目录

负载均衡常见问题

1.负载均衡是否支持PING

公网负载均衡支持ping，需要任意一个监听器下有后端主机；私网负载均衡目前也支持ping

2.后端 KEC 实例需要外网带宽吗？会否影响负载均衡的服务？

负载均衡服务产生的公网流量费用，由绑定的 EIP 进行收费。建议购买关联 EIP 时，设定合理的带宽峰值上限和计费方式。互联网 Web 业务的流量起伏较大，无法准确预测，若实际用量超过上限，会造成访问丢包，业务不稳定，这时可通过 EIP 的监控告警，监控流量使用情况，动态调整 EIP 带宽上限。

使用 IPv6 负载均衡时，后端 IPv6 KEC 实例需要开通公网访问能力。

3.健康检查提示后端实例异常该如何处理

请按以下步骤进行排查：

- 确保您直接通过服务器访问到您的应用服务。
- 确保后端服务器已开启了相应的端口。
- 检查后端服务器内部是否有防火墙之类的防护软件，可能导致负载均衡系统无法与后端服务器通讯。
- 检查负载均衡检查参数设置是否正确。
- 建议使用静态页面来健康检查。
- 检查后端的服务器是否有高负载导致云服务器对外响应慢。
- 确保服务器内部是否有做 iptables 限制。

4.能否支持获取客户端真实 IP？

4层：内网请参考文档：https://docs.ksyun.com/documents/6232；公网可通过命令行：netstat -aptn|grep ESTAB 获取。 7层：通过 HTTP Header X-forwarded-for 获取真实客户端 IP。

5.SLB 的 TCP 连接超时时间是多少？

900秒 idle timeout

6.负载均衡 cookies 会话保持方式的原理是什么？

在 cookie 插入模式下，SLB 将负责插入 cookie ，后端服务器无需作出任何修改。当客户进行第一次请求时，客户 HTTP 请求（不带 cookie ）进入 SLB ，SLB 根据负载平衡算法策略选择后端一台服务器，并将请求发送至该服务器，后端服务器进行 HTTP 回复（不带 cookie ）被发回 SLB ，然后 SLB 插入 cookie ，将 HTTP 回复（带 cookie ）返回到客户端。

当客户请求再次发生时，客户 HTTP 请求（带有上次 SL B插入的 cookie ）进入 SLB，然后 SLB 读出 cookie 里的会话保持数值，将 HTTP 请求（带有与上面同样的 cookie ）发到指定的服务器，然后后端服务器进行请求回复，由于服务器并不写入 cookie ，HTTP 回复将不带有 cookie ，回复流量再次经过进入 SLB 时，SLB 再次写入更新后的会话保持 cookie 。

7.四层负载均衡和七层负载均衡有什么区别？

四层均衡能力，是基于 IP +端口的负载均衡；七层是基于应用层信息（如 HTTP 头部、URL 等）的负载均衡。

四到七层负载均衡，就是在对后台的服务器进行负载均衡时，依据四层的信息或七层的信息来决定怎么样转发流量。 比如四层的负载均衡，就是通过发布三层的 IP 地址（VIP），然后加四层的端口号，来决定哪些流量需要做负载均衡，对需要处理的流量进行 NAT 处理，转发至后台服务器，并记录下这个 TCP 或者 UDP 的流量是由哪台服务器处理的，后续这个连接的所有流量都同样转发到同一台服务器处理。

七层的负载均衡，就是在四层的基础上，再考虑应用层的特征，比如同一个 Web 服务器的负载均衡，除了根据 VIP 加 80 端口辨别是否需要处理的流量，还可根据七层的 URL、浏览器类别、语言来决定是否要进行负载均衡。七层负载均衡，也称为“内容交换”，也就是主要通过报文中的真正有意义的应用层内容，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。

七层负载均衡要根据真正的应用层内容选择服务器，只能先代理最终的服务器和客户端建立连接(三次握手)后，才可能接受到客户端发送的真正应用层内容的报文，然后再根据该报文中的特定字段，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。负载均衡设备在这种情况下，更类似于一个代理服务器。负载均衡和前端的客户端以及后端的服务器

会分别建立 TCP 连接。

8.可通过配置内网型负载均衡，将流量从端口 A 转发回同一台服务器的其他端口吗？

不可以。对服务器A（1.1.1.1）端口 a 的访问可通过内网型负载均衡将请求转发至服务器B（1.1.1.2）的端口 b。但无法将流量转发至同一台服务器A（1.1.1.1）的另一端口 b。

9.负载均衡是否支持同一个后端云服务器加入不同的监听器

支持，但是云服务器只能加入同一个 VPC 实例下的负载均衡器下

10.负载均衡是否支持同一个云服务器使用不同端口加入同一个监听器

支持，同一云服务器使用不同端口加入同一个监听器，视为多个不同的 RS
例如：

```
VIP:80 -> host-A: 8080
       -> host-A: 8081
```

注意： 控制台上在添加后端服务器时需分多次添加，每次填入不同的后端端口

11.同一个负载均衡是否支持加入不同 VPC 下的云服务器

不支持。负载均衡实例是和 VPC 关联的服务，同一个负载均衡下的云主机实例只能属于同一个 VPC。

12.什么是后端服务器权重？

用户可以指定后端服务器池内各 KEC 的转发权重，权重比越高的 KEC 将被分配到更多的访问请求，用户可以根据后端 KEC 的对外服务能力和情况来区别设定。

如果您同时开启了会话保持功能，那么有可能会造成对后端应用服务器的访问并不是完全相同的，建议您可以暂时关闭会话保持功能再观察一下是否依然存在这种情况。

13.UDP 协议与 TCP 协议有什么区别？

TCP 是面向连接的协议，在正式收发数据前，必须和对方建立可靠的连接。UDP 是面向非连接的协议，它在数据发送前不与对方先进行三次握手，而是直接进行数据包发送传送。UDP 协议主要适用于关注实时性而相对不注重可靠性的场景，如视频聊天、金融实时行情推送、DNS、物联网等。

14.负载均衡流量统计和 EIP 流量统计区别

EIP 流量是在公网出入口统计，是限速后从公网到达这个 EIP 的所有流量；负载均衡流量统计是负载均衡下所有监听器（所有监听端口）的流量总和，除了公网流量，还包含从金山云机房内部访问该负载均衡的流量。负载均衡除了流量统计，还包含新建连接数和活跃连接数的统计。

15.挂载到负载均衡的服务器对应端口是否会直接放行

负载均衡支持直接放行挂载的真实服务器

16.是否支持 WS/WSS 协议

在创建支持 HTTP/HTTPS 协议的监听器时，默认支持 WS/WSS 协议（web socket），是否会转换为 WS 取决于客户端。 说明：负载均衡与后端服务的连接需要采用 HTTP/1.1，建议后端服务器采用支持 HTTP/1.1 的 Web Server 。若负载均衡与后端服务超过 60 秒无消息交互，会主动断开连接，如需要维持连接一直不中断，需要主动实现保活机制，每 60 秒内进行一次报文交互。

17.负载均衡的端口健康检查状态显示检查未开启

由于公网负载均衡需要配置公网IP才完成配置，所以在负载均衡未绑定EIP的情况下，配置不完整，功能不开启，所以健康检查为检查未开启。

HTTPS常见问题

1.HTTPS 支持的加密套件有哪些？

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA

2.HTTPS 支持哪些版本的 SSL/TLS 安全协议？

负载均衡 HTTPS 目前支持的 ssl_protocols：TLSv1 TLSv1.1 TLSv1.2

3.SLB 目前支持哪些类型的证书？

目前支持服务器证书和 CA 证书的上传，服务器证书需要上传证书内容和私钥，CA 证书只需要上传证书内容；这两种类型的证书都只支持 PEM 编码格式的上传。

4.一个监听器可以绑定多少个 HTTPS 证书？

用户使用 HTTPS 单向认证，一个监听可以绑定多个服务器证书。

5.一个证书可以应用于多少个负载均衡器，多少个监听器？

一个证书可以应用于一个或多个负载均衡器，或多个监听器。

6.为什么 HTTPS 协议实际产生的流量会比账单流量多一些？

如果用户使用 HTTPS 协议，将会使用一些流量用于协议握手，因此其实际产生的流量会比账单流量更多一些。

7.证书如何上传？

可以通过负载均衡控制台上传。

8.证书区分地域吗？

不区分。

9.HTTPS 监听使用什么端口？

不强制，建议使用 443 端口。

10.证书上传后是否可以删除？

暂时不提供删除功能。

11.证书需要上传到后端服务器吗？

不需要，负载均衡 HTTPS 提供证书管理系统管理和存储用户证书，证书不需要上传到后端服务器，用户上传到证书管理系统的私钥都会加密存储。

12.证书过期后如何处理？

当前证书过期后，需要用户手动更新证书。

13.添加证书报错怎么办？

可能是私钥内容错误，需要用户替换为新的满足需求的证书。

14.HTTPS 证书 passphrase 如何处理

不支持导入 passpharse 加密的证书，请先解密再上传

负载均衡超时问题

1.负载均衡各类型监听连接超时时间如下

TCP 900秒

UDP 120秒

HTTP 300秒

HTTPS 300秒

注：当前负载均衡未开放超时时间给用户自行设置

2. HTTP Keep-alive 超时时间

如果客户端访问 SLB HTTP 监听时使用长连接，那么这条连接最长的空闲时间为 300 秒，即如果超过 300 秒没有发送任何 HTTP 请求，这条连接将会被 SLB 主动断开。如果您的业务可能会出现超过 300 秒的空闲，需要检测连接的断开并重新发起连接。

3.为什么会碰到 VIP 连接访问超时?

注：访问超时场景很多，本文档主要是从服务端入手分析

CASE 1 VIP 被安全防护

如流量黑洞和清洗，WAF 防护（waf 的特点是为建连后向 client 和 lvs 双向发送rst 报文）

CASE 2 客户端端口不足

尤其容易发生在压测的时候，客户端端口不足会导致建立连接失败，负载均衡默认会抹除 tcp 连接的 timestamp 属性，linux 协议栈的 tw_reuse （ time_wait 状态连接复用)无法生效，time_wait 状态连接堆积导致客户端端口不足

解决方法：

客户端端使用长连接代替短连接。

使用 RST 报文断开连接（ socket 设置 SO_LINGER 属性） ，而不是发 FIN 包这种方式断开。

CASE 3 后端服务器 accept 队列满

后端服务器 accept 队列满，导致后端服务器不回复 syn_ack 报文，客户端超时。

解决方法：默认的 net.core.somaxconn 参数为 128 ，执行 sysctl -w net.core.somaxconn=1024 或者其它更大的值，并重启后端服务器上的应用。

CASE 4 从4层负载均衡后端服务器访问该4层负载均衡 VIP

4 层负载均衡，在该负载均衡的后端服务器上再去访问该负载均衡 VIP，这个目前是无法支持的，会导致连接失败，常见的场景是用户后端应用使用 URL 拼接的方式跳转访问

CASE 5 对连接超时的 rst 处理不当

负载均衡上建立 TCP 连接后如果 900s 未活动，则会向 client 和 rs 双向发送 rst 断开连接，有的应用对 rst 异常处理不当，可能会对已关闭的连接再次发送数据导致应用超时 _注： 这种情况建议调整 后端主机 系统内核参数 net.ipv4.tcp_keepalive_time 为小于 900s 的值。

后端主机支持获取客户端信息

概述

用户在主机（RS）侧加载内核模块 kgwttm.ko 后，对于 TCP 连接：可以通过 socket 函数簇函数获取客户端（ Client ）真实信息，包括客户端 IP（ cip ）、客户端端口（ cport ）。

场景介绍

Client 在 ksc 之外，通过 LB 访问 RS，RS 可为虚机或物理机。

Client 与 RS 同属 ksc，可通过 LB 访问 RS，RS 可为云服务器或裸金属服务器。

场景一：6to4

Client 通过其 ipv6 地址（cip6）访问 vip6:vport 服务，后端与 RS 通信为 ipv4 地址。Option 字段携带 cip6:cport 信息。

场景二：4to4

Client 通过其 ipv4 地址（ cip4 ）访问 vip4:vport 服务，后端与 RS 通信为 ipv4 地址。Option 字段携带 cip4:cport 信息。

实现原理

在 fullnat（ fnat ）/ttm_cip 模式下，负载均衡（ LB ）向后端 RS 转发报文时，会以其 local ip（lip）作为报文源 ip。即，常规模式下在 RS 端得不到 client 端真实 ip/port 信息。 在 tcp 建立连接阶段，ttm 通过 tcp 握手的第三个报文（ack）内利用 tcp option 字段加入 client ip/port 信息，实现在 RS 端的信息获取。Option 字段由 ksc-gw 封装进报文，由加载 kgwttm 的rs 端解封装。Option 字段格式如下所示：

```
/* 自定义 client 信息结构体 */
struct ttm_peer {
 __u16 af;
 __be16 port;
 union {
 __u32 all[4];
 __be32 ip;
 __be32 ip6[4];
 struct in_addr in;
 struct in6_addr in6;
 };
};
```

源码下载链接

源码下载链接(4to4和6to4场景均支持)

使用方法

支持内核版本：v2.6* 、v3.* 、v4.*(特定版本环境可提供技术支持)

安装

1. 内核环境准备

   ```
   # rpm -qa | grep kernel-devel-$(uname -r)
   # yum install "kernel-devel-$(uname -r)"
   ```

2. 解压 kgwttm_ipv4.zip

   ```
   # tar -zxf kgwttm_ipv4.zip
   ```

3. 编译

   ```
   # cd kgwttm
   # ./build.sh
   ```

4. 安装

   ```
   # sudo rpm -ivh kmod-kgwttm-v.*.rpm
   ```

5. 检查是否安装成功

   ```
   #lsmod | grep kgwttm
   ```

使用及验证

以 python 为例： RS 端，用户在 tcp socket server 端 accept()返回成功后，通过调用 socket.getsockopt(level,optname[, buflen])，其中 level 赋值 socket.SOL_IP, optname 赋值 1345，buflen 赋值 20。获取返回信息后，按照 struct ttm_data_v6 解析相应返回值即可。

在 6to4 以及 4to4 模式中，我们提供socket.getsockopt()指定 optname 为 1345 方式获取 client端信息。

4to4 场景，还可通过 socket.accept()返回参数或调用 socket.getpeername()获取 client 端信息。 具体样例参考附录

6to4场景后端用nginx解析时需适配

1. 官网下载nginx Code
2. 解压进入目录，configure 配置 ./configure --prefix=/usr/local/nginx --without-http_rewrite_module --add-module=../ngx_http_realipv6_module-master-6d00184751fea369efcda8e8b70306bfaf43d991/ --without-http_gzip_module （模块源码下载） 注：配置阶段需要加载一个解析模块，用于获取前端网关在option中传送过来的真实源IPv6 地址

```
//ngx_http_realipv6_module.c

#include <ngx_config.h>
#include <ngx_core.h>
#include <ngx_http.h>
#include <linux/types.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define NGX_TTM_BASE_CTL (64 + 1024 + 64 + 64 + 64 + 64) /* base */
#define NGX_TTM_SO_GET_PEER (NGX_TTM_BASE_CTL + 1)

typedef struct ttm_peer_s ttm_peer_t;

struct ttm_peer_s
{
    __u16 af;
    __be16 port;
    union {
        __u32 all[4];
        __be32 ip;
        __be32 ip6[4];
        struct in_addr in;
        struct in6_addr in6;
    };
};

typedef struct
{
    ngx_str_t header;
    ngx_flag_t set_real_ipv6;
} ngx_http_realipv6_loc_conf_t;

typedef struct
{
    ngx_str_t remote_ipv6_addr;
} ngx_http_realipv6_ctx_t;

static ngx_int_t ngx_http_realipv6_handler(ngx_http_request_t *r);
static void ngx_http_realipv6_cleanup(void *data);
static void *ngx_http_realipv6_create_loc_conf(ngx_conf_t *cf);
static ngx_int_t ngx_http_realipv6_add_variables(ngx_conf_t *cf);
static ngx_int_t ngx_http_realipv6_init(ngx_conf_t *cf);
static ngx_int_t ngx_http_realipv6_remote_addr_variable(ngx_http_request_t *r,
                                                        ngx_http_variable_value_t *v, uintptr_t data);

static ngx_command_t ngx_http_realipv6_commands[] = {

    {ngx_string("set_real_ipv6_header"),
     NGX_HTTP_MAIN_CONF | NGX_HTTP_SRV_CONF | NGX_HTTP_LOC_CONF | NGX_CONF_TAKE1,
     ngx_conf_set_str_slot,
     NGX_HTTP_LOC_CONF_OFFSET,
     offsetof(ngx_http_realipv6_loc_conf_t, header),
     NULL},

    {ngx_string("set_real_ipv6"),
     NGX_HTTP_MAIN_CONF | NGX_HTTP_SRV_CONF | NGX_HTTP_LOC_CONF | NGX_CONF_FLAG,
     ngx_conf_set_flag_slot,
     NGX_HTTP_LOC_CONF_OFFSET,
     offsetof(ngx_http_realipv6_loc_conf_t, set_real_ipv6),
     NULL},

    ngx_null_command};

static ngx_http_module_t ngx_http_realipv6_module_ctx = {
    ngx_http_realipv6_add_variables, /* preconfiguration */
    ngx_http_realipv6_init,          /* postconfiguration */

    // ngx_http_realipv6_add_variables, /* create main configuration */
    NULL,
```

```
    NULL, /* init main configuration */

    NULL, /* create server configuration */
    NULL, /* merge server configuration */

    ngx_http_realipv6_create_loc_conf, /* create location configuration */
    NULL                               /* merge location configuration */
};

ngx_module_t ngx_http_realipv6_module = {
    NGX_MODULE_V1,
    &ngx_http_realipv6_module_ctx, /* module context */
    ngx_http_realipv6_commands,    /* module directives */
    NGX_HTTP_MODULE,               /* module type */
    NULL,                          /* init master */
    NULL,                          /* init module */
    NULL,                          /* init process */
    NULL,                          /* init thread */
    NULL,                          /* exit thread */
    NULL,                          /* exit process */
    NULL,                          /* exit master */
    NGX_MODULE_V1_PADDING};

static ngx_http_variable_t ngx_http_realipv6_vars[] = {

    {ngx_string("x_real_ipv6"), NULL,
     ngx_http_realipv6_remote_addr_variable, 0, 0, 0},

    {ngx_null_string, NULL, NULL, 0, 0, 0}};

static ngx_int_t
ngx_http_realipv6_handler(ngx_http_request_t *r)
{
    u_char *p;
    socklen_t len;
    size_t plen;
    ngx_connection_t *c;
    ngx_http_realipv6_loc_conf_t *rlcf;
    ngx_pool_cleanup_t *cln;
    ngx_http_realipv6_ctx_t *ctx;

    rlcf = ngx_http_get_module_loc_conf(r, ngx_http_realipv6_module);

    if (!rlcf->set_real_ipv6)
    {
        return NGX_DECLINED;
    }
    c = r->connection;
    ttm_peer_t peer;
    len = sizeof(peer);
    if (getsockopt(c->fd, IPPROTO_IP, NGX_TTM_SO_GET_PEER, &peer, &len) < 0)
    {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "nginx getsockopt failed");
        return NGX_DECLINED;
    }
    char addr[NGX_SOCKADDR_STRLEN];
    const char *ret = inet_ntop(peer.af, &peer.all, addr, sizeof(addr));
    if (ret == NULL)
    {
        ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
                      "nginx inet_ntop failed");
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }

    cln = ngx_pool_cleanup_add(r->pool, sizeof(ngx_http_realipv6_ctx_t));
    if (cln == NULL)
    {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }
    ctx = cln->data;
    ngx_http_set_ctx(r, ctx, ngx_http_realipv6_module);

    c = r->connection;
    plen = ngx_strlen(ret);
    p = ngx_pnalloc(c->pool, plen);
    if (p == NULL)
    {
        return NGX_HTTP_INTERNAL_SERVER_ERROR;
    }
    ngx_memcpy(p, ret, plen);
    cln->handler = ngx_http_realipv6_cleanup;
    ctx->remote_ipv6_addr.len = plen;
    ctx->remote_ipv6_addr.data = p;
```

```
    return NGX_DECLINED;
}

static void
ngx_http_realipv6_cleanup(void *data)
{
    return;
}

static void *
ngx_http_realipv6_create_loc_conf(ngx_conf_t *cf)
{
    ngx_http_realipv6_loc_conf_t *conf;

    conf = ngx_pcalloc(cf->pool, sizeof(ngx_http_realipv6_loc_conf_t));
    if (conf == NULL)
    {
        return NULL;
    }

    ngx_str_null(&conf->header);
    conf->set_real_ipv6 = NGX_CONF_UNSET;

    return conf;
}

static ngx_int_t
ngx_http_realipv6_add_variables(ngx_conf_t *cf)
{
    ngx_http_variable_t *var, *v;

    for (v = ngx_http_realipv6_vars; v->name.len; v++)
    {
        var = ngx_http_add_variable(cf, &v->name, v->flags);
        if (var == NULL)
        {
            return NGX_ERROR;
        }

        var->get_handler = v->get_handler;
        var->data = v->data;
    }

    return NGX_OK;
}

static ngx_int_t
ngx_http_realipv6_init(ngx_conf_t *cf)
{
    ngx_http_handler_pt *h;
    ngx_http_core_main_conf_t *cmcf;

    cmcf = ngx_http_conf_get_module_main_conf(cf, ngx_http_core_module);

    h = ngx_array_push(&cmcf->phases[NGX_HTTP_POST_READ_PHASE].handlers);
    if (h == NULL)
    {
        return NGX_ERROR;
    }

    *h = ngx_http_realipv6_handler;

    return NGX_OK;
}

static ngx_int_t
ngx_http_realipv6_remote_addr_variable(ngx_http_request_t *r,
                                       ngx_http_variable_value_t *v, uintptr_t data)
{
    ngx_str_t *remote_ipv6_addr;
    ngx_pool_cleanup_t *cln;
    ngx_http_realipv6_ctx_t *ctx;

    ctx = ngx_http_get_module_ctx(r, ngx_http_realipv6_module);

    if (ctx == NULL && (r->internal || r->filter_finalize))
    {
        /*
         * if module context was reset, the original address
         * can still be found in the cleanup handler
         */
        for (cln = r->pool->cleanup; cln; cln = cln->next)
        {
            if (cln->handler == ngx_http_realipv6_cleanup)
```

```
            {
                ctx = cln->data;
                break;
            }
        }
    }

    // remote_ipv6_addr = ctx->remote_ipv6_addr;
    remote_ipv6_addr = ctx ? &ctx->remote_ipv6_addr : &r->connection->addr_text;
    v->len = remote_ipv6_addr->len;
    v->valid = 1;
    v->no_cacheable = 0;
    v->not_found = 0;
    v->data = remote_ipv6_addr->data;

    return NGX_OK;
}
```

3. make&& make install
4. 将编译好的nginx 更新到/usr/sbin/目录下
5. 调整配置文件nginx.conf

//nginx.conf

```
user nobody;

worker_processes 1;

error_log logs/error.log;

error_log logs/error.log notice;

error_log logs/error.log info;

pid logs/nginx.pid;

events { worker_connections 1024; }

http { include mime.types; default_type application/octet-stream;

log_format  main  '$x_real_ipv6 $remote_addr - $remote_user [$time_local] "$request" ' //默认这两行是注释掉，去掉注释。添加Log输出变量 x_real_ipv6
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';

access_log  /var/log/nginx/access.log  main; //打开log文件实时输出log 日志

sendfile        on;
#tcp_nopush     on;

#keepalive_timeout  0;
keepalive_timeout  65;

#gzip  on;

server {
    listen       80;
    server_name  localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location / {
        root   html;
        index  index.html index.htm;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
```

```
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}
}

# another virtual host using mix of IP-, name-, and port-based configuration
#
#server {
#    listen       8000;
#    listen       somename:8080;
#    server_name  somename  alias  another.alias;

#    location / {
#        root   html;
#        index  index.html index.htm;
#    }
#}

# HTTPS server
#
#server {
#    listen       443 ssl;
#    server_name  localhost;

#    ssl_certificate      cert.pem;
#    ssl_certificate_key  cert.key;

#    ssl_session_cache    shared:SSL:1m;
#    ssl_session_timeout  5m;

#    ssl_ciphers  HIGH:!aNULL:!MD5;
#    ssl_prefer_server_ciphers  on;

#    location / {
#        root   html;
#        index  index.html index.htm;
#    }
#}

}


### 卸载
1. 检查 rpm 包安装情况
```language
# rpm  -qa|grep kgwttm
```

2. 卸载

```
# rpm -e `rpm -qa|grep ttm`
```

3. 再次检查

```
# lsmod|grep kgwttm
```

附录
```

用户安装 kgwttm 模块后，可直接运行源码包内 example_ipv6/ 目录下的 c/python server 端 demo 实例。基于业务场景提供 client 端 tcp 访问，在 RS 侧观察效果。

代码示例一：PYTHON

```
… … # 用户 socket 逻辑
ss, addr = s.accept()
# 4to4 场景，可直接获得 cip/cport 信息
# 6to4 场景，获得 xgw 代理 lip/lport 信息
print 'get client info:', addr # 4to4 方法一
print 'getpeername:', ss.getpeername() # 4to4 方法二
# 0 == SOL_IP; 1345 == TTM_SO_GET_PEER; 20 == buff len
str = ss.getsockopt(0, 1345, 20) # 4to4 方法三 / 6to4 方法
# 按照 struct ttm_data_v6 格式解析
obj_p = struct.Struct('!bbH4s12s')
data = obj_p.unpack(str)
# 根据协议簇解析对应信。2 == AF_INET， 10 == AF_INET6
# 4to4 场景推荐使用上述两种方式获取 client 信息
print 'port:', data[2]
if(data[0] == 2):
 print 'ipv4:', inet_ntop(data[0], data[3])
else:
 print 'ipv6:', inet_ntop(data[0], data[3]+data[4]) # 10 is AF_INET6
… … # 用户后续逻辑
```

代码示例二：C

```
#define TTM_BASE_CTL (64+1024+64+64+64+64) /* base */
#define TTM_SO_GET_PEER (TTM_BASE_CTL+1) /* 1345, 自定义 id */
/* 自定义 client 信息结构体 */
struct ttm_peer {
 __u16 af;
 __be16 port;
 union {
 __u32 all[4];
 __be32 ip;
 __be32 ip6[4];
 struct in_addr in;
 struct in6_addr in6;
 };
};
 … … //用户 socket 逻辑
 new_fd=accept(sockfd,(struct sockaddr *)&client_addr, &sin_size);
 /* 4to4 场景，可直接获得 cip/cport 信息 */
 fprintf(stderr,"Server get connection from %s:%u\n",
 inet_ntoa(client_addr.sin_addr),
 ntohs(client_addr.sin_port));
 /* 4to4 或 6to4 获取 client 信息 */
 if (getsockopt(new_fd, IPPROTO_IP, TTM_SO_GET_PEER, &peer, &len) < 0) {
 fprintf(stderr, "getsockopt failed\n");
 } else {
 inet_ntop(peer.af, &peer.all, &addr_string, sizeof(addr_string));
 printf("getsockopt success(addr:%s port:%u)\n", addr_string, ntohs(peer.port));
 }
```

负载均衡访问日志服务

日志服务说明

- 负载均衡的日志服务只针对 SLB 下的七层监听器进行日志记录并上传；
- 每小时上传一个日志文件，以天为粒度创建文件夹；
- 配置下发 3 ~ 10 min 后开始采集数据。

操作说明

1. 经典型负载均衡可开启访问日志功能，点击公网类型负载均衡，下方会弹出详情界面，找到访问日志下【编辑】按钮，弹出对话框，选择访问日志【开启】，并输入相应的 KS3 Bucket 地址（ KS3 Bucket 需与 SLB 在同一地域）；
2. 若无现有 Bucket，点击【新建 Bucket】按钮，即跳转至 KS3 控制台进行新建操作（新建Bucket可以是私密权限），日志服务器将获取Bucket外链进行日志上传；

3. 完成日志服务创建，并从相应的 KS3 Bucket 查阅日志文件，文件夹命名如下图；

4. 每小时日志文件名由{实例id}+{UNIX时间戳}及其他固定字段拼接而成（若一小时内无访问记录或后端服务器无任何响应，则不生成该小时的日志文件）。

日志信息

变量名	说明
time_iso8601	时间戳
remote_addr:$remote_port	client ip地址和端口
server_addr:$server_port	监听器地址和端口
upstream_addr	后端服务器地址
request_time	请求处理时间，单位是秒
upstream_response_time	后端返回时间，单位是秒
status:tengine	返回的状态码
upstream_status	后端服务器返回的状态码
request_length	请求长度
body_bytes_sent	发送给客户端的字节数，不包括响应头的大小
request_method:http	请求方法
http_host	请求url的host
request_uri	客户端请求的uri
server_protocol	server段协议
http_referer	来源页面
http_user_agent	用户浏览器其他信息，浏览器版本、浏览器类型等

裸金属服务器获取源IP方法

Centos

第一步，查看系统内核版本

例如内核版本为3.10.0-957.1.3.el7.x86_64

```
# uname -r #
# 3.10.0-957.1.3.el7.x86_64
```

第二步，下载对应的内核RPM包

下载centosrpm包，从中找到对应内核版本的RPM

如未找到可从centos官方获取 https://wiki.centos.org/HowTos/I_need_the_Kernel_Source

```
# rpm -ivh kernel-devel-3.10.0-327.el7.x86_64.rpm
```

第三步，安装依赖环境

```
# sudo yum install rpm-build redhat-rpm-config asciidoc hmaccalc perl-ExtUtils-Embed pesign xmlto
$ sudo yum install audit-libs-devel binutils-devel elfutils-devel elfutils-libelf-devel gcc
$ sudo yum install ncurses-devel newt-devel numactl-devel pciutils-devel python-devel zlib-devel
```

第四步，编译&安装TTM模块

下载ttm源码

```
# tar -zxvf kgwttm.tar.gz
# cd kgwttm
# ./build.sh
# rpm -ivh *.rpm
# sh /usr/local/bin/kgwttm-insmod.sh
# lsmod |grep kgwttm
```

Ubuntu

编译&安装TTM模块

下载ttm源码

```
# tar -zxvf kgwttm.tar.gz
# cd kgwttm
# make
# sock_def_readable_addr=`cat /proc/kallsyms | grep sock_def_readable | awk '{print $1}'`
# insmod kgwttm.ko sk_data_ready_addr="0x$sock_def_readable_addr"
```