Monitoring NGINX

BY K YOUNG & JOHN MATSON

CONTENTS

WHAT IS NGINX?

NGINX (pronounced "engine X") is a popular HTTP server and reverse proxy server. As an HTTP server, NGINX serves static content very efficiently and reliably, using relatively little memory. As a reverse proxy server, it can be used as a single, controlled point of access for multiple backend servers or for additional applications such as caching and load balancing. NGINX is available as a free, open-source product or in a more full-featured, commercially distributed version called NGINX Plus.

NGINX can also be used as a mail proxy and a generic TCP proxy, but this Refcard does not directly address NGINX monitoring for these use cases.

KEY NGINX METRICS

By monitoring NGINX you can catch two categories of issues: resource issues within NGINX itself, as well as problems developing elsewhere in your web infrastructure. Some of the metrics most NGINX users will benefit from monitoring include: requests per second, which provides a high-level view of combined end-user activity; server error rate, which indicates how often your servers are failing to process seemingly valid requests; and request processing time, which describes how long your servers are taking to process client requests (and which can point to slowdowns or other problems in your environment).

More generally, there are at least three key categories of metrics to watch:

- Basic activity metrics
- Error metrics
- Performance metrics

Below we'll break down a few of the most important NGINX metrics in each category, as well as metrics for a fairly common use case that deserves special mention: using NGINX Plus for reverse proxying. We will also describe how you can monitor all of these metrics with your graphing or monitoring tools of choice.

This Refcard references metric terminology introduced in our Monitoring 101 series, which provides a framework for metric collection and alerting.

BASIC ACTIVITY METRICS

Whatever your NGINX use case, you will no doubt want to monitor how many client requests your servers are receiving and how those requests are being processed.

NGINX Plus can report basic activity metrics exactly like open-source NGINX, but it also provides a secondary module that reports metrics slightly differently. This Refcard first discusses open-source NGINX, then the additional reporting capabilities provided by NGINX Plus.

NGINX

The diagram below shows the lifecycle of a client connection and how the open-source version of NGINX collects metrics during a connection.

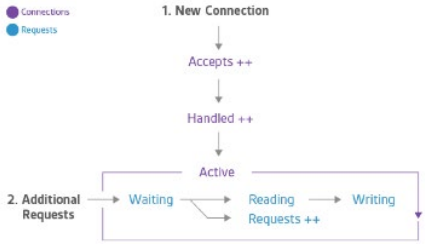

Figure 1: *Open-Source NGINX Client Connection Lifecycle*

Accepts, **handled**, and **requests** are ever-increasing counters. **Active**, **waiting**, **reading**, and **writing** grow and shrink with request volume.

NAME	DESCRIPTION	METRIC TYPE
accepts	Count of client connections attempted by NGINX	Resource: Utilization
handled	Count of successful client connections	Resource: Utilization
active	Currently active client connections	Resource: Utilization
dropped (calculated)	Count of dropped connections (accepts - handled)	Work: Errors*
requests	Count of client requests	Work: Throughput

*Strictly speaking, dropped connections is a metric of resource saturation, but since saturation causes NGINX to stop servicing some work (rather than queuing it up for later), "dropped" is best thought of as a work metric.

The accepts counter is incremented when an NGINX worker picks up a request for a connection from the OS, whereas handled is incremented when the worker actually gets a connection for the request (by establishing a new connection or reusing an open one). These two counts are usually the same—any divergence indicates that connections are being dropped, often because a resource limit, such as NGINX's worker_connections limit, has been reached.

Once NGINX successfully handles a connection, the connection moves to an active state, where it remains as client requests are processed:

ACTIVE STATE	
Waiting	An active connection may also be in a Waiting substate if there is no active request at the moment. New connections can bypass this state and move directly to Reading, most commonly when using "accept filter" or "deferred accept," in which case NGINX does not receive notice of work until it has enough data to begin working on the response. Connections will also be in the Waiting state after sending a response if the connection is set to keep-alive.
Reading	When a request is received, the connection moves out of the waiting state, and the request itself is counted as Reading. In this state NGINX is reading a client request header. Request headers are lightweight, so this is usually a fast operation.
Writing	After the request is read, it is counted as Writing, and remains in that state until a response is returned to the client. This means that the request is Writing while NGINX is waiting for results from upstream systems (systems "behind" NGINX), and while NGINX is operating on the response. Requests will often spend the majority of their time in the Writing state.

Often a connection will only support one request at a time. In this case, the number of Active connections == Waiting connections + Reading requests + Writing requests. However, the newer SPDY and HTTP/2 protocols allow multiple concurrent requests/ responses to be multiplexed over a connection, so Active may be less than the sum of Waiting, Reading, and Writing. (As of this writing, NGINX does not support HTTP/2, but expects to add support during 2015.

NGINX PLUS

As mentioned above, all of open-source NGINX's metrics are available within NGINX Plus, but Plus can also report additional metrics. This section covers the metrics that are only available from NGINX Plus.

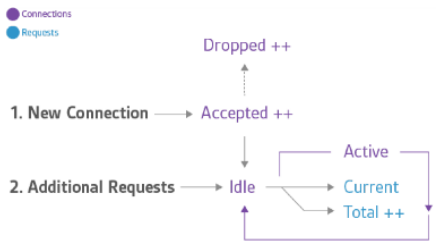

Figure 2: *NGINX Plus Client Connection Lifecycle*

Accepted, dropped, and total are ever-increasing counters. Active, idle, and current track the current number of connections or requests in each of those states, so they grow and shrink with request volume.

NAME	DESCRIPTION	METRIC TYPE
accepted	Count of client connections attempted by NGINX	Resource: Utilization
dropped	Count of dropped connections	Work: Errors*
active	Currently active client connections	Resource: Utilization
idle	Client connections with zero current requests	Resource: Utilization
total	Count of client requests	Work: Throughput

The accepted counter is incremented when an NGINX Plus worker picks up a request for a connection from the OS. If the worker fails to get a connection for the request (by establishing

a new connection or reusing an open one), then the connection is dropped, and the dropped counter is incremented. Ordinarily connections are dropped because a resource limit, such as NGINX Plus's worker_connectionslimit, has been reached.

Active and idle are the same as the active and waiting states in open-source NGINX as described above, with one key exception: in open-source NGINX, waiting falls under the active umbrella, whereas in NGINX Plus idle connections are excluded from the active count. Current is the same as the combined reading + writing states in open-source NGINX.

Total is a cumulative count of client requests. Note that a single client connection can involve multiple requests, so this number may be significantly larger than the cumulative number of connections. In fact, (total / accepted) yields the average number of requests per connection.

METRIC DIFFERENCES BETWEEN OPEN-SOURCE AND PLUS

NGINX (OPEN-SOURCE)	NGINX PLUS
accepts	accepted
dropped must be calculated	dropped is reported directly
reading + writing	current
waiting	idle
active (includes "waiting" states)	active (excludes "idle" states)
requests	total

METRIC TO ALERT ON: DROPPED CONNECTIONS

The number of connections that have been dropped is equal to the difference between accepts and handled (NGINX) or is exposed directly as a standard metric (NGINX Plus). Under normal

circumstances, dropped connections should be zero. If your rate of dropped connections per unit time starts to rise, look for possible resource saturation.

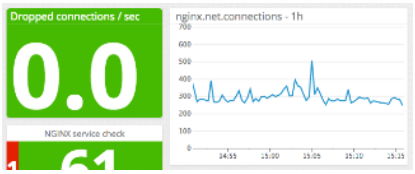

Figure 3: *NGINX Dropped Connections Per Second*

METRIC TO ALERT ON: REQUESTS PER SECOND

Sampling your request data (requests in open-source, or total in Plus) with a fixed time interval provides you with the number of requests you're receiving per unit of time—often minutes or seconds. Monitoring this metric can alert you to spikes in incoming web traffic, whether legitimate or nefarious, or sudden drops, which are usually indicative of problems. A drastic

change in requests per second can alert you to problems brewing somewhere in your environment, even if it cannot tell you exactly where those problems lie. Note that all requests are counted the same, regardless of their URLs.

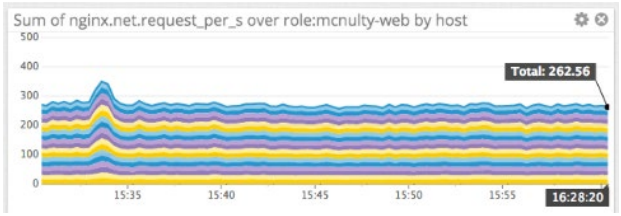

Figure 4: *NGINX Requests Per Second*

COLLECTING ACTIVITY METRICS

Open-source NGINX exposes these basic server metrics on a simple status page. Because the status information is displayed in a standardized form, virtually any graphing or monitoring tool can be configured to parse the relevant data for analysis, visualization, or alerting. NGINX Plus provides a JSON feed with much richer data. A later section of this Refcard will discuss how to enable NGINX metrics collection.

ERROR METRICS

NGINX error metrics tell you how often your servers are returning errors instead of producing useful work. Client errors are represented by 4xx status codes, server errors with 5xx status codes.

NAME	DESCRIPTION	METRIC TYPE	AVAILABILITY
4xx codes	Count of client errors	Work: Errors	NGINX logs, NGINX Plus
5xx codes	Count of server errors	Work: Errors	NGINX logs, NGINX Plus

METRIC TO ALERT ON: SERVER RATE ERROR

Your server error rate is equal to the number of 5xx errors divided by the total number of status codes (1xx, 2xx, 3xx, 4xx, 5xx), per unit of time (often one to five minutes). If your error rate starts to climb over time, you may need to investigate the source of the error. If it spikes suddenly, urgent action may be required, as clients are likely to report errors to the end user.

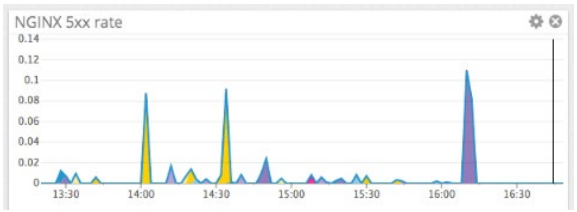

Figure 5: *NGINX Server Error Rate Metrics*

5xx (server error) codes are a valuable metric to monitor, particularly as a share of total response codes. NGINX Plus allows you to easily extract the number of 5xx codes per upstream server, as well as the total number of responses, to determine that particular server's error rate.

A note on client errors: while it is tempting to monitor 4xx, there is limited information you can derive from that metric since it measures client behavior without offering any insight into particular URLs. In other words, a change in 4xx could be noise, e.g. web scanners blindly looking for vulnerabilities.

COLLECTING ERROR METRICS

Although open-source NGINX does not make error rates immediately available for monitoring, there are at least two ways to capture that information:

- Use the expanded status module available with commercially-supported NGINX Plus.
- Configure NGINX's log module to write response codes in access logs.

PERFORMANCE METRICS

NAME	DESCRIPTION	METRIC TYPE	AVAILABILITY
request time	Time to process each request, in seconds	Work: Performance	NGINX logs

METRIC TO ALERT ON: REQUEST PROCESSING TIME

The request time metric logged by NGINX records the processing time for each request, from the reading of the first client bytes to fulfilling the request. Long response times can point to problems upstream.

COLLECTING PROCESSING TIME METRICS

NGINX and NGINX Plus users can capture data on processing time by adding the $request_time variable to the access log format.

REVERSE PROXY METRICS

One of the most common ways to use NGINX is as a reverse proxy. The commercially-supported NGINX Plus exposes a large number of metrics about backend (or "upstream") servers, which are relevant to a reverse proxy setup. This section highlights a few of the key upstream metrics that are available to users of NGINX Plus.

NAME	DESCRIPTION	METRIC TYPE	AVAILABILITY
Active connections by upstream server	Currently active client connections	Resource: Utilization	NGINX Plus
5xx codes by upstream server	Server errors	Work: Errors	NGINX Plus
Available servers per upstream group	Servers passing health checks	Resource: Availability	NGINX Plus

NGINX Plus segments its upstream metrics first by group, and then by individual server. So if, for example, your reverse proxy is distributing requests to five upstream web servers, you can see at a glance whether any of those individual servers is overburdened, and also whether you have enough healthy servers in the upstream group to ensure good response times.

ACTIVITY METRICS

The number of active connections per upstream server can help you verify that your reverse proxy is properly distributing work across your server group. If you are using NGINX as a load balancer, significant deviations in the number of connections handled by any one server can indicate that the server is struggling to process requests in a timely manner or that the load-balancing method (e.g. round-robin or IP hashing) you have configured is not optimal for your traffic patterns.

AVAILABILITY METRICS

For another view of the health of your web servers, NGINX also makes it simple to monitor the health of your upstream groups via the total number of servers currently available within each group. In a large reverse proxy setup, you may not care very much about the current state of any one server, just as long as your pool of available servers is capable of handling the load. But monitoring the total number of servers that are up within each upstream group can provide a very high-level view of the aggregate health of your web servers.

UPSTREAM METRICS

NGINX Plus upstream metrics are exposed on the internal NGINX Plus monitoring dashboard, and are also available via a JSON interface that can serve up metrics into virtually any external monitoring platform. See examples in the next section on collecting NGINX metrics.

HOW TO COLLECT NGINX METRICS

How you go about capturing metrics depends on which version of NGINX you are using, as well as which metrics you wish to access. Free, open-source NGINX and the commercial product NGINX Plus both have status modules that report metrics, and NGINX can also be configured to report certain metrics in its logs:

METRICS	AVAILABILITY		
	NGINX (OPEN-SOURCE)	NGINX PLUS	NGINX LOGS
accepts / accepted	✓	✓	
handled	✓	✓	
dropped	✓	✓	
active	✓	✓	
requests / total	✓	✓	
4xx codes		✓	✓
5xx codes		✓	✓
request time			✓

METRICS COLLECTION: NGINX (OPEN-SOURCE)

Open-source NGINX exposes several basic metrics about server activity on a simple status page, provided that you have the HTTP stub status module enabled. To check if the module is already enabled, run:

```
nginx -V 2>&1 | grep -o with-http_stub_status_module
```

The status module is enabled if you see with-http_stub_status_ module as output in the terminal.

If that command returns no output, you will need to enable the status module. You can use the --with-http_stub_status_ module configuration parameter when building NGINX from source:

```
configure \
...\
--with-http_stub_status_module
make
sudo make install
```

After verifying the module is enabled or enabling it yourself, you will also need to modify your NGINX configuration to set up a locally accessible URL (e.g. /nginx_status) for the status page:

```
ver {
location /atus {
stub_status on;
      access_log off;
      allow 127.0.0.1;
      deny all;
}
```

Note: The server blocks of the NGINX config are usually found not in the master configuration file (e.g., /etc/nginx/nginx.conf) but in

supplemental configuration files that are referenced by the master config. To find the relevant configuration files, first locate the master config by running:

```
nginx -t
```

Open the master configuration file listed, and look for lines beginning with "include" near the end of the http block, such as:

```
include /etc/nginx/conf.d/*.conf;
```

In one of the referenced config files you should find the main server block, which you can modify as above to configure NGINX metrics reporting. After changing any configurations, reload the configs by executing:

```
nginx -s reload
```

Now you can view the status page to see your metrics:

```
Active connections: 24
server  accepts handled requests
1156958 1156958 4491319
Reading: 0 Writing: 18 Waiting : 6
```

Note that if you are trying to access the status page from a remote machine, you will need to whitelist the remote machine's IP address in your status configuration, just as 127.0.0.1 is whitelisted in the configuration snippet above.

The NGINX status page is an easy way to get a quick snapshot of your metrics, but for continuous monitoring you will need to automatically record that data at regular intervals. Parsers for the NGINX status page already exist for monitoring tools such as Nagios and Datadog, as well as for the statistics collection daemoncollectD.

METRICS COLLECTION: NGINX PLUS

NGINX Plus provides many more metrics through its ngx_http_status_module than are available in open-source NGINX. Among the additional metrics exposed by NGINX Plus are bytes streamed, as well as information about upstream systems and caches. NGINX Plus also reports counts of all HTTP status code types (1xx, 2xx, 3xx, 4xx, 5xx). A sample NGINX Plus status board is available here.

Figure 6: *NGINX Plus Metrics Collection*

Note: the Active connections on the NGINX Plus status dashboard are defined slightly differently than the Active state connections in the metrics collected via the open-source NGINX stub status module. In NGINX Plus metrics, Active connections do not include connections in the Waiting state (aka Idle connections).

NGINX Plus also reports metrics in JSON format for easy integration with other monitoring systems. With NGINX Plus, you can see the metrics and health status for a given upstream grouping of servers, or drill down to get a count of just the response codes from a single server in that upstream:

```
nginx -V 2>&1 | grep -o with-http_stub_status_module
```

To enable the NGINX Plus metrics dashboard, you can add a status server block inside the http block of your NGINX configuration. (See the section above on collecting metrics from open-source NGINX for instructions on locating the relevant config files.) For example, to set up a status dashboard at your.ip. address:8080/status.html and a JSON interface at your.ip. address:8080/status, you would add the following server block:

```
server {
listen 8080;
root /usr/share/nginx/html;
location /status {
status;
    }
location = /status.html {
}
```

The status pages should be live once you reload your NGINX configuration:

```
nginx -s reload
```

The official NGINX Plus docs have more details on how to configure the expanded status module.

METRICS COLLECTION: NGINX LOGS

NGINX's log module writes configurable access logs to a destination of your choosing. You can customize the format of your logs and the data they contain by adding or subtracting variables. The simplest way to capture detailed logs is to add the following line in the server block of your config file (see the section on collecting metrics from open-source NGINX for instructions on locating your config files):

```
access_log logs/host.access.log combined;
```

After changing any NGINX configurations, reload the configs by executing:

```
nginx -s reload
```

The "combined" log format, included by default, captures a number of key data points, such as the actual HTTP request and the corresponding response code. In the example logs below, NGINX logged a 200 (success) status code for a request for /index. html and a 404 (not found) error for the nonexistent /fail.

```
127.0.0.1 - - [19/Feb/2015:12:10:46 -0500] "GET /index.
html HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel
Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/40.0.2214.111 Safari 537.36"
7.0.0.1 - - [19/Feb/2015:12:11:05 -0500] "GET /fail HTTP/1.1"
404 570 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.111
Safari/537.36"
```

You can log request processing time as well by adding a new log format to thehttp block of your NGINX config file:

```
log_format nginx '$remote_addr - $remote_user [$time_ local] '
 '"$request" $status $body_bytes_sent $request_time '
 '"$http_referer" "$http_user_agent"';
```

And by adding or modifying the access_log line in the server block of your config file:

```
access_log logs/host.access.log nginx;
```

After reloading the updated configs (by running nginx -s reload), your access logs will include response times, as seen below. The units are seconds, with millisecond resolution. In this instance, the server received a request for /big.pdf, returning a 206 (success) status code after sending 33973115 bytes. Processing the request took 0.202 seconds (202 milliseconds):

```
127.0.0.1 - - [19/Feb/2015:15:50:36 -0500] "GET /big.pdf
HTTP/1.1" 206 33973115 0.202 "-" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/40.0.2214.111 Safari/537.36"
```

You can use a variety of tools and services to parse and analyze NGINX logs. For instance, rsyslog can monitor your logs and pass them to any number of log-analytics services; you can use a free, open-source tool such as logstash to collect and analyze logs; or you can use a unified logging layer such as Fluentd to collect and parse your NGINX logs.

HOW TO COLLECT NGINX METRICS

Which NGINX metrics you monitor will depend on the tools available to you, and whether the insight provided by a given metric justifies the overhead of monitoring that metric. For instance, measuring error rates may be important enough to your organization to justify investing in NGINX Plus or implementing a system to capture and analyze logs. Eventually you will recognize additional, more specialized metrics that are particularly relevant to your own infrastructure and use cases. Of course, what you monitor will depend on the tools you have and the metrics available to you.

HOW TO COLLECT NGINX METRICS

NGINX: nginx.org

NGINX Documentation: nginx.org/en/docs

NGINX Plus: nginx.com

NGINX Plus Live Activity: nginx.com/products/live-activity-monitoring

How to Monitor NGINX with Datadog: datadoghq.com/blog/how-to-monitor-nginxwith-datadog

ABOUT THE AUTHORS

K YOUNG is Director of Strategic Initiatives at Datadog. He has more than a decade of experience leading teams developing software to run at scale.

JOHN MATSON is the Content Developer at Datadog, where he writes about monitoring, metrics, and making sense of complex infrastructures. In a past life, John was a staff writer and editor at *Scientific American* magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com