链滴

nuster: 基于 HAProxy 的高性能 HTTP 缓存服务器和 RESTful NoSQL 缓存服务器

作者：wubodibo

原文链接：https://ld246.com/article/1531098759587

来源网站：链滴

nuster

Wiki | English | 中文 | 日本語

基于HAProxy的高性能HTTP缓存服务器和RESTful NoSQL缓存服务器。

中文版更新可能不及时，最新版请参照英文版README.md

目录

介绍

nuster是一个基于HAProxy的高性能HTTP缓存服务器和RESTful NoSQL缓存服务器，完全兼容HAPrxy，并且利用HAProxy的ACL功能来提供非常细致的缓存规则。

特性

HTTP/TCP负载均衡器

nuster可以作为HTTP/TCP负载均衡器使用。

- 继承了HAProxy的所有特性，完全兼容HAProxy

- 负载均衡
- 前端后端HTTPS
- HTTP压缩
- HTTP重写重定向
- HTTP信息增删改
- HTTP2
- 监控
- 粘性
- 访问控制
- 内容切换

HTTP缓存服务器

nuster也可以用作类似Varnish或者Nginx那样的HTTP缓存服务器，来缓存动态或者静态的HTTP资
。

- HAProxy的所有特性(HTTPS, HTTP/2, ACL, etc)
- 非常快
- 强大的动态缓存功能
  - 基于HTTP method, URI, path, query, header, cookies, etc
  - 基于HTTP request or response contents, etc
  - 基于environment variables, server state, etc
  - 基于SSL version, SNI, etc
  - 基于connection rate, number, byte, etc
- 缓存管理
- 缓存清除
- 缓存统计信息
- 缓存生存时间
- 持久化

RESTful NoSQL缓存服务器

nuster也可以用作RESTful NoSQL缓存服务器, 用HTTP POST/GET/DELETE 来 添加/取得/删除 Key/
alue.

可以像Memcached或者Redis那样放在应用和数据库之间作为内部KV缓存使用，也可以放在用户和
用之间作为面向用户的NoSQL使用。

支持header, cookie等等，所以可以将不同的用户数据存到相同的路劲。

- HAProxy的所有特性(HTTPS, HTTP/2, ACL, etc)
- 有条件的缓存

- 内部KV缓存
- 面向用户缓存
- 支持任何类型的数据
- 支持所有编程语言，不需要特定的库，只需HTTP支持
- 持久化

性能

非常快, 单进程模式下是nginx的3倍，多进程下nginx的2倍，varnish的3倍。

详见benchmark

入门指南

下载

生产环境的话从Download下载最新稳定版, 其他情况可以git clone。

编译

```
make TARGET=linux2628 USE_LUA=1 LUA_INC=/usr/include/lua5.3 USE_OPENSSL=1 USE_PCRE=1 USE_ZLIB=1
make install PREFIX=/usr/local/nuster
```

添加USE_PTHREAD_PSHARED=1使用pthread

如果不需要可以删除USE_LUA=1 LUA_INC=/usr/include/lua5.3 USE_OPENSSL=1 USE_PCRE=1 USE_ZLIB=1

具体可以参考HAProxy README。

配置文件

准备一个配置文件: nuster.cfg

```
global
   nuster cache on data-size 100m uri /_nuster
   nuster nosql on data-size 200m
   master-worker # v3
defaults
   mode http
frontend fe
   bind *:8080
   #bind *:4433 ssl crt example.com.pem alpn h2,http/1.1
   use_backend be2 if { path_beg /_kv/ }
   default_backend be1
backend be1
   nuster cache on
   nuster rule img ttl 1d if { path_beg /img/ }
```

```
    nuster rule api ttl 30s if { path /api/some/api }
    server s1 127.0.0.1:8081
    server s2 127.0.0.1:8082
backend be2
    nuster nosql on
    nuster rule r1 ttl 3600
```

nuster监听8080端口，接受HTTP请求。

/_kv/开头的请求分配到backend be2, 可以发送HTTP POST/GET/DELETE到/_kv/any_key 来 添加/得/删除 Key/Value.

其他的请求都被分配到backend be1, 并且会被转发到服务器s1 or s2. 其中/img/*请求会被缓存1天而/api/some/api会被缓存30秒。

启动

/usr/local/nuster/sbin/nuster -f nuster.cfg

Docker

```
docker pull nuster/nuster
docker run -d -v /path/to/nuster.cfg:/etc/nuster/nuster.cfg:ro -p 8080:8080 nuster/nuster
```

使用方法

nuster基于HAProxy, 支持所有的HAProxy指令。

基本

配置文件里有四个基本的sections: global, defaults, frontend and backend。

- global
  - 定义全局指令
  - 需要定义 nuster cache on or nuster nosql on，否则cache和nosql无法使用
- defaults
  - 定义 frontend, backend的默认参数
  - 可以在 frontend or backend section重新定义
- frontend
  - 定义监听端口等等面向用户的设置
- bankend
  - 定义后端服务器等等设置
  - 需要设置 nuster cache on or nuster nosql on, 否则该backend没有nosql或者nosql功能
  - 需要设置 nuster rule

可以定义多个frontend or backend . 如果定义了nuster cache|nosql off或者没有nuster cache|nosq on|off, nuster就是一个HAProxy。

无法在listen里定义nuster。

具体参考/doc下的HAProxy文档, 或者在线HAProxy文档

As TCP loader balancer

```
frontend mysql-lb
  bind *:3306
  mode tcp
  default_backend mysql-cluster
backend mysql-cluster
  balance roundrobin
  mode tcp
  server s1 10.0.0.101:3306
  server s2 10.0.0.102:3306
  server s3 10.0.0.103:3306
```

As HTTP/HTTPS loader balancer

```
frontend web-lb
  bind *:80
  #bind *:443 ssl crt XXX.pem
  mode http
  default_backend apps
backend apps
  balance roundrobin
  mode http
  server s1 10.0.0.101:8080
  server s2 10.0.0.102:8080
  server s3 10.0.0.103:8080
  #server s4 10.0.0.101:8443 ssl verify none
```

As HTTP cache server

```
global
   nuster cache on data-size 200m
frontend fe
   bind *:8080
   default_backend be
backend be
   nuster cache on
   nuster rule all
   server s1 127.0.0.1:8081
```

As RESTful NoSQL cache server

```
global
   nuster nosql on data-size 200m
frontend fe
```

```
    bind *:8080
    default_backend be
backend be
    nuster nosql on
    nuster rule r1 ttl 3600
```

指令

global: nuster cache|nosql

syntax:

nuster cache on|off [data-size size] [dict-size size] [dir DIR] [dict-cleaner n] [data-cleaner n] [disk-cleaner n] [disk-loader n] [disk-saver n] [purge-method method] [uri uri]

nuster nosql on|off [data-size size] [dict-size size] [dir DIR] [dict-cleaner n] [data-cleaner n] [disk-cleaner n] [disk-loader n] [disk-saver n]

default: none

context: global

控制是否开启cache或者nosql。

会分配一块data-size + dict-size的共享内存来存储HTTP头，数据，key等等，临时数据从系统内存分配。

如果没有足够内存，新的请求不会被缓存直到有内存被释放。

data-size

和dict-size一起决定内存块的大小。

可以使用m, M, g 和 G. 默认是1MB，同时也是最小值。

dict-size

决定hash table的大小.

可以使用m, M, g 和 G. 默认是1MB，同时也是最小值。

这个决定hash table buckets的大小，并非key的大小，key存在共享内存中。

dict-size(bucket数) 不等于 **key数**. 就算key的数量超过了dict-size，只要整个共享内存有空间，新key仍然可以被添加。

不过如果key数超过dict-size(bucket数)性能也许会下降. dict-size可以设为大概的最大key数乘以8。

将来版本会删除dict-size, 像第一版本那样自动伸缩

dir

设置硬盘缓存文件的根目录，必须设置以开启硬盘缓存功能。

dict-cleaner

每次检查 dict-cleaner 个entry，无效的entry将被删除（默认100）

data-cleaner

每次检查 data-cleaner 个entry，无效的data将被删除（默认100）

disk-cleaner

每次检查 disk-cleaner 个硬盘缓存文件，无效的文件将被删除（默认100）

disk-loader

启动后每次加载 disk-loader 个硬盘缓存文件的信息到内存（默认100）

disk-saver

每次检查 disk-saver 个data，并将需要保存至硬盘的data保存到硬盘（默认100）

详细请参考nuster rule disk mode.

purge-method [cache only]

自定义PURGE用的HTTP method，最大14个字符，默认是 PURGE.

uri [cache only]

定义并开启cache manager/stats API

nuster cache on uri /_my/_unique/_/_cache/_uri

cache manager/stats默认是关闭的. 如果开启了，主义开启访问控制(see FAQ).

具体请参考缓存管理 和 缓存统计.

proxy: nuster cache|nosql

syntax:

nuster cache [on|off]

nuster nosql [on|off]

default: on

context: backend

决定是否在这个backend开启cache/nosql。
如果这个section有filter，记得放在最后。

nuster rule

syntax: nuster rule name [key KEY] [ttl TTL] [code CODE] [disk MODE] [if|unless condition]

default: none

context: backend

定义cache/nosql的生效条件，需要定义至少一个rule。

```
nuster cache on

# cache request `/asdf` for 30 seconds
nuster rule asdf ttl 30 if { path /asdf }

# cache if the request path begins with /img/
nuster rule img if { path_beg /img/ }

# cache if the response header `cache` is `yes`
acl resHdrCache res.hdr(cache) yes
nuster rule r1 if resHdrCache
```

可以定义多个rule，按定义顺序先后匹配。

```
acl pathA path /a.html
nuster cache on
nuster rule all ttl 3600
nuster rule path01 ttl 60 if pathA
```

rule path01永远不会被匹配。

name

定义rule的name。

在cache manager API中使用, 不必唯一但是建议不同的rule用不同的name，否则相同name的rule作一样。

key KEY

定义cache/nosql的key, 由下列关键字加组成

- method: http method, GET/POST...
- scheme: http or https
- host: the host in the request
- uri: first slash to end of the url
- path: the URL path of the request

- delimiter: '?' if query exists otherwise empty
- query: the whole query string of the request
- header_NAME: the value of header NAME
- cookie_NAME: the value of cookie NAME
- param_NAME: the value of query NAME
- body: the body of the request

CACHE的默认key是 method.scheme.host.uri, NoSQL的默认key是 GET.scheme.host.uri.

Example

GET http://www.example.com/q?name=X&type=Y

http header:
GET /q?name=X&type=Y HTTP/1.1
Host: www.example.com
ASDF: Z
Cookie: logged_in=yes; user=nuster;

生成:

- method: GET
- scheme: http
- host: www.example.com
- uri: /q?name=X&type=Y
- path: /q
- delimiter: ?
- query: name=X&type=Y
- header_ASDF: Z
- cookie_user: nuster
- param_type: Y
- body: (empty)

默认key产生GET\0http\0www.example.com\0/q?name=X&type=Y\0, 而key method.scheme.host.path.header_ASDF.cookie_user.param_type 则生成 GET\0http\0www.example.com\0/q\0Z\0uster\0Y\0.

\0是NULL字符

相同key的请求则会直接返回cache给客户端。

ttl TTL

设置缓存生存时间，过期后缓存会被删除。 可以使用 d, h, m and s。默认0秒.
如果不希望失效则设为0

code CODE1,CODE2...

默认只缓存200的响应，如果需要缓存其他的则可以添加，all会缓存任何状态码。

cache-rule only200
cache-rule 200and404 code 200,404
cache-rule all code all

disk MODE

定义缓存持久模式

- off: 默认模式，仅保存在内存
- only: 不保存在内存，仅保存在硬盘
- sync: 保存到内存和硬盘后返回给客户端
- async: 保存到内存后立即换回给客户的，内存数据会由master进程在一定时间后保存至硬盘

if|unless condition

定义ACL条件

ACL分别在请求阶段和响应阶段执行。

当下述条件满足时，会进行缓存：

1. 在请求阶段ACL为真
2. 请求阶段ACL为假，但是响应阶段ACL为真

当使用否定的ACL或者某些样本获取方法时，需要特别注意

比如

1. 缓存以 /img/开头的请求
nuster rule img if { path_beg /img/ }

请求阶段要么为真要么为假，因为在响应阶段无法获取path所以永远为假。

2. 缓存响应的http头部 Content-Type 为 image/jpeg
nuster rule jpeg if { res.hdr(Content-Type) image/jpeg }

因为在请求阶段无法获取res.hdr所以永远为假，在响应阶段要么为真要么为假。

3. 以 /img/开头，并且响应头 Content-Type 为image/jpeg时缓存

如果定义为下面的规则，则不会成功：

nuster rule img if { path_beg /img/ } { res.hdr(Content-Type) image/jpeg }

因为在响应阶段无法获取path所以永远为假，而在请求阶段无法获取res.hdr所以永远为假，那么这个

CL就永远无法匹配。

需要如下来定义：

```
http-request set-var(txn.pathImg) path
acl pathImg var(txn.pathImg) -m beg /img/
acl resHdrCT res.hdr(Content-Type) image/jpeg
nuster rule r3 if pathImg resHdrCT
```

4. 另一个例子，缓存所有不以 /api/ 开头的请求

下面不正确：

```
acl NoCache path_beg /api/
nuster rule r3 if !NoCache
```

因为虽然在响应阶段path并不存在，所以NoCache永远为假，而 !NoCache 为真，所有的请求都会缓存。

需要改成:

```
http-request set-var(txn.path) path
acl NoCache var(txn.path) -m beg /api/
nuster rule r1 if !NoCache
```

会添加一些新的样本获取方法来简化这些操作。

详见HAProxy configuration的**7. Using ACLs and fetching samples**

Cache

nuster也可以用作类似Varnish或者Nginx那样的HTTP缓存服务器，来缓存动态或者静态的HTTP资。

出了HAProxy的SSL, HTTP, HTTP2, 重写重定向，增删改Header等等，还提供了下面的功能。

缓存管理

缓存可以通过uri定义一个endpoint并发送HTTP请求来进行管理。

定义并且开启

```
nuster cache on uri /nuster/cache
```

基本用法

```
curl -X POST -H "X: Y" http://127.0.0.1/nuster/cache
```

记得进行访问控制

缓存开启关闭

rule可以通过manager uri动态开启关闭，关闭的rule不会再进行匹配。

headers

header	value	description
state	enable	enable rule
	disable	disable rule
name	rule NAME	the rule to be enabled/disabled
	proxy NAME	all rules belong t proxy NAME
	*	all rules

相同name的rule都会被开启关闭。

Examples

- 关闭rule r1

curl -X POST -H "name: r1" -H "state: disable" http://127.0.0.1/nuster/cache

- 关闭backend app1b的所有rule

curl -X POST -H "name: app1b" -H "state: disable" http://127.0.0.1/nuster/cache

- 开启所有的rule

curl -X POST -H "name: *" -H "state: enable" http://127.0.0.1/nuster/cache

缓存生存时间

更改缓存TTL，只会影响后续的新缓存，不会影响已经存在的缓存。

headers

header	value	description
ttl	new TTL	see ttl in nuster rule
name	rule NAME	the rule to be changed
	proxy NAME	all rules belong t proxy NAME
	*	all rules

Examples

```
curl -X POST -H "name: r1" -H "ttl: 0" http://127.0.0.1/nuster/cache
curl -X POST -H "name: r2" -H "ttl: 2h" http://127.0.0.1/nuster/cache
```

同时设置state和ttl

同时设置state和ttl

```
curl -X POST -H "name: r1" -H "ttl: 0" -H "state: enabled" http://127.0.0.1/nuster/cache
```

缓存清除

There are several ways to purge cache by making HTTP PURGE requests to the manager uri d fined by uri.

You can define customized http method using purge-method MYPURGE other than the defau t PURGE in case you need to forward PURGE to backend servers.

删除一个特定URL

```
curl -XPURGE https://127.0.0.1/imgs/test.jpg
```

生成key GET.scheme.host.uri, 并删除那个key。

默认key 包含Host, 如果缓存时用了http://example.com/test 而在localhost删除是需要Host header:

```
curl -XPURGE -H "Host: example.com" http://127.0.0.1/test
```

通过name删除

可以通过带上name header来 PURGE

headers

header	value	description
name	nuster rule NAME	caches belong to rule ${NAME} will be purged
	proxy NAME	caches belong to proxy ${NAME}
	*	all caches

Examples

```
# 删除所有缓存
curl -X PURGE -H "name: *" http://127.0.0.1/nuster/cache
# 删除backend applb的所有缓存
curl -X PURGE -H "name: app1b" http://127.0.0.1/nuster/cache
# 删除所有rule r1生成的缓存
curl -X PURGE -H "name: r1" http://127.0.0.1/nuster/cache
```

通过host删除

通过带上x-hostheader来删除所有属于这个host的缓存。

headers

header	value	description
x-host	HOST	the ${HOST}

Examples

curl -X PURGE -H "x-host: 127.0.0.1:8080" http://127.0.0.1/nuster/cache

通过path删除

默认情况下，query部分也包含在key中，所以相同的path不同的query会产生不同的缓存。

比如nuster rule imgs if { path_beg /imgs/ }, 然后请求

curl https://127.0.0.1/imgs/test.jpg?w=120&h=120
curl https://127.0.0.1/imgs/test.jpg?w=180&h=180

会生成两个缓存，因为query不一样。

如果要删除这些缓存，可以

如果知道所有的query，那么可以一个一个删除

curl -XPURGE https://127.0.0.1/imgs/test.jpg?w=120&h=120
curl -XPURGE https://127.0.0.1/imgs/test.jpg?w=180&h=180

大多数情况下不知道所有的query

如果query部分不重要，则可以从key里面删除query

定义nuster rule imgs key method.scheme.host.path if { path_beg /imgs }, 这样的话只会生成一缓存，那么就可以不用query删除缓存

curl -XPURGE https://127.0.0.1/imgs/test.jpg

大多数情况需要query

通过rule name删除

curl -X PURGE -H "name: imgs" http://127.0.0.1/nuster/cache

但是如果rule被定义成了 nuster rule static if { path_beg /imgs/ /css/ }，则无法只删除imgs

因此，可以通过path删除

headers

header	value	description
path	PATH	caches with ${PATH will be purged
x-host	HOST	and host is ${HOST}

Examples

删除所有path是/imgs/test.jpg的缓存
curl -X PURGE -H "path: /imgs/test.jpg" http://127.0.0.1/nuster/cache

```
# 删除所有path是/imgs/test.jpg 并且host是127.0.0.1:8080的缓存
curl -X PURGE -H "path: /imgs/test.jpg" -H "x-host: 127.0.0.1:8080" http://127.0.0.1/nuster/cache
```

通过正则删除

也可以通过正则删除，所有匹配正则的缓存将被删除。

headers

header	value	description
regex	REGEX	caches which pat match with ${REGEX} will be purged
x-host	HOST	and host is ${HOST}

Examples

```
# 删除所有 /imgs 开头 .jpg结尾的缓存
curl -X PURGE -H "regex: ^/imgs/.*\.jpg$" http://127.0.0.1/nuster/cache
#delete all caches which path starts with /imgs and ends with .jpg and belongs to 127.0.0.1:80
curl -X PURGE -H "regex: ^/imgs/.*\.jpg$" -H "127.0.0.1:8080" http://127.0.0.1/nuster/cache
```

PURGE 注意事项

1. **开启访问控制**

2. 如果有多个header，按照 name, path & host, path, regex & host, regex, host的顺序处理

curl -XPURGE -H "name: rule1" -H "path: /imgs/a.jpg": purge by name

3. 如果有重复的header，处理第一个

curl -XPURGE -H "name: rule1" -H "name: rule2": purge by rule1

4. regex **不是 glob**

比如 /imgs下的.jpg文件是^/imgs/.*\.jpg$ 而不是 /imgs/*.jpg

5. 通过rule name或proxy name删除缓存时，需要注意这两种方法只在当前进程有效。如果重启了程则无法通过这两种方法删除缓存文件，因为rule name信息和proxy name信息并没有保存在缓存件中。

6. 只有disk load结束后才能通过host or path or regex 来删除缓存文件。是否已经load结束可以查stats URL。

缓存统计

可以通过GET uri定义的endpoint来获取缓存统计信息。

Eanble and define the endpoint

```
nuster cache on uri /nuster/cache
```

Usage

curl http://127.0.0.1/nuster/cache

Output

- used_mem: http缓存使用的内存，不包括overhead
- req_total: 开启了cache的所有的backend的总请求数，不包含那些没有cache的backend请求数
- req_hit: cache击中数
- req_fetch: 从后端取得数量
- req_abort: 中断的请求

NoSQL

nuster也可以用作RESTful NoSQL缓存服务器, 用HTTP POST/GET/DELETE 来 添加/取得/删除 Key/alue.

基本操作

Set

```
curl -v -X POST -d value1 http://127.0.0.1:8080/key1
curl -v -X POST --data-binary @icon.jpg http://127.0.0.1:8080/imgs/icon.jpg
```

Get

curl -v http://127.0.0.1:8080/key1

Delete

curl -v -X DELETE http://127.0.0.1:8080/key1

Response

Check status code.

- 200 OK
  - POST/GET: 成功
  - DELETE: 总是
- 400 Bad request
  - 空值
  - 不正确的acl, rules, etc

- 404 Not Found
  - POST: rule tests失败
  - GET: not found
- 405 Method Not Allowed
  - 其他的methods
- 500 Internal Server Error
  - 发生未知错误
- 507 Insufficient Storage
  - 超过data-size

分用户的data

通过在key里加入header, cookie等等，可以将不同的用户数据存到相同的路劲。

```
nuster rule r1 key method.scheme.host.uri.header_userId if { path /mypoint }
nuster rule r2 key method.scheme.host.uri.cookie_sessionId if { path /mydata }
```

Set

```
curl -v -X POST -d "333" -H "userId: 1000" http://127.0.0.1:8080/mypoint
curl -v -X POST -d "555" -H "userId: 1001" http://127.0.0.1:8080/mypoint

curl -v -X POST -d "userA data" --cookie "sessionId=ijsf023xe" http://127.0.0.1:8080/mydata
curl -v -X POST -d "userB data" --cookie "sessionId=rosre329x" http://127.0.0.1:8080/mydata
```

Get

```
curl -v http://127.0.0.1:8080/mypoint
< 404 Not Found

curl -v -H "userId: 1000" http://127.0.0.1:8080/mypoint
< 200 OK
333

curl -v --cookie "sessionId=ijsf023xe" http://127.0.0.1:8080/mydata
< 200 OK
userA data
```

客户端

支持任何支持HTTP的客户端，库: curl, postman, python requests, go net/http, etc.

硬盘持久化

配置文件

```
global
    master-worker
    nuster cache on data-size 10m dir /tmp/cache
    nuster nosql on data-size 10m dir /tmp/nosql
backend be
    nuster cache on
    nuster rule off   disk off   ttl 1m if { path_beg /disk-off }
    nuster rule only  disk only  ttl 1d if { path_beg /disk-only }
    nuster rule sync  disk sync  ttl 1h if { path_beg /disk-sync }
    nuster rule async disk async ttl 2h if { path_beg /disk-async }
    nuster rule others ttl 100
```

1. /disk-off 仅保存在内存
2. /disk-only 仅保存在硬盘
3. /disk-sync 保存至内存和硬盘后返回给客户端
4. /disk-async 保存至内存后立即换回给客户端，内存数据会在一定时间后被缓存至硬盘
5. 其他的所有请求都仅保存在内存

Sample fetches

Nuster 加入了一些新的sample fetches

nuster.cache.hit: boolean

表示是否是HIT缓存，可以像如下使用

```
http-response set-header x-cache hit if { nuster.cache.hit }
```

FAQ

无法启动，报错: not in master-worker mode

在global 添加 master-worker 或者启动时使用-W参数。

如何调试?

在global添加debug， 或者带-d启动nuster

nuster相关的调试信息以[nuster开头

如何缓存POST请求?

添加option http-buffer-request

如果自定义了key的话需要使用body关键字

请求body可能不完整，详见HAProxy configuration 的 **option http-buffer-request**小节

另外可以为post请求单独设置一个后端

如何做访问控制?

类似

```
acl network_allowed src 127.0.0.1
acl purge_method method PURGE
http-request deny if purge_method !network_allowed
```

如何开启HTTP2?

```
bind :443 ssl crt pub.pem alpn h2,http/1.1
```

Example

```
global
   nuster cache on data-size 100m
   nuster nosql on data-size 100m
   master-worker # v3
   # daemon
   # debug
defaults
   retries 3
   option redispatch
   timeout client  30s
   timeout connect 30s
   timeout server  30s
frontend web1
   bind *:8080
   mode http
   acl pathPost path /search
   use_backend app1a if pathPost
   default_backend app1b
backend app1a
   balance roundrobin
   # mode must be http
   mode http

   # http-buffer-request must be enabled to cache post request
   option http-buffer-request

   acl pathPost path /search

   # enable cache for this proxy
   nuster cache

   # cache /search for 120 seconds. Only works when POST/PUT
   nuster rule rpost key method.scheme.host.uri.body ttl 120 if pathPost
```

```
    server s1 10.0.0.10:8080
backend app1b
    balance     roundrobin
    mode http

    nuster cache on

    # cache /a.jpg, not expire
    acl pathA path /a.jpg
    nuster rule r1 ttl 0 if pathA

    # cache /mypage, key contains cookie[userId], so it will be cached per user
    acl pathB path /mypage
    nuster rule r2 key method.scheme.host.path.delimiter.query.cookie_userId ttl 60 if pathB

    # cache /a.html if response's header[cache] is yes
    http-request set-var(txn.pathC) path
    acl pathC var(txn.pathC) -m str /a.html
    acl resHdrCache1 res.hdr(cache) yes
    nuster rule r3 if pathC resHdrCache1

    # cache /heavy for 100 seconds if be_conn greater than 10
    acl heavypage path /heavy
    acl tooFast be_conn ge 100
    nuster rule heavy ttl 100 if heavypage tooFast

    # cache all if response's header[asdf] is fdsa
    acl resHdrCache2 res.hdr(asdf)  fdsa
    nuster rule resCache ttl 0 if resHdrCache1

    server s1 10.0.0.10:8080

frontend web2
    bind *:8081
    mode http
    default_backend app2
backend app2
    balance     roundrobin
    mode http

    # disable cache on this proxy
    nuster cache off
    nuster rule all

    server s2 10.0.0.11:8080

frontend nosql_fe
    bind *:9090
    default_backend nosql_be
backend nosql_be
    nuster nosql on
    nuster rule r1 ttl 3600
```

Contributing

- Join the development
- Give feedback
- Report issues
- Send pull requests
- Spread nuster

License