

## Prometheus metric library for Nginx

This is a Lua library that can be used with Nginx to keep track of metrics and expose them on a separate web page to be pulled by Prometheus.

### Installation

To use this library, you will need the `ngx_lua` nginx module. You can either use a lua-enabled nginx-based server like OpenResty, or a regular nginx server with the module enabled: for example, on Debian 10 you can simply install `libnginx-mod-http-lua` (but please read the known issues if you use a later Debian version).

The library file - `prometheus.lua` - needs to be available in `LUA_PATH`. If this is the only Lua library you use, you can just point `lua_package_path` to the directory with this git repo checked out (see example below).

OpenResty users will find this library in opm. It is also available via luarocks.

### Quick start guide

To track request latency broken down by server name and request count broken down by server name and status, add the following to the `http` section of `nginx.conf`:

```
1 lua_shared_dict prometheus_metrics 10M;
2 lua_package_path "/path/to/nginx-lua-prometheus/?.lua;;";
3
4 init_worker_by_lua_block {
5     prometheus = require("prometheus").init("prometheus_metrics")
6
7     metric_requests = prometheus:counter(
8         "nginx_http_requests_total", "Number of HTTP requests", {"host", "
9         status"})
10    metric_latency = prometheus:histogram(
11        "nginx_http_request_duration_seconds", "HTTP request latency", {"
12        host"})
13    metric_connections = prometheus:gauge(
14        "nginx_http_connections", "Number of HTTP connections", {"state"})
15 }
```

---

```
16 metric_requests:inc(1, {ngx.var.server_name, ngx.var.status})
17 metric_latency:observe(tonumber(ngx.var.request_time), {ngx.var.
    server_name})
18 }
```

This: \* configures a shared dictionary for your metrics called `prometheus_metrics` with a 10MB size limit; \* registers a counter called `nginx_http_requests_total` with two labels: `host` and `status`; \* registers a histogram called `nginx_http_request_duration_seconds` with one label `host`; \* registers a gauge called `nginx_http_connections` with one label `state`; \* on each HTTP request measures its latency, recording it in the histogram and increments the counter, setting current server name as the `host` label and HTTP status code as the `status` label.

Last step is to configure a separate server that will expose the metrics. Please make sure to only make it reachable from your Prometheus server:

```
1 server {
2     listen 9145;
3     allow 192.168.0.0/16;
4     deny all;
5     location /metrics {
6         content_by_lua_block {
7             metric_connections:set(ngx.var.connections_reading, {"reading"})
8             metric_connections:set(ngx.var.connections_waiting, {"waiting"})
9             metric_connections:set(ngx.var.connections_writing, {"writing"})
10            prometheus:collect()
11        }
12    }
13 }
```

Metrics will be available at `http://your.nginx:9145/metrics`. Note that the gauge metric in this example contains values obtained from nginx global state, so they get set immediately before metrics are returned to the client.

## API reference

### `init()`

**syntax:** `require("prometheus").init(dict_name, [options])`

Initializes the module. This should be called once from the `init_worker_by_lua_block` section of nginx configuration.

- `dict_name` is the name of the nginx shared dictionary which will be used to store all metrics. Defaults to `prometheus_metrics` if not specified.
- `options` is a table of configuration options that can be provided. Accepted options are:

- 
- `prefix` (string): metric name prefix. This string will be prepended to metric names on output.
  - `error_metric_name` (string): Can be used to change the default name of error metric (see Built-in metrics for details).
  - `sync_interval` (number): sets the sync interval for per-worker counters and key index (in seconds). This sets the boundary on eventual consistency of counter metric increments, and metric resets/deletions. Defaults to 1.

Returns a `prometheus` object that should be used to register metrics.

Example:

```
1 init_worker_by_lua_block {
2     prometheus = require("prometheus").init("prometheus_metrics", {
3         sync_interval=3})
4 }
```

### **prometheus:counter()**

**syntax:** `prometheus:counter(name, description, label_names)`

Registers a counter. Should be called once for each counter from the `init_worker_by_lua_block` section.

- `name` is the name of the metric.
- `description` is the text description that will be presented to Prometheus along with the metric. Optional (pass `nil` if you still need to define label names).
- `label_names` is an array of label names for the metric. Optional.

Naming section of Prometheus documentation provides good guidelines on choosing metric and label names.

Returns a `counter` object that can later be incremented.

Example:

```
1 init_worker_by_lua_block {
2     prometheus = require("prometheus").init("prometheus_metrics")
3
4     metric_bytes = prometheus:counter(
5         "nginx_http_request_size_bytes", "Total size of incoming requests")
6     metric_requests = prometheus:counter(
7         "nginx_http_requests_total", "Number of HTTP requests", {"host", "
8         status"})
9 }
```

---

## **prometheus:gauge()**

**syntax:** `prometheus:gauge(name, description, label_names)`

Registers a gauge. Should be called once for each gauge from the `init_worker_by_lua_block` section.

- `name` is the name of the metric.
- `description` is the text description that will be presented to Prometheus along with the metric. Optional (pass `nil` if you still need to define label names).
- `label_names` is an array of label names for the metric. Optional.

Returns a `gauge` object that can later be set.

Example:

```
1 init_worker_by_lua_block {
2     prometheus = require("prometheus").init("prometheus_metrics")
3
4     metric_connections = prometheus:gauge(
5         "nginx_http_connections", "Number of HTTP connections", {"state"})
6 }
```

## **prometheus:histogram()**

**syntax:** `prometheus:histogram(name, description, label_names, buckets)`

Registers a histogram. Should be called once for each histogram from the `init_worker_by_lua_block` section.

- `name` is the name of the metric.
- `description` is the text description. Optional.
- `label_names` is an array of label names for the metric. Optional.
- `buckets` is an array of numbers defining bucket boundaries. Optional, defaults to 20 latency buckets covering a range from 5ms to 10s (in seconds).

Returns a `histogram` object that can later be used to record samples.

Example:

```
1 init_worker_by_lua_block {
2     prometheus = require("prometheus").init("prometheus_metrics")
3
4     metric_latency = prometheus:histogram(
5         "nginx_http_request_duration_seconds", "HTTP request latency", {"host"})
6 }
```

---

```
6 metric_response_sizes = prometheus:histogram(  
7     "nginx_http_response_size_bytes", "Size of HTTP responses", nil,  
8     {10,100,1000,10000,100000,1000000})  
9 }
```

### **prometheus:collect()**

**syntax:** prometheus:collect()

Presents all metrics in a text format compatible with Prometheus. This should be called in `content_by_lua_block` to expose the metrics on a separate HTTP page.

Example:

```
1 location /metrics {  
2     content_by_lua_block { prometheus:collect() }  
3     allow 192.168.0.0/16;  
4     deny all;  
5 }
```

### **prometheus:metric\_data()**

**syntax:** prometheus:metric\_data()

Returns metric data as an array of strings.

### **counter:inc()**

**syntax:** counter:inc(*value*, *label\_values*)

Increments a previously registered counter. This is usually called from `log_by_lua_block` globally or per server/location.

- *value* is a value that should be added to the counter. Defaults to 1.
- *label\_values* is an array of label values.

The number of label values should match the number of label names defined when the counter was registered using `prometheus:counter()`. No label values should be provided for counters with no labels. Non-printable characters will be stripped from label values.

Example:

```
1 log_by_lua_block {  
2     metric_bytes:inc(tonumber(ngx.var.request_length))  
}
```

---

```
3 metric_requests:inc(1, {ngx.var.server_name, ngx.var.status})
4 }
```

### **counter:del()**

**syntax:** counter:del(*label\_values*)

Delete a previously registered counter. This is usually called when you don't need to observe such counter (or a metric with specific label values in this counter) any more. If this counter has labels, you have to pass *label\_values* to delete the specific metric of this counter. If you want to delete all the metrics of a counter with labels, you should call `Counter:reset()`.

- *label\_values* is an array of label values.

The number of label values should match the number of label names defined when the counter was registered using `prometheus:counter()`. No label values should be provided for counters with no labels. Non-printable characters will be stripped from label values.

This function will wait for `sync_interval` before deleting the metric to allow all workers to sync their counters.

### **counter:reset()**

**syntax:** counter:reset()

Delete all metrics for a previously registered counter. If this counter have no labels, it is just the same as `Counter:del()` function. If this counter have labels, it will delete all the metrics with different label values.

This function will wait for `sync_interval` before deleting the metrics to allow all workers to sync their counters.

### **gauge:set()**

**syntax:** gauge:set(*value*, *label\_values*)

Sets the current value of a previously registered gauge. This could be called from `log_by_lua_block` globally or per server/location to modify a gauge on each request, or from `content_by_lua_block` just before `prometheus::collect()` to return a real-time value.

- *value* is a value that the gauge should be set to. Required.
- *label\_values* is an array of label values.

---

## **gauge:inc()**

**syntax:** `gauge:inc(value, label_values)`

Increments or decrements a previously registered gauge. This is usually called when you want to observe the real-time value of a metric that can both be increased and decreased.

- `value` is a value that should be added to the gauge. It could be a negative value when you need to decrease the value of the gauge. Defaults to 1.
- `label_values` is an array of label values.

The number of label values should match the number of label names defined when the gauge was registered using `prometheus:gauge()`. No label values should be provided for gauges with no labels. Non-printable characters will be stripped from label values.

## **gauge:del()**

**syntax:** `gauge:del(label_values)`

Delete a previously registered gauge. This is usually called when you don't need to observe such gauge (or a metric with specific label values in this gauge) any more. If this gauge has labels, you have to pass `label_values` to delete the specific metric of this gauge. If you want to delete all the metrics of a gauge with labels, you should call `Gauge:reset()`.

- `label_values` is an array of label values.

The number of label values should match the number of label names defined when the gauge was registered using `prometheus:gauge()`. No label values should be provided for gauges with no labels. Non-printable characters will be stripped from label values.

## **gauge:reset()**

**syntax:** `gauge:reset()`

Delete all metrics for a previously registered gauge. If this gauge have no labels, it is just the same as `Gauge:del()` function. If this gauge have labels, it will delete all the metrics with different label values.

## **histogram:observe()**

**syntax:** `histogram:observe(value, label_values)`

---

Records a value in a previously registered histogram. Usually called from `log_by_lua_block` globally or per server/location.

- `value` is a value that should be recorded. Required.
- `label_values` is an array of label values.

Example:

```
1 log_by_lua_block {
2     metric_latency:observe(tonumber(ngx.var.request_time), {ngx.var.
      server_name})
3     metric_response_sizes:observe(tonumber(ngx.var.bytes_sent))
4 }
```

## histogram:reset()

**syntax:** histogram:reset()

Delete all metrics for a previously registered histogram.

This function will wait for `sync_interval` before deleting the metrics to allow all workers to sync their counters.

## Built-in metrics

The module increments an error metric called `nginx_metric_errors_total` (unless another name was configured in `init()`) if it encounters an error (for example, when `lua_shared_dict` becomes full). You might want to configure an alert on that metric.

## Caveats

### Usage in stream module

For now, there is no way to share a dictionary between HTTP and Stream modules in Nginx. If you are using this library to collect metrics from stream module, you will need to configure a separate endpoint to return them. Here's an example.

```
1 server {
2     listen 9145;
3     content_by_lua_block {
4         local sock = assert(ngx.req.socket(true))
5         local data = sock:receive()
6         local location = "GET /metrics"
```



---

```
7     if string.sub(data, 1, string.len(location)) == location then
8         ngx.say("HTTP/1.1 200 OK")
9         ngx.say("Content-Type: text/plain")
10        ngx.say("")
11        ngx.say(table.concat(prometheus:metric_data(), ""))
12    else
13        ngx.say("HTTP/1.1 404 Not Found")
14    end
15 }
16 }
```

## Known issues

### libnginx-mod-http-lua broken in some Debian and Ubuntu versions

Note that recent stable versions of Debian and Ubuntu are known to package `ngx_lua` version incompatible with the version of `nginx` shipped in the same distro. This results in nginx process segfaulting when the lua module is used, making it completely unusable. In such case nginx error logs will clearly indicate that the process crashed, e.g.:

```
1 [alert] 123#123: worker process 45678 exited on signal 11
```

The following versions of Debian and Ubuntu have been known to have this issue:

- Debian 11 (bug #994178)
- Ubuntu 20.04 and 21.04 (bug #1893753)

## Troubleshooting

### Make sure that nginx lua module is enabled

If you experience problems indicating that nginx doesn't know how to interpret lua scripts, please make sure that the lua module is enabled. You might need something like this in your `nginx.conf`:

```
1 load_module modules/ndk_http_module.so;
2 load_module modules/nginx_http_lua_module.so;
```

### Keep lua code cache enabled

This module expects the `lua_code_cache` option to be `on` (which is the default).

---

## Try using an older version of the library

If you are seeing library initialization errors, followed by errors for each metric change request (e.g. *attempt to index global '...' (a nil value)*), you are probably using an old version of lua-nginx-module. For example, this will happen if you try using the latest version of this library with the `nginx-extras` package shipped with Ubuntu 16.04.

If you cannot upgrade nginx and lua-nginx-module, you can try using an older version of this library; it will not have the latest performance optimizations, but will still be functional. The recommended older release to use is 0.20181120.

## Development

### Install dependencies for testing

- `luarocks install luacheck`
- `luarocks install luaunit`

### Run tests

- `luacheck --globals ngx -- prometheus.lua`
- `lua prometheus_test.lua`
- `cd integration && ./test.sh` (requires Docker and Go)

### Releasing new version

- update CHANGELOG.md
- update version in the `dist.ini`
- rename `.rockspec` file and update version inside it
- commit changes
- create a new Git tag: `git tag 0.XXXXXXXX && git push origin 0.XXXXXXXX`
- push to luarocks: `luarocks upload nginx-lua-prometheus-0.20181120-1.rockspec`
- upload to OPM: `opm build && opm upload`

## Credits

- Created and maintained by Anton Tolchanov (@knyar)

- 
- Metrix prefix support contributed by david birdsong (@davidbirdsong)
  - Gauge support contributed by Cosmo Petrich (@cosmopetrich)
  - Performance improvements and per-worker counters are contributed by Wangchong Zhou (@ff-fonion) / @Kong.
  - Metric name tracking improvements contributed by Jan Dolinár (@dolik-rce)

## **License**

Licensed under MIT license.

## **Third Party License**

Following third party modules are used in this library:

- Kong/lua-resty-counter

This module is licensed under the Apache 2.0 license.

Copyright (C) 2019, Kong Inc.

All rights reserved.