varnish网站加速缓存代理

**重点内容：CDN与varnish简介、varnish工作原理、部署varnish实现网站加速与高可用；**

**一、varnish简介：**

**1.Varnish：**一款高性能的开源HTTP加速器，2006年发布的第一个版本0.9，发展到目前很多门户网站已经部署了varnish，并且反应都很好，甚至反应比squid还稳定，且效率更高，资源占用更少。在反向代理，web加速方面，varnish已经有足够能力代替squid。挪威最大的在线报纸 Verdens Gang 使用3台Varnish代替了原来的12台Squid，性能比以前更好。

2.**作者：**Poul-Henning Kamp是FreeBSD的内核开发者之一。

**3.储存介质：**内存、硬盘与CPU内的L1、L2，甚至有L3缓存。

**4.CDN （content delivery network）内容分发（推送）网络：**是在现有的Internet中增加一层新的网络架构，将网络内容发布到最接近用户的网络边缘（边缘服务器），使用户的网站访问请求尽可能避免影响访问速度的因素；

推荐阅读百度百科CDN：https://baike.baidu.com/item/CDN/420951?fr=aladdin

知乎CDN：https://www.zhihu.com/question/37353035

**5.缓存注意事项：**

1）缓存的内容通常不宜太长，否则后端真正的数据源发生改变了，前端的缓存依然会提供旧的内容响应给客户；

2）缓存服务器通常使用内存作为缓存的存储介质，与磁盘相比，内存的造价要昂贵的多，所以不可能把所有数据都缓存进来，只能缓存经常被用户重复访问的热点数据；

3）缓存有的是时候会缓存一些有问题的页面，比如说错误的页面、旧的页面等，我们应该有一种机制，可以人工智能清理一些有问题的缓存条目，保证用户在访问任何页面时都不出现问题；

4）任何有用户私有信息的一定不能缓存，否则有可能会被其他用户看到，从而泄露用户隐私；

**二、varnish工作原理：**

**1.varnish的系统架构：**

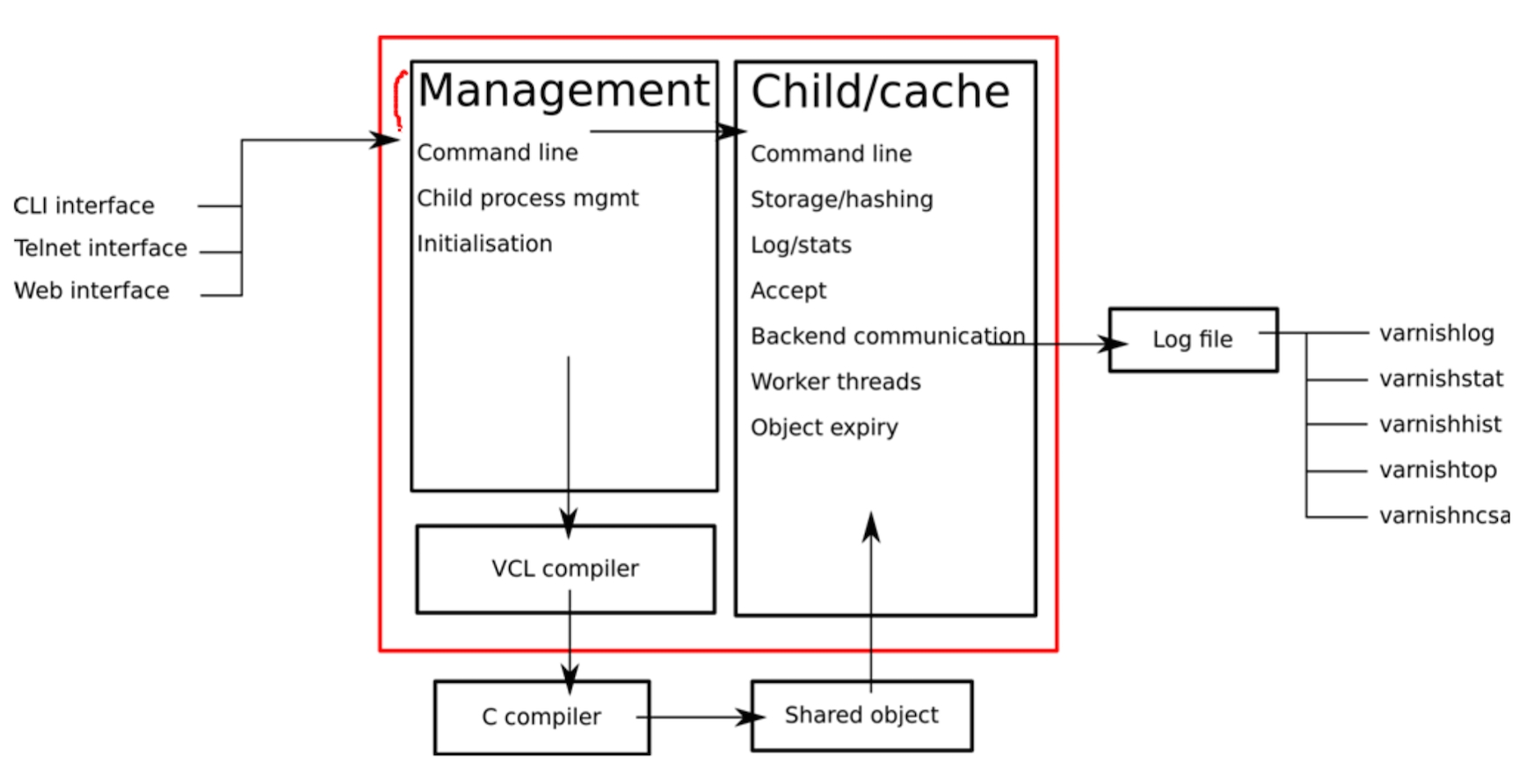
**Management管理进程**：主要实现加载新的配置、编译 VCL、监控 varnish、初始化 varnish以及提供一个命令行接口；

**child缓存进程**：也被称之为cache进程，包含多种类型的线程如下：

Acceptor 线程：接收新的连接请求并响应；

Worker 线程：child 进程会为每个会话启动一个 worker 线程；

Expiry 线程：从缓存中清理过期内容；



**2. varnish 的后端缓存存储：**

varnish 支持不同类型的后端缓存存储类型，这可以在 varnishd 启动时使用-s 选项指定。类型包括：

file：使用特定的磁盘文件存储全部的缓存数据，不过key和文件描述符还是保存在内存中，如果重启varnish，将找不到缓存数据；

malloc：将缓存的数据完全存放在内存中，工作效率更高的模式；

persistent(experimental)：与 file 的功能相同，但可以持久存储数据（即重启varnish 数据时不会被清除），仍处于测试期；

**3.varnish 与 squid 的对比：**

优点： varnish 具有更好的稳定性、更快的访问速度、更多的并发连接支持数，可以通过管理端口管理缓存等优势；

缺点：在高并发状态下， varnish 消耗更多的 CPU、 I/O 和内存资源。 varnish 进程一旦挂起、崩溃或者重启，缓存的数据会从内存中释放，此时所有的请求都将会转发到后端服务器上，给后端服务器造成很大压力。

**4.varnish的工作状态：**

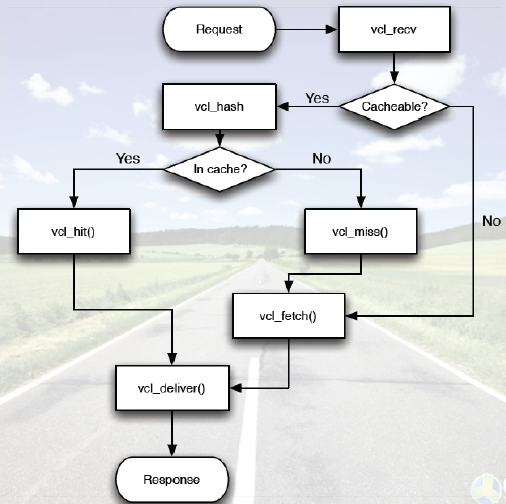
1）Receive 状态：请求处理入口状态，根据 VCL 规则判断该请求应该 Pass 或 Pipe，还是进入 Lookup(本地查询)。

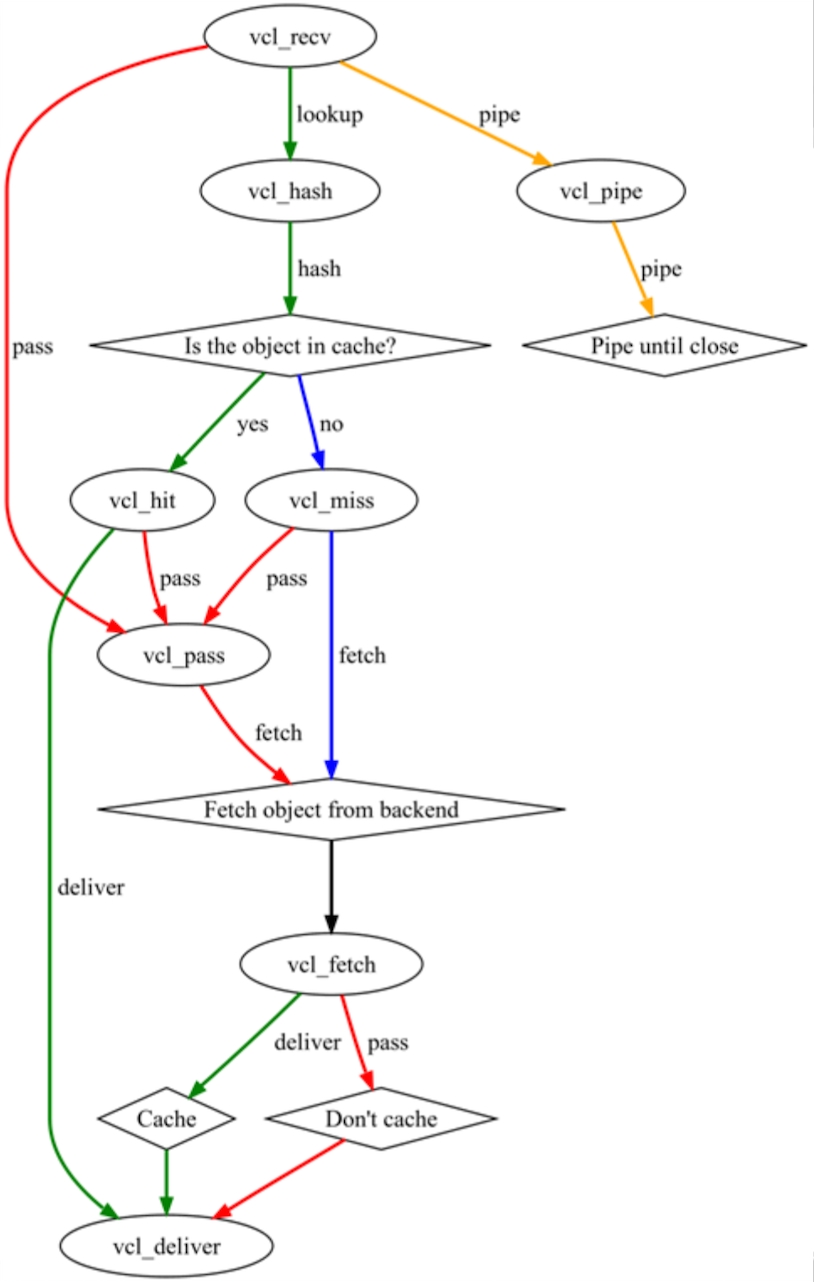
2）Lookup 状态：进入此状态后，会在 hash 表中查找数据，若找到，则进入 Hit 状态，否则进入 Miss 状态。

3）Fetch 状态：在 Fetch 状态下，对请求进行后端获取，发送请求，获得数据，并进行本地存储。

4）Deliver 状态：将获取到的数据发送给客户端，然后完成本次请求。

5）Pipe 状态：不通过 varnish，开通“管道”，直接有后端真实的 web 节点回复客户端请求。





**总结：**state engine的常见数据流向：

1.vcl\_recv --> lookup --> vcl\_hash --> cached --> vcl\_hit-->  vcl\_deliver

数据报文进来后先检查，如果可缓存，则送进缓存中匹配，如果命中了，则直接响应给客户；

2.vcl\_recv --> lookup --> vcl\_hash --> cached --> vcl\_miss --> vcl\_backend\_fetch --> vcl\_deliver

数据报文进来之后先检查，如果可缓存，则送进缓存中匹配，如果没命中缓存，varnish会替用户去后端应用服务器请求对应的页面，先缓存下来，然后响应给客户；

3.vcl\_recv --> vcl\_pipe --> backendserver

送往vcl\_pipe的报文不会做任何处理直接送到后端服务器；

4.vcl\_recv --> vcl\_pass --> vcl\_backend\_fetch --> vcl\_deliver

通常用于不能缓存的内容，不检查缓存直接送往vcl\_pass，经由pass送到后端服务器去响应；

**网站：**http://blog.sina.com.cn/s/blog\_130affe1d0102w4ft.html

**三、搭建Varnish+httpd实现网站加速及高可用：**

**案例环境：**

|  |  |  |  |
| --- | --- | --- | --- |
| 系统 | IP地址 | 主机名 | 所需软件 |
| centos 7.4 x64 1708 | 192.168.100.101 | varnish | varnish-4.0.1.tar.gz |
| centos 7.4 x64 1708 | 192.168.100.102 | web1 | httpd、php、mariadb |
| centos 7.4 x64 1708 | 192.168.100.103 | web2 | httpd |

**案例步骤：**

* **安装并配置varnish代理；**
* **配置后端web节点；**
* **客户端访问测试代理；**
* **验证服务端日志记录情况；**
* **安装并配置varnish代理；**

[root@varnish ~]# yum -y install libtool ncurses-devel pcre-devel libxslt groff pkgconfig libedit-devel python-imaging python-docutils gcc gcc-c++

[root@varnish ~]# tar zxf varnish-4.0.1.tar.gz -C /usr/src/

[root@varnish ~]# cd /usr/src/varnish-4.0.1/

[root@varnish varnish-4.0.1]# ./configure --prefix=/usr/local/varnish --enable-debugging-symbols --enable-dependency-tracking --enable-developer-warnings &&make &&make install  
[root@varnish varnish-4.0.1]# cd

[root@varnish ~]# ln -s /usr/local/varnish/bin/\* /usr/local/bin/

[root@varnish ~]# ln -s /usr/local/varnish/sbin/\* /usr/local/sbin/

[root@varnish ~]# cd /usr/local/varnish/

[root@varnish varnish]# ls

##样例配置文件模板：/usr/local/varnish/share/doc/varnish/example.vcl

bin include lib sbin share var

[root@varnish varnish]# vi /usr/local/varnish/default.vcl

vcl 4.0;

import directors;

probe check {

.url = "/"; ##探测后端主机健康状态时请求的URL，默认为/

.window = 5; ##设置最近检查次数，用来判断后端主机的健康状态

.threshold = 4; ##在window的检查次数中，至少有几次正常，才会认为主机是正常状态

.interval = 2s; ##检查时间间隔

.timeout = 1s; ##检查请求的过期时常

}

backend web1 {

.host = "192.168.100.102";

.port = "80";

.probe = check;

}

backend web2 {

.host = "192.168.100.103";

.port = "80";

.probe = check;

}

sub vcl\_init {

new bar = directors.round\_robin(); ##设置server 池，算法：random随机、round-robin加权轮询、dns基于 DNS 名称解析进行调度算法

bar.add\_backend(web1);

bar.add\_backend(web2);

}

sub vcl\_recv {

# set req.backend\_hint = bar.backend(); ##如果不设置if条件匹配，则将所有请求转发到pool池中，则使用此配置项

if (req.url ~ "(?i)\.php$") { ##以php结尾的请求，转发到web1节点

set req.backend\_hint = web1;

} else {

set req.backend\_hint = bar.backend();

}

if (req.url ~"(?i)^/login") { ##请求以/login结尾，不进行缓存，直接转发到pool bar

return(pass);

}

}

sub vcl\_backend\_response {

if (beresp.http.cache.control !~ "s-maxage") {

if (bereq.url ~ "(?i)\.jpg$") {

set beresp.ttl = 3600s; #设置.jpg结尾的图片TTL时长，加长其缓存时长

unset beresp.http.Set-Cookie; #取消追踪图片的cookie信息

}

}

if (beresp.http.cache.control !~ "s-maxage") {

if (bereq.url ~ "(?i)\.css$") {

set beresp.ttl = 3600s;

unset beresp.http.Set-Cookie;

}

}

}

sub vcl\_deliver {

if (obj.hits>0) { #自定义响应报文的首部信息

set resp.http.X-Cache = "Hit via "+ server.ip;

}else {

set resp.http.X-Cache = "Miss via "+ server.ip;

}

}

:wq

[root@varnish varnish]# varnishd -f default.vcl -a 0.0.0.0:80

启动varnish服务其他用法：

[root@varnish varnish]# varnishd -? ##可以查看命令的帮助用法

[root@varnish varnish]# varnishd -f /usr/local/varnish/default.vcl -a 0.0.0.0:80 -s file,/var/lib/varnish/varnish\_storage.bin,1G -p thread\_pool\_min=5 -p thread\_pool\_max=500 -p thread\_pool\_timeout=300 ##将其缓存数据存放类型改为file，并设置文件位置及大小，设置线程池最小线程数，最大线程数，线程超时时间；

[root@varnish varnish]# netstat -utpln |grep varnish

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 1866/varnishd

tcp 0 0 127.0.0.1:44730 0.0.0.0:\* LISTEN 1866/varnishd

* **配置后端web节点；**

[root@web1 ~]# yum -y install httpd php php-mysql mariadb-server mariadb

[root@web1 ~]# echo "web1" >/var/www/html/index.html

[root@web1 ~]# cat <<END >>/var/www/html/index.php

<?php

phpinfo();

?>

END

[root@web1 ~]# systemctl start httpd mariadb

[root@web1 ~]# systemctl enable httpd mariadb

[root@web2 ~]# yum -y install httpd

[root@web2 ~]# echo "web2" >/var/www/html/index.html

[root@web1 ~]# systemctl start httpd

[root@web1 ~]# systemctl enable httpd

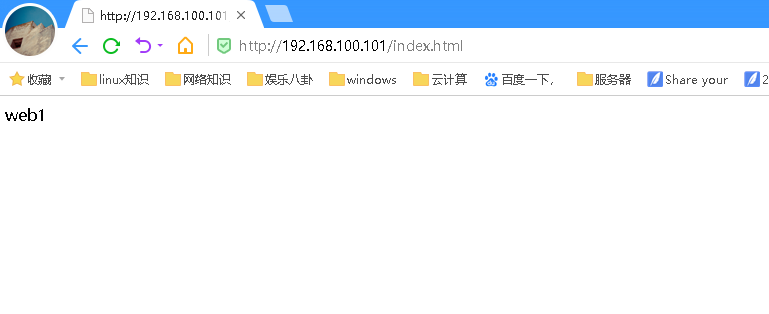
* **客户端访问测试代理；**

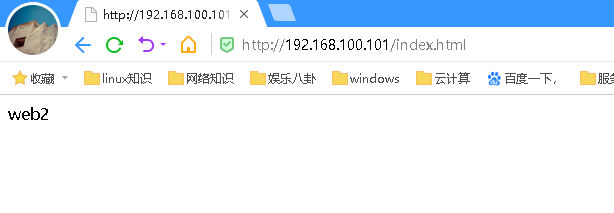
**总结：**

**1.当后端节点httpd服务正常，则访问其服务；**

**2.当后端节点httpd服务停止，则会访问其varnish的缓存数据；**

**3.如若无缓存数据或者缓存数据过期，则varnish会将其客户端请求代理到其他web节点；**







* **验证服务端日志记录情况；**

[root@varnish ~]# varnishlog ##动态查看varnish代理日志，ctrl C 退出

[root@web1 ~]# tail -3 /var/log/httpd/access\_log ##查看节点web1的日志

192.168.100.101 - - [16/Jun/2018:02:54:27 +0800] "GET / HTTP/1.1" 200 5 "-" "-"

192.168.100.101 - - [16/Jun/2018:02:54:29 +0800] "GET / HTTP/1.1" 200 5 "-" "-"

192.168.100.101 - - [16/Jun/2018:02:54:31 +0800] "GET / HTTP/1.1" 200 5 "-" "-"

[root@web2 ~]# tail -3 /var/log/httpd/access\_log ##查看节点web2的日志

192.168.100.101 - - [16/Jun/2018:03:00:20 +0800] "GET / HTTP/1.1" 200 5 "-" "-"

192.168.100.101 - - [16/Jun/2018:03:00:22 +0800] "GET / HTTP/1.1" 200 5 "-" "-"

192.168.100.101 - - [16/Jun/2018:03:00:24 +0800] "GET / HTTP/1.1" 200 5 "-" "-"