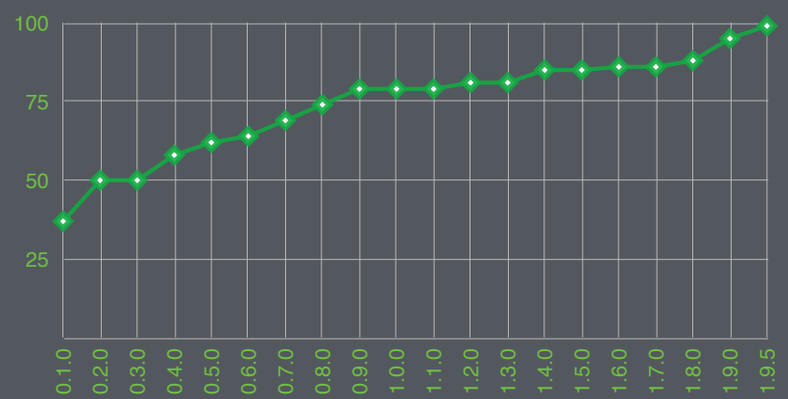
My name is Ruslan Ermilov and I’m a developer at NGINX.

This talk is about an ongoing project that will add support for dynamically loadable modules into NGINX.

Modules Growth



Since its public launch nginx had a notion of modules.
The first public version (0.1.0, released in October 2004) already had over 30 modules, including 22 http modules. (5 core, 9 event)
Today, after a decade of continuous development, the current version (1.9.5), which we encourage you to try, has 97 modules!

Anatomy of Modules

- 10 core — generic and OS-specific stuff
- 9 event
- 63 http
- 9 stream
- 7 mail

Let’s look under the hood. Core modules implement basic things such as

- generic types and objects - these are numbers, strings, arrays, lists, hashes, buffers, queues, and trees;
- they also provide memory management API (pool allocator, shared memory with slab allocator), locking primitives (such as mutexes, spinlocks, read-write locks), implement several checksumming and hashing algorithms, configuration file parsing and configuration management interfaces, logging (incl. syslog support), timekeeping, asynchronous resolver, connection management;
- they are responsible for OS-specific tasks such as file and socket I/O, process management, IPC, timers, CPU affinity, thread pools for offloading blocking I/O operations;

Anatomy of Modules

- 10 core — generic and OS-specific stuff
- 9 event — connection processing methods
- 63 http
- 9 stream
- 7 mail

event modules are responsible for connection processing; they implement epoll() support on Linux, kqueue() on BSD-like systems, /dev/poll (Solaris 7 11/99+ and some other UNIX derivatives), eventport (Solaris 10), and archaic methods such as select() and poll()
NGINX automatically chooses the most efficient method available for the platform

Anatomy of Modules

- 10 core — generic and OS-specific stuff
- 9 event — connection processing methods
- 63 http —HTTP (0.9-2.0) & reverse proxy server
- 9 stream — generic TCP proxy
- 7 mail — mail proxy

- http modules… it is what makes NGINX a good web server — as you may have heard already, NGINX is now the most used web server among the top 100k web servers in the world
- new to 1.9.0 release, the set of stream modules adds NGINX to the market of generic TCP proxy servers
- mail modules allow for proxying of IMAP/POP3/SMTP using NGINX

More Modules

- http://wiki.nginx.org/3rdPartyModules — 122 modules
- NGINX Plus — adds 12 modules + extends a lot

- over the years, the growing NGINX community has developed over ten dozens of modules!
- we know quite a few companies that write their own modules for NGINX
- and last but not least, our commercial product, NGINX+, adds another dozen of modules (these are modules for media streaming [hls & f4f aka hds], session logging, extended monitoring and status, upstream dynamic configuration & health checks, ntlm & least_time upstream balancers, request stickiness), to name a few. N+ also extends some existing modules, like cache purge support, limiting the max. number of connections to upstream servers [max_conns], or keeping upstream configuration up-to-date by tracking DNS changes [resolve]).

Static Modules

- --with-*foo*-module / --without-*foo*-module
- --add-module=/path/to*foo*

```
$ auto/configure --with-http_v2_module \
                 --with-http_ssl_module \
                 --add-module=nginx-rtmp-module-1.1.7
```


The set of modules that constitute an NGINX binary is currently fixed at compile time by the «configure» script options. Standard modules are included or excluded by «with and without module» options. External modules may be added with the —add-module options. Though the ability to compile in external modules appeared in the early versions of NGINX, the demand on dynamic module loading was low. The number of external modules was pretty moderate, and to be honest, implementing support for dynamic loading wasn’t a high priority task for Igor.

Fears & Excuses

- dlopen()
- https://en.wikipedia.org/wiki/Dependency_hell
- http://nginx.org/en/docs/control.html#upgrade

Nevertheless, it was considered. And while dlopen() is a mature system interface available in almost every modern UNIX-like OS, the dynamic loading can open a can of worms, cumulatively known as «Dependency hell». Some call it «DLL hell». One of the typical problems with dynamic loading is when two modules are built with different versions of the same library. This can manifest itself in random crashes or other hard to diagnose problems.
One of the little-known features that NGINX offers allows you to «upgrade its executable on-the-fly». Using this feature, it’s possible to run another nginx binary that was built with the different set of modules, yet without the interruption in service. It works well when you have sources for NGINX and sources of all modules that you need, and have a freedom of mixing them in a way YOU need it.

Demand on Dynamic Loading

- vendor packages
- NGINX Plus packages
- 3rd party modules

Unfortunately, this is not always an option. Vendors ship NGINX in binary packages, and they usually provide several of them, each with its own set of modules. For example, on Ubuntu 14.04 there are 5 base packages (nginx-core, nginx-light, nginx-full, nginx-extras, nginx-naxsi). N+ also comes with several packages. But what if you're not happy with a set of modules? You don't have the flexibility to just add another module to the crew, you have to recompile NGINX. This means you can't use the provided packages. As the number of 3rd party modules grew, and especially with the launch of N+ in August 2013, it became obvious that we should invest some time and allow modules to be loaded dynamically, without the need of recompiling NGINX.

Demand on Dynamic Loading

- the number of vendor packages is reduced
- other companies can supply their binary modules
- NGINX Plus can be run with custom modules

- This has a potential of reducing the number of packages that vendors ship, because individual modules can be shipped as distinct packages in addition to one base package
- because N+ is partly closed source, you can’t recompile it with the custom set of modules; if we support dynamic loading, we may allow certain modules that pass our integrity checks to be loaded with N+
- and like I mentioned earlier, there’s also a growing demand from other companies that are interested in either running N+ with their own modules, or to be able to supply their binary modules against the open source version of NGINX.

Building Dynamic Modules

```
$ procstat -v 36185 | grep / | grep r-x
  PID              START                END PRT  RES PRES REF SHD   FL TP PATH
36185          0x400000           0x499000 r-x  153  816   2   1 CN-- vn objs/nginx
36185        0x800698000        0x8006b3000 r-x   22   23  32   0 CN-- vn /libexec/ld-elf.so.1
36185        0x8008b5000        0x8008c3000 r-x   14   16  22  10 CN-- vn /lib/libcrypt.so.5
36185        0x800ad5000        0x800b3c000 r-x   32   41   5   2 CN-- vn /usr/local/lib/libpcre.so.1.2.3
36185        0x800d3c000        0x800d4c000 r-x   16   17  13   6 CN-- vn /lib/libmd.so.6
36185        0x800f4c000        0x800f61000 r-x   21   22  17   8 CN-- vn /lib/libz.so.6
36185        0x801162000        0x8012d6000 r-x  294  322  60  28 CN-- vn /lib/libc.so.7
36185        0x80150b000        0x801524000 r-x   24   25  25  12 CN-- vn /lib/libthr.so.3
36185        0x802000000        0x802001000 r-x    1   12   2   1 CN-- vn objs/ngx_http_empty_gif_module.so
36185        0x802201000        0x802204000 r-x    3   17   2   1 CN-- vn objs/ngx_http_geoip_module.so
36185        0x802404000        0x80243f000 r-x   59   69   2   1 CN-- vn /usr/local/lib/libGeoIP.so.1.6.6
```


The working prototype was written in two hours. It could load the empty_gif and GeoIP modules dynamically.

Building Dynamic Modules

```
$ auto/configure --help | fgrep dynamic
  --with-http_xslt_module=dynamic    enable dynamic ngx_http_xslt_module
  --with-http_image_filter_module=dynamic
                                     enable dynamic ngx_http_image_filter_module
  --with-http_geoip_module=dynamic   enable dynamic ngx_http_geoip_module
  --with-http_split_clients_module=dynamic
                                     enable dynamic ngx_http_split_clients_module
  --with-http_empty_gif_module=dynamic
                                     enable dynamic ngx_http_empty_gif_module
  --with-mail=dynamic                enable dynamic POP3/IMAP4/SMTP proxy module
  --add-dynamic-module=PATH          enable external dynamic module
```


The syntax for building dynamic modules just extends the existing syntax. Basically, you tell that you want to build some modules dynamically, and the rest is handled by the «configure» scripts. They generate the necessary makefile rules to build nginx and requested modules.

Building Dynamic Modules

```
$ auto/configure --with-http_geoip_module=dynamic

$ make

$ ldd objs/nginx objs/ngx_http_geoip_module.so
objs/nginx:
        …
        libc.so.7 => /lib/libc.so.7 (0x801162000)
objs/ngx_http_geoip_module.so:
        …
        libGeoIP.so.1 => /usr/local/lib/libGeoIP.so.1 (0x801eb0000)
```


For example, if you build the dynamic GeoIP module, the dependency on the GeoIP library shifts from the primary nginx binary to the module. The module can then be packaged and distributed separately.

Building Dynamic Modules

```
$ auto/configure --with-http_geoip_module=dynamic \
                 --add-dynamic-module=nginx-rtmp-module-1.1.7
$ make

$ ls -l objs/*.so
-rwxr-xr-x+ 1 ru  staff   17136 Feb 30 18:06 objs/ngx_http_geoip_module.so
-rwxr-xr-x+ 1 ru  staff  318852 Feb 30 18:06 objs/ngx_rtmp_module.so
```


Building an external dynamic module is not much different. There’s only a slight change to the existing syntax. I’ll talk more about external modules a bit later.

Loading Modules

```
# nginx.conf

load_module modules/ngx_http_geoip_module.so;
load_module …;

http {
    geoip_country         GeoIP.dat;
    geoip_city            GeoLiteCity.dat;
    geoip_proxy           192.168.100.0/24;
    geoip_proxy           2001:0db8::/32;
    geoip_proxy_recursive on;
    …
```


For the configuration file, the new directive, load_module, is introduced. It tells NGINX to load a module at program's start-up. Once NGINX is running, you can change the list of modules by editing the configuration file and signaling NGINX to reload its configuration. NGINX will check the syntax validity and try to apply the new configuration. If this fails for some reason, it rolls back changes and continues to work with the old configuration. If you changed the set of modules, unused modules will be unloaded after applying a new configuration.

Gory Details

- modules loading order

So, let’s talk about implementation details.
Some modules have interdependencies. For example, if we take three so called «index» modules - random_index, index, and auto index - they should be loaded in this particular order. The same holds true for the chains of filter modules.

Gory Details

- modules loading order
- not everything can be made dynamic

Not all modules can be made dynamic. There are some modules that are an integral part of NGINX, which cannot be cut.

Gory Details

- modules loading order
- not everything can be made dynamic
- modules != modules

There are also some complex modules. For example, the mail is actually a set of several modules, but they are written in such a way that makes it impossible to cut specific modules. So now, mail is a single loadable module that provides 8 NGINX modules.

Gory Details

- modules loading order
- not everything can be made dynamic
- modules != modules
- no API

NGINX doesn’t have a cleanly separated programming API, or some abstraction layer that can be used by external modules. NGINX has a lot of conditional compilation defines that may influence the sizes and contents of various structures. This has to be taken into account when building a module. That is, to build a module NOT as part of nginx build, you have to use not only the same version of header files, but also the same set of compilation defines.

Gory Details

- modules loading order
- not everything can be made dynamic
- modules != modules
- no API
- subtleties

there are some corner cases, like what to do when a new configuration removes some filter modules, but we have to roll back to previous configuration due to issues. Internally, the filter modules are organized as chains, and we’d have to restore the filter chain to its previous state as well. This is one of the tasks that is still not solved.

Module Developers

```
# ngx_http_response-0.3/config

ngx_module_type=HTTP_ADDON
ngx_module_name=ngx_http_response_module
ngx_module_incs=
ngx_module_deps=
ngx_module_srcs="$ngx_addon_dir/ngx_http_response_module.c"
ngx_module_libs=

. auto/module

ngx_addon_name=$ngx_module_name
```


So, what it all means for module developers? If you’re not interested in dynamically loading your module, you don’t have to worry -- everything will continue to work as before. If you do however, you’ll be required to make some simple changes to your module’s configuration file. An example on the screen shows how such a config may look like. Basically, you specify the type and name of your module, supply the list of header files and an optional path to them, the list of source files, and libraries the module depends on, if any. Everything like before, but in a slightly different format. The auto/module script will handle the rest -- it will either generate makefile rules to build the module statically or dynamically, as you’ve requested. More complex modules may require more changes and multiple calls to auto/module script.

Who & When?

- Maxim Dounin <mdounin@mdounin.ru>
- Ruslan Ermilov <ru@nginx.com>
- End of Year

So, who’s in charge? There are two developers involved into the project, and you can see their names on the screen. We expect to finish and commit the whole work by the end of this year. It’s split into several stages internally, so we’ll probably publish some intermediate results earlier.

Thank you!

- Questions?
- Find me on the «Ask NGINX experts» booth at 2pm

Well, thank you!

If you’ve got some specific questions about dynamic modules, I welcome you to the «ask NGINX experts» booth starting at 2pm. Or you can ask your questions now.