Using NGINX as an Effective and Highly Available Content Cache

Kevin Jones
Technical Solutions Architect
@webopsx
Agenda

• Quick intro to…
  • NGINX
  • Content Caching

• Caching with NGINX
  • How caching functionality works
  • How to enable basic caching

• Advanced caching with NGINX
  • How to increase availability using caching
  • When and how to enable micro-caching
  • How to fine tune the cache
  • How to architect for high availability
  • Various configuration tips and tricks!
  • Various examples!
NGINX

Solves complexity...

Web Server
Reverse Proxy
Load Balancer
Content Cache
Streaming Media Server

MORE INFORMATION AT NGINX.COM
350 million

total sites and counting...

running on NGINX
53% of the Top 10,000 most visited websites
36% of all instances on Amazon Web Services

Source: W3Techs December 2013 Web Server Survey
Contexts, Directives and Parameters... Oh my.
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;
events {
    worker_connections 1024;
}
http {
    include /etc/nginx/mime.types;
default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] 
        "$request" $status $body_bytes_sent "$http_referer" 
        "$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    upstream api-backends {
        server 10.0.1.11:8080;
        server 10.0.1.12:8080;
    }

    server {
        listen 10.0.1.10:80;
        server_name example.com;

        location / {
            root /usr/share/nginx/html;
            index index.html index.htm;
        }

        location ^~ /api {
            proxy_pass http://api-backends;
        }
    }
}
include /path/to/more/virtual_servers/*.conf;
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" 
    '$status $body_bytes_sent "$http_referer" 
    '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  /var/log/nginx/access.log  main;
    upstream api-backends {
        server 10.0.1.11:8080;
        server 10.0.1.12:8080;
    }
    server {
        listen 10.0.1.10:80;
        server_name example.com;
        location / {
            root /usr/share/nginx/html;
            index index.html index.htm;
        }
        location ^~ /api {
            proxy_pass http://api-backends;
        }
    }
}

include /path/to/more/virtual_servers/*.conf;
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
  worker_connections 1024;
}

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

  log_format main '
    $remote_addr - $remote_user [$time_local] "$request"
    $status $body_bytes_sent "$http_referer"
    "$http_user_agent" "$http_x_forwarded_for"';

  access_log /var/log/nginx/access.log main;

  upstream api-backends {
    server 10.0.1.11:8080;
    server 10.0.1.12:8080;
  }

  server {
    listen 10.0.1.10:80;
    server_name example.com;

    location / {
      root /usr/share/nginx/html;
      index index.html index.htm;
    }

    location ~ /api {
      proxy_pass http://api-backends;
    }

    include /path/to/more/virtual_servers/*.conf;
  }
}
Variables
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
  worker_connections 1024;
}

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

  log_format main '$remote_addr - $remote_user [$time_local] "$request"
          $status $body_bytes_sent "$http_referer"
          "$http_user_agent" "$http_x_forwarded_for"';

  access_log /var/log/nginx/access.log main;

  upstream api-backends {
    server 10.0.1.11:8080;
    server 10.0.1.12:8080;
  }

  server {
    listen 10.0.1.10:80;
    server_name example.com;

    location / {
      root /usr/share/nginx/html;
      index index.html index.htm;
    }

    location ^~ /api {
      proxy_pass http://api-backends;
    }
  }
}

include /path/to/more/virtual_servers/*.conf;
http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
    ' $status $body_bytes_sent "$http_referer" '
    '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    map $http_user_agent $dynamic {
        "~*Mobile" mobile.example.com;
        default desktop.example.com;
    }

    server {
        listen 10.0.1.10:80;
        server_name example.com;

        location / {
            root /usr/share/nginx/html;
            index index.html index.htm;
        }

        location ^~ /api {
            proxy_pass http://$dynamic;
        }
    }

    include /path/to/more/virtual_servers/*.conf;
}

nginx.org/en/docs/varindex.html
Layer 7 Request Routing
http {
...

server {

    map $http_user_agent $blocked {
        ~badbot        '1';
        default        '';
    }

    if ( $blocked ) {
        return 444;
    }

    listen 10.0.1.10:80;
    server_name website.com *.example.com;

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
    }

    location ^~ /api {
        proxy_pass http://api-backends;
    }

}

include /path/to/more/virtual_servers/*.conf;

}
The Basics of Content Caching
Client initiates request (e.g. GET /file)

Proxy Cache determines if response is already cached if not proxy cache will fetch from the origin server

Origin Server serves response along with all cache control headers (e.g. Cache-Control, Etag, etc.)

Proxy Cache caches the response and serves it to the client
Cache Headers

- **Cache-Control** - used to specify directives for caching mechanisms in both, requests and responses. (e.g. Cache-Control: max-age=600 or Cache-Control: no-cache)

- **Expires** - contains the date/time after which the response is considered stale. If there is a Cache-Control header with the "max-age" or "s-max-age" directive in the response, the Expires header is ignored. (e.g. Expires: Wed, 21 Oct 2015 07:28:00 GMT)

- **Last-Modified** - contains the date and time at which the origin server believes the resource was last modified. HTTP dates are always expressed in GMT, never in local time. Less accurate than the ETag header (e.g. Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT)

- **ETag** - is an identifier (or fingerprint) for a specific version of a resource. (e.g. ETag: “58efdcd0-268")
Content caching with NGINX is simple.
proxy_cache_path

Syntax:  \texttt{proxy_cache_path \ path \ [levels=levels] \ [use_temp_path=on|off] \ [keys_zone=name:size] \ [inactive=time]  \\ [max_size=size] \ [manager_files=number] \ [manager_sleep=time] \ [manager_threshold=time]  \\ [loader_files=number] \ [loader_sleep=time] \ [loader_threshold=time] \ [purger=on|off] \ [purger_files=number]  \\ [purger_sleep=time] \ [purger_threshold=time];}

Default:  -

Context:  http

Definition:  Sets the path and other parameters of a cache. Cache data are stored in files. The file name in a cache is a result of applying the MD5 function to the cache key.

```toml
http {

  proxy_cache_path /tmp/nginx/micro_cache/ levels=1:2 keys_zone=large_cache:10m  
  max_size=300g inactive=14d;

  ...
}
```
proxy_cache_key

**Syntax:**  \texttt{proxy_cache_key \textit{string};}

**Default:**  \texttt{proxy_cache_key $scheme$proxy_host$request_uri;}

**Context:**  http, server, location

**Definition:**  Defines a key for caching. Used in the proxy_cache_path directive.

```nginx
server {
    proxy_cache_key $scheme$proxy_host$request_uri$cookie_userid;
    ...
}
```
proxy_cache

Syntax:  `proxy_cache zone | off;`

Default:  `proxy_cache off;`

Context:  `http, server, location`

Definition:  Defines a shared memory zone used for caching. The same zone can be used in several places.

```
location ^~ /video {
    ...
    proxy_cache large_cache;
}
```
proxy_cache_valid

Syntax:  proxy_cache_valid [code...] time;

Default:  -

Context:  http, server, location

Definition:  Sets caching time for different response codes.
http {

    proxy_cache_path /tmp/nginx/cache levels=1:2 keys_zone=cache:10m
    max_size=100g inactive=7d use_temp_path=off;

    ...

    server {
        ...

        location / {
            ...
            proxy_pass http://backend.com;
        }

        location ^~ /images {
            ...
            proxy_cache cache;
            proxy_cache_valid 200 301 302 12h;
            proxy_pass http://images.origin.com;
        }
    }
}

Basic Caching
1. HTTP Request: GET /images/hawaii.jpg

Cache Key: http://origin/images/hawaii.jpg
md5 hash: 51b740d1ab03f287d46da45202c84945

2. NGINX checks if hash exists in memory. If it does not the request is passed to the origin server.

3. Origin server responds

4. NGINX caches the response to disk and places the hash in memory

5. Response is served to client
NGINX Processes

```
# ps aux | grep nginx
root     14559  0.0  0.1  53308  3360 ?        Ss   Apr12   0:00 nginx: master process /usr/sbin/nginx
       -c /etc/nginx/nginx.conf
nginx   27880  0.0  0.1  53692  2724 ?        S    00:06   0:00 nginx: worker process
nginx   27881  0.0  0.1  53692  2724 ?        S    00:06   0:00 nginx: worker process
nginx   27882  0.0  0.1  53472  2876 ?        S    00:06   0:00 nginx: cache manager process
nginx   27883  0.0  0.1  53472  2552 ?        S    00:06   0:00 nginx: cache loader process
```

- **Cache Manager** - activated periodically to check the state of the cache. If the cache size exceeds the limit set by the `max_size` parameter to the `proxy_cache_path` directive, the cache manager removes the data that was accessed least recently, as well as the cache considered inactive.

- **Cache Loader** - runs only once, right after NGINX starts. It loads metadata about previously cached data into the shared memory zone.
Caching is not just for HTTP

Tip: NGINX can also be used to cache other backends using their unique cache directives. (e.g. fastcgi_cache, uwsgi_cache and scgi_cache)

Alternatively, NGINX can also be used to retrieve content directly from a memcached server.
Initial... Tips and Tricks!
Tip: The more relevant information in your log the better. When troubleshooting you can easily add the proxy cache KEY to the log_format for debugging. For a list of all variables see the “Alphabetical index of variables” on nginx.org.
Add Response Headers

**Tip:** Using the `add_header` directive you can add useful HTTP response headers allowing you to debug your NGINX deployment rather easily.

```nginx
server {
    ...  
    # add HTTP response headers
    add_header CC-X-Request-ID $request_id;
    add_header X-Cache-Status $upstream_cache_status;
}
```
Cache Status

- **MISS** – The response was not found in the cache and so was fetched from an origin server. The response might then have been cached.

- **BYPASS** – The response was fetched from the origin server instead of served from the cache because the request matched a proxy_cache_bypass directive. The response might then have been cached.

- **EXPIRED** – The entry in the cache has expired. The response contains fresh content from the origin server.

- **STALE** – The content is stale because the origin server is not responding correctly, and proxy_cache_use_stale was configured.

- **UPDATING** – The content is stale because the entry is currently being updated in response to a previous request, and proxy_cache_use_stale updating is configured.

- **REVALIDATED** – The proxy_cache_revalidate directive was enabled and NGINX verified that the current cached content was still valid (ETag, If-Modified-Since or If-None-Match).

- **HIT** – The response contains valid, fresh content direct from the cache.
Using cURL to Debug…

Tip: Use cURL or Chrome developer tools to grab the request ID or other various headers useful for debugging.

```
# curl -I 127.0.0.1/images/hawaii.jpg
HTTP/1.1 200 OK
Server: nginx/1.11.10
Date: Wed, 19 Apr 2017 22:20:53 GMT
Content-Type: image/jpeg
Content-Length: 21542868
Connection: keep-alive
Last-Modified: Thu, 13 Apr 2017 20:55:07 GMT
ETag: "58efe5ab-148b7d4"
OS-X-Request-ID: 1e7ae2cf83732e8859bc3e38df912ed1
CC-X-Request-ID: d4a5f7a8d25544b1409c351a22f42960
X-Cache-Status: HIT
Accept-Ranges: bytes
```
Troubleshooting the Proxy Cache

Tip: A quick and easy way to determine the hash of your cache key can be accomplished using echo, pipe and md5sum

```
# grep -ri d4a5f7a8d25544b1409c351a22f42960 /var/log/nginx/adv_access.log
rid="d4a5f7a8d25544b1409c351a22f42960" pck="http://origin/images/hawaii.jpg"
site="webopsx.com" server="localhost" dest_port="80" dest_ip="127.0.0.1" ...

# echo -n "http://origin/images/hawaii.jpg" | md5sum
51b740d1ab03f287d46da45202c84945 -

# tree /tmp/nginx/micro_cache/5/94/
  /tmp/nginx/micro_cache/5/94/
    51b740d1ab03f287d46da45202c84945

0 directories, 1 file
```
Cache Contents

```
# head -n 14 /tmp/nginx/micro_cache/5/94/51b740d1ab03f287d46da45202c84945
??X?X??Xb?!bv?"58efe5ab-148b7d4"
KEY: http://origin/images/hawaii.jpg
HTTP/1.1 200 OK
Server: nginx/1.11.10
Date: Wed, 19 Apr 2017 23:51:38 GMT
Content-Type: image/jpeg
Content-Length: 21542868
Last-Modified: Thu, 13 Apr 2017 20:55:07 GMT
Connection: keep-alive
ETag: "58efe5ab-148b7d4"
OS-X-Request-ID: 1e7ae2cf83732e8859bc3e38df912ed1
Accept-Ranges: bytes

?wExifII>(i?Nl?0230??HH??
```
Micro-Caching

“Size matters not.”
Types of Content

Static Content
- Images
- CSS
- Simple HTML

Dynamic Content
- Blog Posts
- Status
- API Data (Maybe?)

User Content
- Shopping Cart
- Unique Data
- Account Data

Easy to cache

Micro-cacheable!

Cannot Cache
http {
  upstream backend {
    keepalive 20;
    server 127.0.0.1:8080;
  }
  proxy_cache_path /var/nginx/micro_cache levels=1:2 keys_zone=micro_cache:10m max_size=100m inactive=600s;
  ...
  server {
    listen 80;
    ...
    proxy_cache micro_cache;
    proxy_cache_valid any 1s;
    location / {
      proxy_http_version 1.1;
      proxy_set_header Connection "";
      proxy_set_header Accept-Encoding "";
      proxy_pass http://backend;
    }
  }
}
proxy_cache_lock

Syntax:  proxy_cache_lock on | off;

Default:  proxy_cache_lock off;

Context:  http, server, location

Definition:  When enabled, only one request at a time will be allowed to populate a new cache element identified according to the proxy_cache_key directive by passing a request to a proxied server.

Other requests of the same cache element will either wait for a response to appear in the cache or the cache lock for this element to be released, up to the time set by the proxy_cache_lock_timeout directive.

Related:  See the following for tuning…
  •  proxy_cache_lock_age,
  •  proxy_cache_lock_timeout
proxy_cache_use_stale

Syntax: `proxy_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | off ...;`

Default: `proxy_cache_use_stale off;`

Context: `http, server, location`

Definition: Determines in which cases a stale cached response can be used during communication with the proxied server.

```nginx
location /contact-us {

    ... 
    proxy_cache_use_stale error timeout updating http_500 http_502 http_503 http_504;

}
```
http {
    upstream backend {
        keepalive 20;
        server 127.0.0.1:8080;
    }
    proxy_cache_path /var/nginx/micro_cache levels=1:2 keys_zone=micro_cache:10m
    max_size=100m inactive=600s;
    ...
    server {
        listen 80;
        ...
        proxy_cache micro_cache;
        proxy_cache_valid any 1s;
        proxy_cache_lock on;
        proxy_cache_use_stale updating;
        location / {
            ...
            proxy_http_version 1.1;
            proxy_set_header Connection "";
            proxy_set_header Accept-Encoding "";
            proxy_pass http://backend;
        }
    }
}

Final optimization
proxy_cache_background_update

Syntax: proxy_cache_background_update on | off;

Default: proxy_cache_background_update off;

Context: http, server, location

Definition: Allows starting a background subrequest to update an expired cache item, while a stale cached response is returned to the client. Note that it is necessary to allow the usage of a stale cached response when it is being updated.

```location / {
    ...
    proxy_cache_background_update on;
    proxy_cache_lock on;
    proxy_cache_use_stale updating;
}
```
Further Tuning and Optimization
proxy_cache_revalidate

Syntax: \texttt{proxy\_cache\_revalidate} on | off;

Default: \texttt{proxy\_cache\_revalidate} off;

Context: http, server, location

Definition: Enables revalidation of expired cache items using conditional GET requests with the “If-Modified-Since” and “If-None-Match” header fields.
proxy_cache_min_uses

Syntax: `proxy_cache_min_uses number;`

Default: `proxy_cache_min_uses 1;`

Context: `http, server, location`

Definition: Sets the number of requests after which the response will be cached. This will help with disk utilization and hit ratio of your cache.

```plaintext
location ~* /legacy {
    ...
    proxy_cache_min_uses 5;
}
```
proxy_cache_methods

Syntax:  `proxy_cache_methods GET | HEAD | POST ...;

Default: `proxy_cache_methods GET HEAD;

Context: http, server, location

Definition: NGINX only caches GET and HEAD request methods by default. Using this directive you can add additional methods.

If you plan to add additional methods consider updating the cache key to include the $request_method variable if the response will be different depending on the request method.

```
location ~* /data {
    ...
    proxy_cache_methods GET HEAD POST;
}
```
proxy_buffering

Syntax: proxy_buffering on | off;

Default: proxy_buffering on;

Context: http, server, location

Definition: Enables or disables buffering of responses from the proxied server.

When buffering is enabled, nginx receives a response from the proxied server as soon as possible, saving it into the buffers set by the proxy_buffer_size and proxy_buffers directives. If the whole response does not fit into memory, a part of it can be saved to a temporary file on the disk.

When buffering is disabled, the response is passed to a client synchronously, immediately as it is received.
Override Cache-Control headers

**Tip:** By default NGINX will honor all Cache-Control headers from the origin server, in turn not caching responses with Cache-Control set to Private, No-Cache, No-Store or with Set-Cookie in the response header.

Using `proxy_ignore_headers` you can disable processing of certain response header fields from the proxied server.

```nginx
location ^~ /wordpress {
    ...
    proxy_cache cache;
    proxy_ignore_headers Cache-Control;
}
```
Can I Punch Through the Cache?

**Tip:** If you want to disregard the cache and go straight to the origin for a response, you can use the `proxy_cache_bypass` directive.

```nginx
location / {
    ...
    proxy_cache cache;
    proxy_cache_bypass $cookie_nocache $arg_nocache $http_cache_bypass;
}
```
proxy_cache_purge

Syntax:  proxy_cache_methods string ...;

Default:  -

Context:  http, server, location

Definition:  Defines conditions under which the request will be considered a cache purge request. If at least one value of the string parameters is not empty and is not equal to “0” then the cache entry with a corresponding cache key is removed.

The result of successful operation is indicated by returning the 204 (No Content) response.

Note:  NGINX Plus only feature
Example Cache Purge Configuration

**Tip:** Using NGINX Plus, you can issue unique request methods to invalidate the cache

```nginx
proxy_cache_path /tmp/cache keys_zone=mycache:10m levels=1:2 inactive=60s;

map $request_method $purge_method {
    PURGE 1;
    default 0;
}

server {
    listen 80;
    server_name www.example.com;

    location / {
        proxy_pass http://localhost:8002;
        proxy_cache mycache;

        proxy_cache_purge $purge_method;
    }
}
```

- **dynamically set a variable**
- **used later in the configuration**
Useful Examples
Split the Cache Across HDDs

```http
http {
    proxy_cache_path /path/to/hdd1 levels=1:2 keys_zone=my_cache_hdd1:10m
    max_size=10g inactive=60m use_temp_path=off;

    proxy_cache_path /path/to/hdd2 levels=1:2 keys_zone=my_cache_hdd2:10m
    max_size=10g inactive=60m use_temp_path=off;

    split_clients $request_uri $my_cache {
        50% "my_cache_hdd1";
        50% "my_cache_hdd2";
    }

    server {
        ...
        location / {
            proxy_cache $my_cache;
            proxy_pass http://my_upstream;
        }
    }
}
```

- **set a variable using a hash**: `split_clients $request_uri $my_cache {`.
- **used later in the configuration**: `proxy_cache $my_cache;`
Using NGINX for Byte Range Caching

1. Client requests 100 bytes, starting at offset 150

GET Range = 150-249

Caching slice: 100

2. NGINX retrieves enclosing segments

GET range = 100-199
GET range = 200-299

3. Each segment is cached separately

4. NGINX assembles response from the cached segments

/cache/XYZ (slice for 100-199)
/cache/ABC (slice for 200-299)
Slice Caching Example

**Tip:** Ensure the caching of slice requests by adding the `$slice_range` variable to your `proxy_cache_key` and be sure to add HTTP 206 responses to the `proxy_cache_valid` directive. Also it's important to know that in order for NGINX to send byte range requests the connection has to be made over HTTP 1.1.

```nginx
http {
    proxy_cache_path /tmp/mycache keys_zone=mycache:10m;

    server {
        listen 80;

        proxy_cache mycache;

        slice  1m;
        proxy_cache_key $host$uri$is_args$args$slice_range;
        proxy_set_header Range $slice_range;
        proxy_http_version 1.1;
        proxy_cache_valid  200 206 1h;

        location / {
            proxy_pass http://origin.example.com;
        }
    }
}
```
Architecting for High Availability
Two Approaches

- Sharded (High Capacity)
- Shared (Replicated)
Tip: If your primary goal is to achieve high availability while minimizing load on the origin servers, this scenario provides a highly available shared cache. HA cluster should be Active/Passive configuration.
And Failover...

Tip: In the event of a failover there is no loss in cache and the origin does not suffer unneeded proxy requests.
proxy_cache_path /tmp/mycache keys_zone=mycache:10m;

server {
    listen 80;
    proxy_cache mycache;
    proxy_cache_valid 200 15s;

    location / {
        proxy_pass http://secondary;
    }
}

upstream secondary {
    server 192.168.56.11;         # secondary
    server 192.168.56.12 backup;  # origin
}
proxy_cache_path /tmp/mycache keys_zone=mycache:10m;

server {
    listen 80;
    proxy_cache mycache;
    proxy_cache_valid 200 15s;

    location / {
        proxy_pass http://origin;
    }
}

upstream origin {
    server 192.168.56.12; # origin
}
Sharding your Cache

Tip: If your primary goal is to create a very high-capacity cache, shard (partition) your cache across multiple servers. This in turn maximizes the resources you have while minimizing impact on your origin servers depending on the amount of cache servers in your cache tier.
**Hash Load Balancing**

**Tip:** Using the hash load balancing algorithm, we can specify the proxy cache key. This allows each resource to be cached on only one backend server.

```conf
upstream cache_servers {
    hash $scheme$proxy_host$request_uri consistent;

    server prod.cache1.host;
    server prod.cache2.host;
    server prod.cache3.host;
    server prod.cache4.host;
}
```
Combined Load Balancer and Cache

Tip: Alternatively, it is possible to consolidate the load balancer and cache tier into one with the use of various NGINX directives and parameters.
Tip: If needed, a “Hot Cache Tier” can be enabled on the load balancer layer which will give you the same high capacity cache and provide a high availability of specific cached resources.
Links

Documentation
• https://nginx.org
• https://nginx.com

Blogs
• https://www.nginx.com/blog/nginx-caching-guide/
• https://www.nginx.com/blog/benefits-of-microcaching-nginx/
• https://www.nginx.com/blog/shared-caches-nginx-plus-cache-clusters-part-1/
• https://www.nginx.com/blog/smart-efficient-byte-range-caching-nginx/

Webinar
• https://www.nginx.com/resources/webinars/content-caching-nginx-plus/
Thank You

https://www.linkedin.com/in/kevin-jones-19b17b47/
https://www.slideshare.net/KevinJones62
https://www.nginx.com/blog/author/kjones/

Kevin Jones
Technical Solutions Architect
NGINX Inc.

@webopsx