_cookie_name    # 由上游服务器在Set-Cookie响应头中发送（1.7.1），只保持最后一台服务器响应的cookie
$upstream_header_time    # 与上游服务器接收响应报头时花费时间（1.7.10）。
$upstream_response_length   # 从上游服务器获得响应长度，单位字节，多个响应长度用逗号分隔。
$upstream_response_time  # 接收上游服务器的响应时花费时间。
$upstream_status  # 从上游服务器获取的响应状态码。多个响应状态码用逗号分隔。
```

ngx_stream_upstream_module模块内置变量（1.11.4）：

```
$upstream_addr           # 上游服务器IP地址和端口或UNIX域套接字的路径。如果在代理期间连接了多个服务器，用逗号分隔，例如"192.168.1.2:12345, 192.168.1.3:12345, unix:/tmp/sock"。
$upstream_bytes_sent     # 发送到上游服务器的字节数。多个连接的值用逗号分隔。
$upstream_bytes_received # 从上游服务器接收的字节数。多个连接的值用逗号分隔。
$upstream_connect_time   # 连接上游服务器的时间; 单位毫秒，多个连接的时间用逗号分隔。
$upstream_first_byte_time  # 接收数据的第一个字节时间。几个连接的时间由逗号分隔。
$upstream_session_time   # 会话保持时间（以秒为单位）。
```

location块匹配规则

- 表达式类型
- 表达式优先级

location块匹配规则

表达式类型

~	表示正则匹配，区分大小写
~*	表示正则匹配，不区分大小写
^~	普通字符前缀匹配，如果匹配成功则不再继续匹配
=	普通字符精确匹配

location块匹配规则

表达式优先级

第一优先级：“=” 精确匹配，一旦匹配成功，不再继续匹配
第二优先级：“^~”普通字符匹配，一旦匹配成功，不再继续匹配
第三优先级：“~”和“~*”正则表达式，如果多个location正则匹配，优先匹配最长
第四优先级：常规字符串匹配

location块匹配规则

表达式优先级

```
server {
    listen 80;
    server_name localhost;
    location / {
        return 410;
    }
    location = / {
        return 411;
    }
    location = /index.html {
        return 412;
    }
    location /name/ {
        return 413;
    }
    location ^~ /images/ {
        return 414;
    }
    location ~ \.(jpg|png|gif)$ {
        return 415;
    }
    location ~* \.jpg$ {
        return 416;
    }
}
```

URL重定向及其他常用指令

- rewrite指令
- return指令
- set指令
- if指令
- 常用正则表达式符号

URL重定向及其他常用指令

rewrite指令

rewrite：匹配URI，根据定义的规则对其重写或改变。

语法：

rewrite regex replacement [flag]

regex：正则表达式匹配请求的URI

replacement：替换后的URI或URL；如果替换字符串以http、https或$scheme开头，则匹配终止，并返回客户端

flag：标志，参数如下：

last # 停止处理后面rewrite指令，并用替换后的URI重新发起一次请求，再一次匹配location

break # 停止处理后面rewrite指令

redirect # 临时重定向，返回302状态码

permanent # 永久重定向，返回301状态码

URL重定向及其他常用指令

return指令

return：停止处理并返回指定状态码给客户端。

语法：

```
return code [text]
return code URL
return URL
```

参数说明：

```
code  # HTTP状态码
text  # 响应正文
URL   # 临时重定向地址
```

URL重定向及其他常用指令

set指令

set：设置变量

语法：

```
set $variable value
```

URL重定向及其他常用指令

if指令

if：条件判断

语法：

if (condition) {…}

condition可以是以下任何一种：

a. 一个变量名，如果变量的值为空或0，则为false。
b. 使用”=”和”!= ”运算符比较变量与字符。
c. 使用”~”（区分大小写匹配）和” ~*”（不区分大小写匹配）运算符，将变量与正则表达式进行匹配。正则表达式可以是分组匹配，使用$1…$9引用捕获的值。也可以用” !~”和”!~*”取反。如果正则表达式包含” }”或” ;”字符，则整个表达式用单引号或双引号括起来。
d. 使用” -f”和” !-f”操作符检查文件存在。
e. 使用” -d”和” !-d”操作符检查目录存在。
f. 使用” -e”和” !-d”操作符检查文件、目录或符号链接存在。
g. 使用” -x”和” !-x”运算符检查可执行文件。

URL重定向及其他常用指令

常用正则表达式符号

```
.       # 匹配除换行符（\n）以外的任意单个字符
*       # 匹配字符0个或多个
+       # 匹配字符1个或多个
?       # 匹配字符0个或1个
^       # 匹配后面字符开头
$       # 匹配前面字符结尾
{n}     # 匹配花括号前面字符至少n个字符
{n,m}   # 匹配花括号前面字符至少n个字符，最多m个字符
[ ]     # 匹配中括号中的任意一个字符
[a-z]   # 匹配a-z范围内的任意一个字母
[0-9]   # 匹配0-9范围内的任意一个数字
|       # 匹配竖杠两边的任意一个
( )     # 分组匹配，通过$1...$9反向引用
\       # 转义符，将特殊符号转成原有意义
\d      # 匹配数字，等效[0-9]
```

安全配置

- HTTPS
- 防盗链
- 访问控制
- 限流
- Nginx平滑升级

安全配置

HTTPS

- HTTPS是什么
- 为什么要用HTTPS
- HTTPS工作原理
- OpenSSL自签证书
- Nginx配置HTTPS及优化
- SNI是什么

安全配置

HTTPS

HTTPS是什么？

HTTPS（Hyper Text Transfer Protocol over Secure Socket Layer），是以安全为目的的HTTP通道，简单讲是HTTP的安全版。

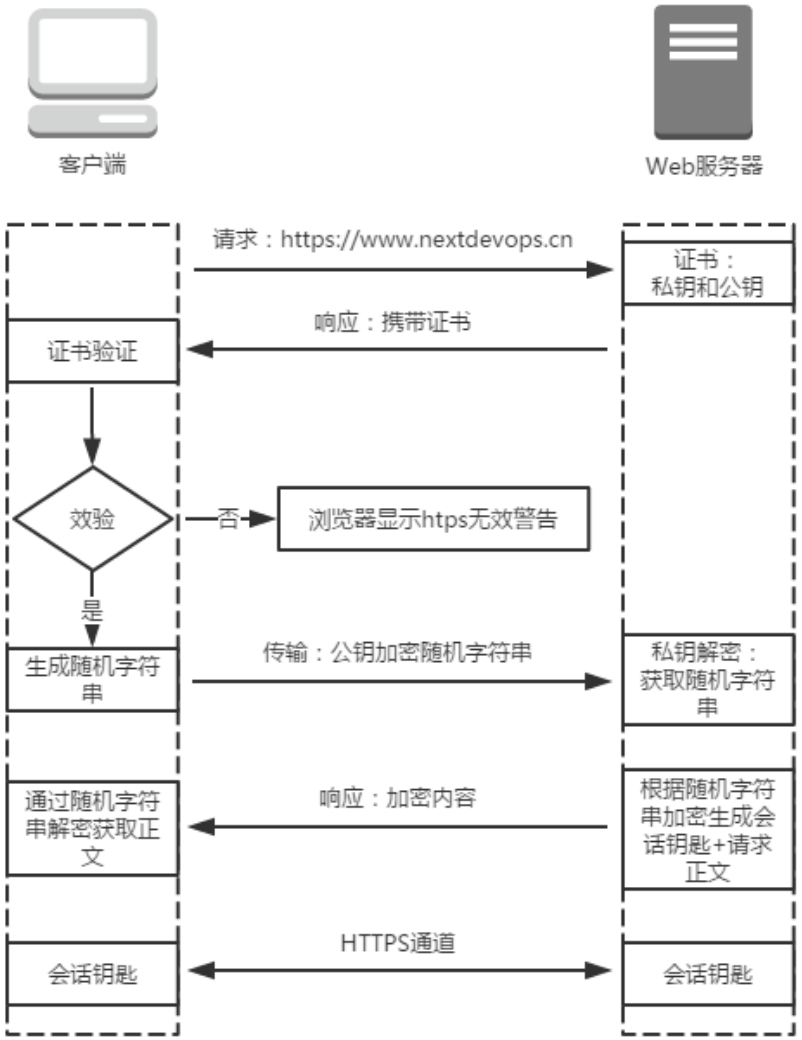
HTTPS由两部分组成：HTTP+SSL/TLS

TLS前身是SSL，TLS1.0对应SSL3.1，TLS1.1对应SSL3.2，TLS1.2对应SSL3.3。

为什么要用HTTPS？

主要防止数据泄密和篡改。

安全配置



HTTPS

1. 浏览器向服务器443端口发送HTTP请求；
2. 服务器收到请求将数字证书返回给浏览器；
3. 浏览器验证证书是否有效，如果不可信任，提示https无效警告；如果证书可信任，则取出里面公钥生成一个随机值，用这个公钥对随机值加密，然后发送给服务器；
4. 服务器收到数据后，用私钥解密，得到随机值，然后生成会话密钥，并把请求的内容与会话密钥一同返回给浏览器；
5. 浏览器收到数据后用之前的私钥解密，获得网页内容并展示；
6. 客户端与服务器通信则用这个会话钥匙进行加解密。

安全配置

HTTPS

OpenSSL自签证书

1、创建CA证书

```
  openssl req \
    -newkey rsa:4096 -nodes -sha256 -keyout ca.key \
    -x509 -days 365 -out ca.crt
```

2、创建一个证书签名请求

```
  openssl req \
    -newkey rsa:4096 -nodes -sha256 -keyout yourdomain.com.key \
    -out yourdomain.com.csr
```

3、创建域名证书

```
  openssl x509 -req -days 365 -in yourdomain.com.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
yourdomain.com.crt
```

安全配置

HTTPS

Nginx配置HTTPS

```
server {
   listen 443 ssl;
   server_name www.nextdevops.cn;
   ssl_certificate ../crt/www.nextdevops.cn.crt;
   ssl_certificate_key ../crt/www.nextdevops.cn.key;
   location / {
       index index.html;
       root /opt/test;
   }
}
```

安全配置

HTTPS

Nginx对HTTPS配置优化

```
http {
    ssl_session_cache   shared:SSL:10m;
    ssl_session_timeout 10m;
    server {
        listen              443 ssl;
        server_name         www.example.com;
        keepalive_timeout   70;

        ssl_certificate      www.example.com.crt;
        ssl_certificate_key  www.example.com.key;
        ssl_protocols        TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers          HIGH:!aNULL:!MD5;
        ...
    }
}
```

安全配置

HTTPS

SNI是什么？

在发起SSL握手请求时，允许客户端携带Host信息。

安全配置

防盗链

- referer
- secure_link模块

安全配置

防盗链

referer

ngx_http_referer_module模块用于在referer请求头字段阻止无效的值请求访问。

示例：

```
location ~* \.(gif|jpg|png|mp4)$ {
    valid_referers none blocked server_names
                   *.nextdevops.cn nextdevops.* www.example.org/images/
                   ~\.google\. ~\.baidu\.;

    if ($invalid_referer) {
        return 403;
    }
}
```

安全配置

防盗链

secure_link模块

ngx_http_secure_link_modle模块用于检测请求连接的真伪，保护资源未授权访问，并限制连接。

该模块提供了两种操作模式：
a） secure_link和secure_link_md5指令启用，定义URL传参的变量，从中提取md5值及有效期进行比对。
b） secure_link_secret指令启用，检查请求连接的真实性。

示例1：

```
location /download/ {
    secure_link $arg_md5,$arg_expires;
    secure_link_md5 "$secure_link_expires$uri secret";
    if ($secure_link = "") {
        return 403;
    }
    if ($secure_link = "0") {
        return 410;
    }
    ...
}
```

安全配置

防盗链

secure_link模块

示例2：

```
location /p/ {
    secure_link_secret secret;

    if ($secure_link = "") {
        return 403;
    }

    rewrite ^ /secure/$secure_link;
}

location /secure/ {
    root html;
    internal;
}
```

安全配置

访问控制

- IP白名单
- HTTP身份认证

安全配置

访问控制

IP白名单

示例：

```
location / {
    deny  192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    deny  all;
}
```

安全配置

访问控制

HTTP身份认证

示例：

```
server {
    listen 88;
    server_name localhost;
    index index.html;
    root /opt/test;
    location / {
        auth_basic "Please enter user name and password";
        auth_basic_user_file ../conf/passwd.db;
    }
}
```

安全配置

限流

- limit_conn
- limit_req
- 压力测试

安全配置

限流

limit_conn

ngx_http_limit_conn_module模块用于限制每个定义的key的连接数，主要是单个IP地址的并发连接数。

示例：

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;
    limit_conn_log_level error;
    limit_conn_status 503;
    ...
    server {
        ...
        location /limit {
            limit_conn addr 1;
        }
    }
}
```

安全配置

限流

limit_req

ngx_http_limit_req_module模块用于限制每个定义的key请求处理速率，主要是从一个单一的IP地址请求的处理速率。

示例：

```
http {
    limit_req_zone $binary_remote_addr zone=qps:10m rate=1r/s;
    limit_conn_log_level error;
    limit_conn_status 503;
    ...
    server {
        ...
        location /search/ {
            limit_req zone=qps burst=5;
            ...
        }
    }
}
```

安全配置

限流

压力测试

采用ab压力测试工具分别验证limit_conn和limit_req。

安全配置

Nginx平滑升级

1）编译新版本

```
cd nginx-1.13.4
./configure --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http_ssl_module --with-
http_realip_module --with-http_geoip_module --with-http_sub_module --with-stream=dynamic --with-
http_stub_status_module
make
```

2）重命名执行文件

```
mv /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.old
```

3）复制编译的执行文件

```
cp objs/nginx /usr/local/nginx/sbin/nginx
```

4）平滑生效可执行文件

```
kill -USR2 `cat /usr/local/nginx/logs/nginx.pid`
```

5）正常关闭老进程

```
kill -QUIT `cat /usr/local/nginx/logs/nginx.pid.oldbin`
```

其他常用模块

- GeoIP
- ImageFilter
- nginx-lua

其他常用模块

下载地址

lua第三方模块

https://github.com/openresty/lua-nginx-module/archive/v0.10.10.tar.gz

IP对应国家数据库

http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz

IP对应地区数据库

http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz

其他常用模块

编译安装

安装依赖库：

```
yum install lua-devel GeoIP-devel  -y
```

编译参数：

```
./configure --prefix=/usr/local/nginx --user=nginx --group=nginx --with-http_ssl_module --with-http_stub_status_module --with-stream=dynamic  --with-http_secure_link_module --with-http_image_filter_module --with-http_geoip_module --add-module=../lua-nginx-module-0.10.10
```

其他常用模块

GeoIP

示例：

```
gunzip GeoIP.dat.gz
gunzip GeoLiteCity.dat.gz
mkdir /usr/local/nginx/geoip
mv GeoIP.dat  GeoLiteCity.dat /usr/local/nginx/geoip
http {
        ...
    geoip_country /usr/local/nginx/geoip/GeoIP.dat;
    geoip_city /usr/local/nginx/geoip/GeoLiteCity.dat;
    log_format  main  '$geoip_city_country_name $geoip_region_name $geoip_city - '
                      '$remote_addr - $remote_user [$time_local] "$request" '
                     '$status $body_bytes_sent "$http_referer" '
                     '"$http_user_agent" "$http_x_forwarded_for"';
        ...
}
```

其他常用模块

ImageFilter

示例：

```
location /img/ {
    root html;
    image_filter resize 150 100;
    image_filter rotate 90;
    error_page   415 = /empty;
}
location = /empty {
    empty_gif;
}
```

其他常用模块

ImageFilter

动态生成缩略图示例：

```
server {
    listen 80;
    server_name localhost;
    root html;
    index index.html;
    location ~* /img/.*_(\d+)x(\d+)\.(jpg|png|gif)$ {
        root html;
        set $img_width $1;
        set $img_height $2;
        image_filter resize $img_width $img_height;
        rewrite (.*)_.*(\..*)  $1$2 break;
    }
}
```

其他常用模块

Lua

环境测试示例：

```
server {
    listen 80;
    server_name localhost;
    root html;
    index index.html;
    location /lua {
        default_type text/plain;
        content_by_lua_file /usr/local/nginx/conf/test.lua;
    }
}
# cat test.lua
local headers = ngx.req.get_headers()
ngx.say(headers.HOST)
```

Nginx高可用性（HA）

- Keepalived高可用软件介绍
- Nginx主备
- Nginx双主

Nginx高可用性（HA）

Keepalived高可用软件介绍

Keepalived是一个可以快速构建高可用服务的解决方案。设计之初是针对LVS负载均衡提供高可用的，它集成对LVS集群管理，包括健康检查、故障剔除等功能。

Keepalived使用VRRP协议实现主备模式，当主服务器发生故障，备服务器接管。

工作原理：

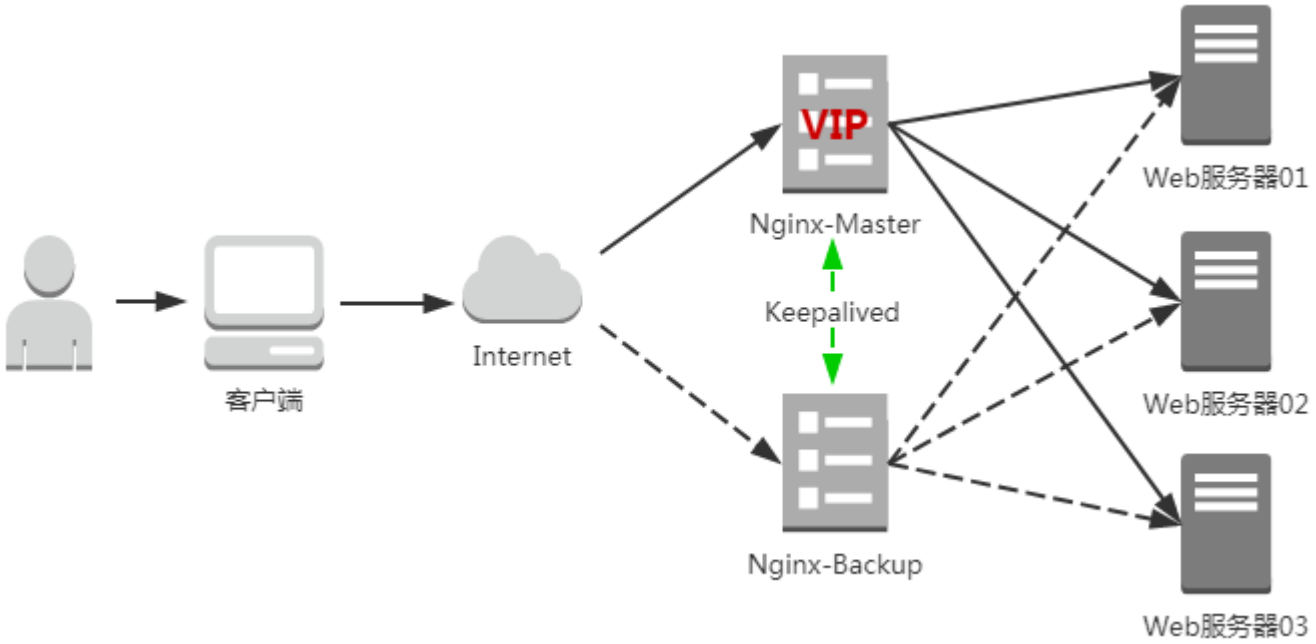
VRRP实例中分为MASTER和BACKUP状态，组成一个热备组，MASTER状态及优先级高的服务器绑定一个虚拟IP（VIP），这个VIP对外提供服务。

热备组内服务器周期性发送VRRP通告信息，一方面BACKUP服务器确定MASTER是否存活，另一方面进行MASTER选举。如果MASTER宕机，BACKUP切换到MASTER状态，接管VIP，对外提供服务；当MASTER恢复后会自动加入热备组切换到MASTER状态，接管VIP，对外提供服务。

Nginx高可用性（HA）

Nginx主备

Ngixn主对外提供服务器，备处于空闲状态。



Nginx高可用性（HA）

Nginx主备

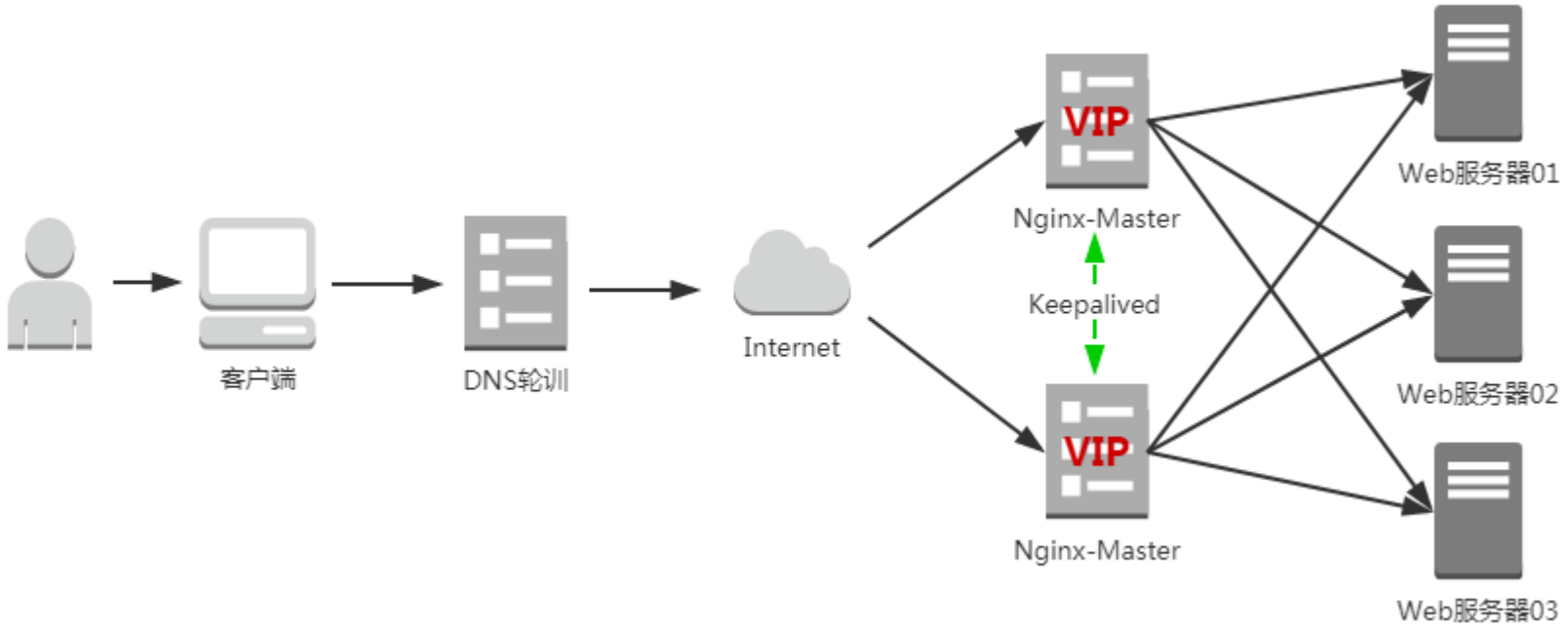
```
global_defs {
    ...
}
vrrp_script check_nginx {
    script "[[ -f /usr/local/nginx/logs/nginx.pid ]] && exit 0 || exit 1"
    interval 1
    weight -20
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
        check_nginx
    }
    virtual_ipaddress {
        192.168.1.191/24
    }
}
```

Nginx高可用性（HA）

Nginx双主

两台Nginx互为主备，DNS解析两个A记录到对应VIP，同时对外提供服务。



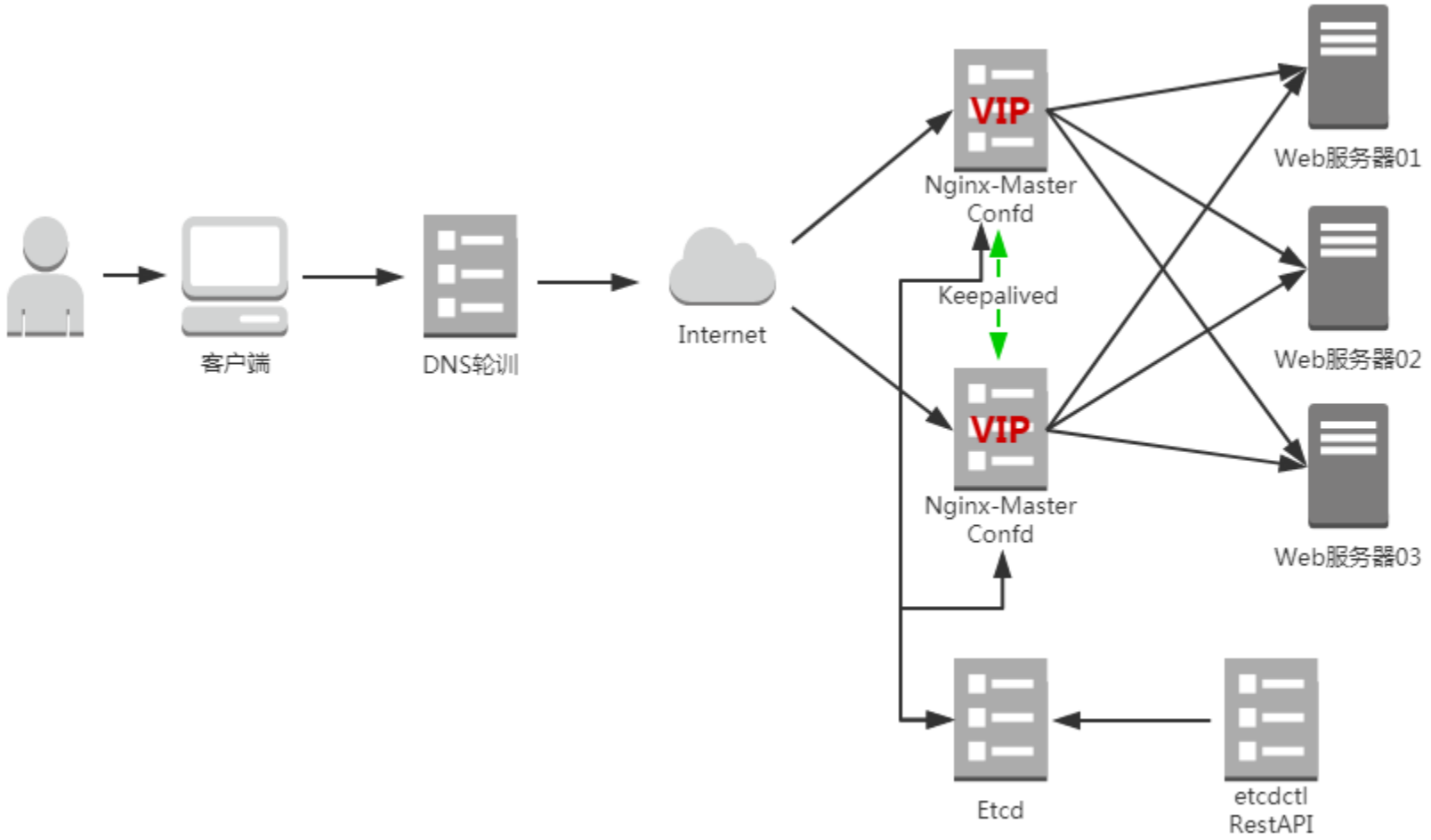
Nginx高可用性（HA）

Nginx双主

```
global_defs {
    ...
}
vrrp_script check_nginx {
    script "[[ -f /usr/local/nginx/logs/nginx.pid ]] && exit 0 || exit 1"
    interval 1
    weight -20
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
        check_nginx
    }
    virtual_ipaddress {
        192.168.1.191/24
    }
}
vrrp_instance VI_2 {
    state BACKUP
    interface eth0
    virtual_router_id 52
    priority 90
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
        check_nginx
    }
    virtual_ipaddress {
        192.168.1.192/24
    }
}
```

Nginx集群节点自动加入

架构图



Nginx集群节点自动加入

组件介绍

etcd：分布式KV存储系统，一般用于共享配置和服务注册与发现。是CoreOS公司发起的一个开源项目。 ETCD存储格式类似于文件系统，以根"/"开始下面一级级目录，最后一个是Key，一个key对应一个Value。
confd：管理本地应用配置文件，使用etcd或consul存储的数据渲染模板，还支持redis、zookeeper。
confd有一个watch功能，通过HTTP API定期监测对应的etcd中目录变化，获取最新的Value，然后渲染模板，更新配置文件。

下载地址：
https://github.com/coreos/etcd/releases/download/v3.1.4/etcd-v3.1.4-linux-amd64.tar.gz
https://github.com/kelseyhightower/confd/releases/download/v0.11.0/confd-0.11.0-linux-amd64

https://pan.baidu.com/s/1c1M9kBm

Nginx集群节点自动加入

安装部署

Etcd安装并启动：

```
# tar zxvf etcd-v3.1.4-linux-amd64.tar.gz
# cd etcd-v3.1.4-linux-amd64
# mv etcd etcdctl /usr/bin/
# nohup etcd --data-dir /var/lib/data.etcd --listen-client-urls http://192.168.1.130:2379 --advertise-client-urls http://192.168.1.130:2379 &>/var/log/etcd.log &
```

key	value
/nginx/www.example.com/server_name	域名
/nginx/www.example.com/upstream/server01	节点01
/nginx/www.example.com/upstream/server02	节点02
/nginx/www.example.com/upstream/server03	节点03

Nginx集群节点自动加入

安装部署

Confd部署与配置：

```
# mv confd-0.11.0-linux-amd64 /usr/bin/confd
# chmod +x /usr/bin/confd
```

1）创建配置目录

```
# mkdir -p /etc/confd/{conf.d,templates}
```

2）创建资源模板

```
# vi /etc/confd/conf.d/www.example.com.toml
[template]
src = "www.example.com.tmpl"
dest = "/usr/local/nginx/conf/vhost/www.example.com.conf"
keys = ["/nginx/www.example.com",]
reload_cmd ="/usr/local/nginx/sbin/nginx -s reload"
```

3）创建Nginx配置文件模板

```
# vi /etc/confd/templates/www.example.com.tmpl
upstream {{getv "/nginx/www.example.com/server_name"}} {
{{range getvs "/nginx/www.example.com/upstream/*"}}
     server {{.}};
{{end}}
}
server {
   server_name {{getv "/nginx/www.example.com/server_name"}};
   location / {
       proxy_pass  http://{{getv "/nginx/www.example.com/server_name"}};
       proxy_set_header Host $host;
       proxy_set_header X-Real-IP $remote_addr;
       proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Nginx集群节点自动加入

测试

启动confd监测etcd中的keys:

```
confd -watch -backend etcd -node http://192.168.1.130:2379
```

Nginx集群节点自动加入

测试

设置key的值：

```
# etcdctl -C http://192.168.1.130:2379 set /nginx/www.example.com/server_name "www.example.com"
# etcdctl -C http://192.168.1.130:2379 set /nginx/www.example.com/upstream/server01 "192.168.1.120:80"
# etcdctl -C http://192.168.1.130:2379 set /nginx/www.example.com/upstream/server02 "192.168.1.120:8080"
```

Nginx集群节点自动加入

etcd RestAPI

```
curl http://192.168.1.130:2379/v2/keys                                    # 查看所有keys
curl -X PUT http://192.168.1.130:2379/v2/keys/test/a_key -d value="789"   # 增改key
curl -X DELETE http://192.168.1.130:2379/v2/keys/test/a_key               # 删除key
curl http://192.168.1.130:2379/v2/keys/test/a_key                         # 查询key的值
curl -X PUT http://192.168.1.130:2379/v2/keys/ttlvar -d value=“ttl_value” -d ttl=10 # 创建有效期的key，单位秒
curl -X PUT http://192.168.1.130:2379/v2/keys/dir -d dir=true             # 创建目录

curl http://192.168.1.130:2379/version                # 查看etcd版本
curl http://192.168.1.130:2379/v2/members             # 列出所有集群成员
curl http://192.168.1.130:2379/v2/stats/leader        # 查看leader
curl http://192.168.1.130:2379/v2/stats/self          # 节点自身信息
curl http://192.168.1.130:2379/v2/stats/store         # 查看集群运行状态
```

谢谢