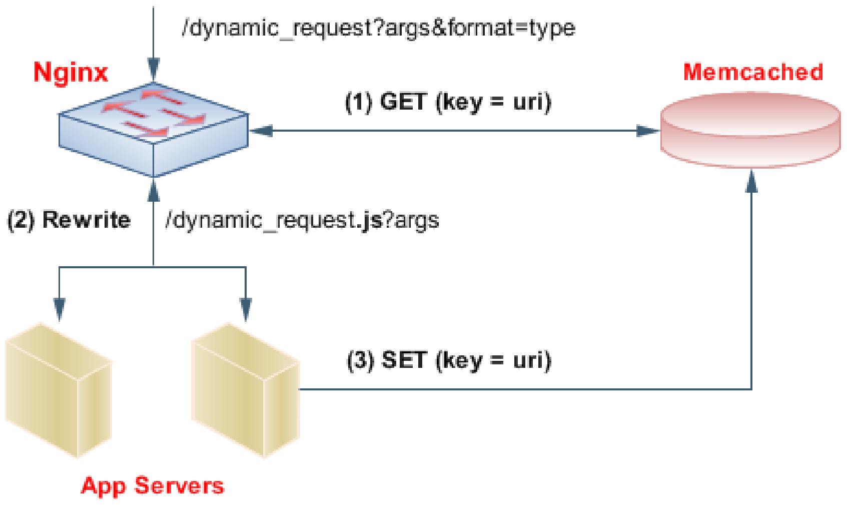
Nginx and Memcached, a 400% boost!

Nginx and Memcached, a 400% boost!

If web architectures, performance, or scalability are topics you would like to keep on top of (who doesn't!), then chances are, you've heard of Nginx ("engine x"). Originally developed by Igor Sysoev for rambler.ru (second largest Russian web-site), it is a high-performance HTTP server / reverse proxy known for its stability, performance, and ease of use. The great track record, a lot of great modules, and an active development community have rightfully earned it a steady uptick of users, and most recently, a notable mention in the Netcraft report.

Memcached module - an easy 4x speed multiplier

Memcached, the darling of every web-developer, is capable of turning almost any application into a speed-demon. Benchmarking one of my own Rails applications resulted in ~850 req/s on commodity, non-optimized hardware - more than enough in the case of this application. However, what if we took Mongrel out of the equation? Nginx, by default, comes prepackaged with the Memcached module, which allows us to bypass the Mongrel servers and talk to Memcached directly. Same hardware, and a quick test later: ~3,550 req/s, or almost a **400% improvement**! Not bad for a five minute tweak!



Think smart, forget cache invalidations

The only snag in our scheme for easy performance gains comes with the fact that more often than not, our application servers contain additional caching policies (read invalidations / authentication), and MIME type logic. The former, as recently documented by Tobias L??tke and Geoffrey Grosenbach, if properly thought through can be solved with some clever URL rewriting policies and automatic TTL timeouts. When implemented correctly, we could simply set the memcached key to be the full request URL, allowing us to completely bypass our app. servers.

MIME-type logic

MIME type magic can be as easy as complex as we wish. If you only serve one content type ('*text/html*', for example), the solution is simple:

> nginx-default.conf

```
location /dynamic_request {
  # Set default type to text/html
  default_type  text/html;

  # ...
}
```

Dynamic argument types, just for fun

However, if we want to serve multiple content-types, or perhaps even parameterize the request type in a query string, we've got some extra work to do. Not unlike any other HTTP server, Nginx checks the filetype extension at the end of every request path to determine the correct content-type header, a solution which unfortunately breaks down in majority of modern, URL friendly web-applications:

1. **GET /dynamic_request.js** - Content-Type = text/javascript
2. **GET /dynamic_request** - Content-Type = ?
3. **GET /dynamic_request?format=js** - Content-Type = ?

Case 1 is easily solved by Nginx directly. Case 2 is tricky, but can be solved via a 'default_type' line in the config as document above. And case 3 will require some additional logic - namely, we can hardcode a rule to rewrite our dynamic query string parameters to automagically add an extension to the path of each incoming request:

> nginx-rewrite.conf

```
location /dynamic_request {
   # append an extenstion for proper MIME type detection
          if ($args ~* format=json) { rewrite ^/dynamic_request/?(.*)$ /dynamic_request.js$1 break;
          if ($args ~* format=xml)  { rewrite ^/dynamic_request/?(.*)$ /dynamic_request.xml$1 break

          memcached_pass 127.0.0.1:11211;
          error_page 404 = @dynamic_request;
}
```

That should do the trick! Cache invalidations are handled, MIME types are served correctly, and our app. servers are bypassed in 95%+ of the cases. Instead, Nginx talks directly to Memcached and only proxies the cache misses - an easy 400% performance boost!