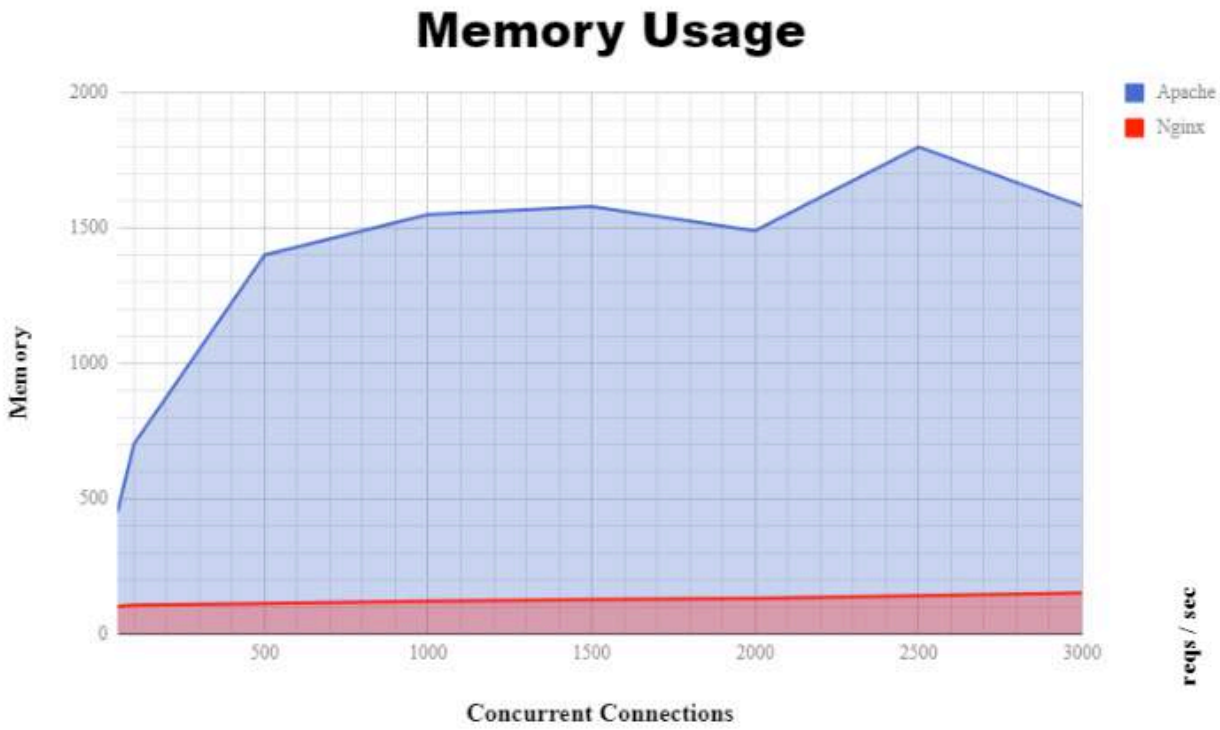
NGINX从入门到精通进阶系列培训

应用篇： 榨干单机NGINX性能的诀窍

廖健雄 j.liao@f5.com

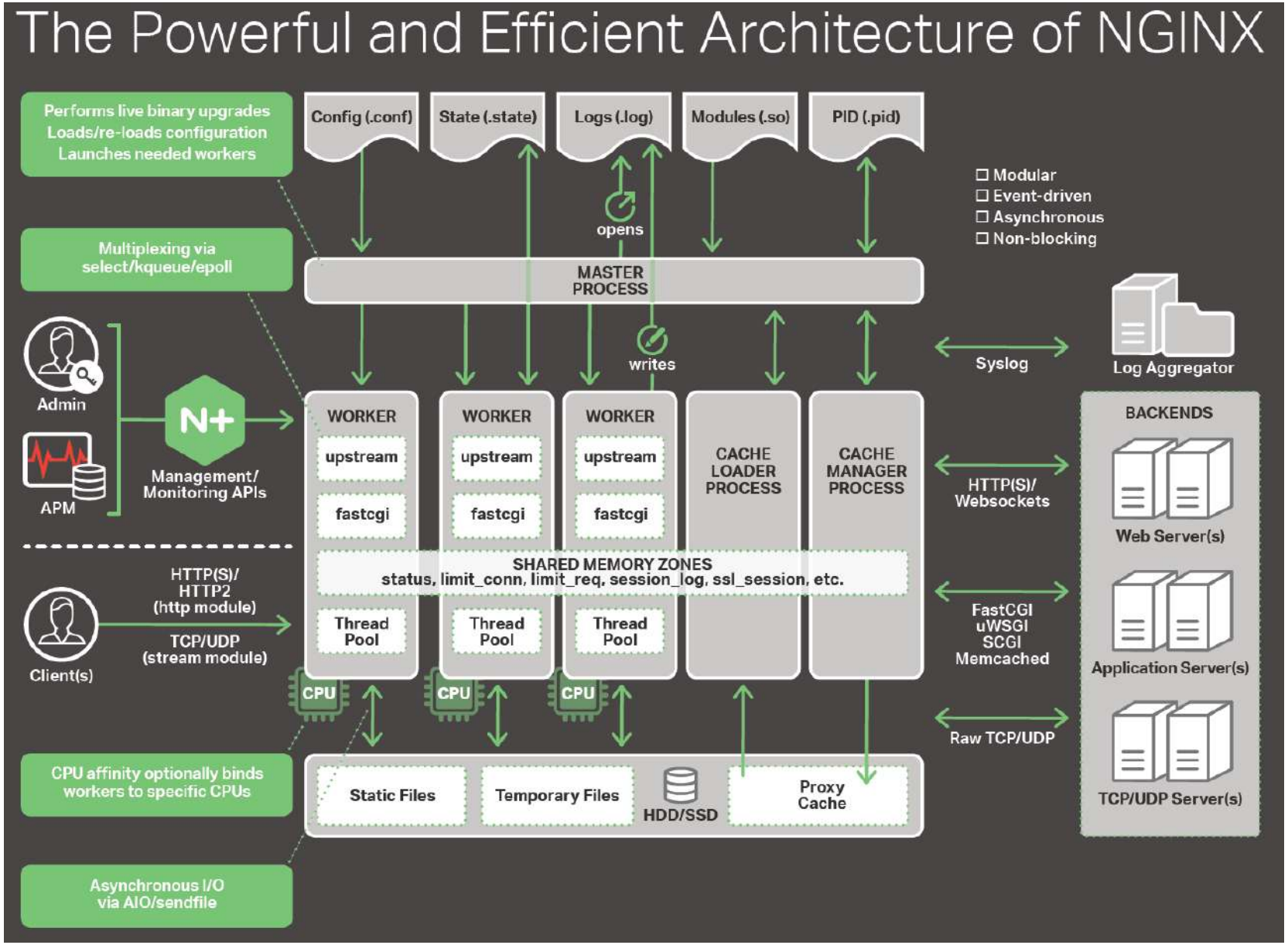
NGINX-国际象棋大师

NGINX VS Apache



https://help.dreamhost.com/hc/en-us/articles/215945987-Web-server-performance-comparison

NGINX架构图


The Powerful and Efficient Architecture of NGINX
Performs live binary upgrades
Loads/re-loads configuration
Launches needed workers
Config (.conf)
State (.state)
Logs (.log)
Modules (.so)
PID (.pid)
opens
Modular
Event-driven
Asynchronous
Non-blocking
Multiplexing via
select/kqueue/epoll
MASTER
PROCESS
writes
Syslog
Log Aggregator
Admin
APM
N+
Management/
Monitoring APIs
WORKER
WORKER
WORKER
upstream
upstream
upstream
fastcgi
fastcgi
fastcgi
CACHE
LOADER
PROCESS
CACHE
MANAGER
PROCESS
BACKENDS
HTTP(S)/
Websockets
Web Server(s)
SHARED MEMORY ZONES
status, limit_conn, limit_req, session_log, ssl_session, etc.
Thread
Pool
Thread
Pool
Thread
Pool
HTTP(S)/
HTTP2
(http module)
TCP/UDP
(stream module)
Client(s)
FastCGI
uWSGI
SCGI
Memcached
Application Server(s)
CPU
CPU
CPU
Raw TCP/UDP
TCP/UDP Server(s)
CPU affinity optionally binds
workers to specific CPUs
Static Files
Temporary Files
HDD/SSD
Proxy
Cache
Asynchronous I/O
via AIO/sendfile

异步非阻塞模型

传统服务器：

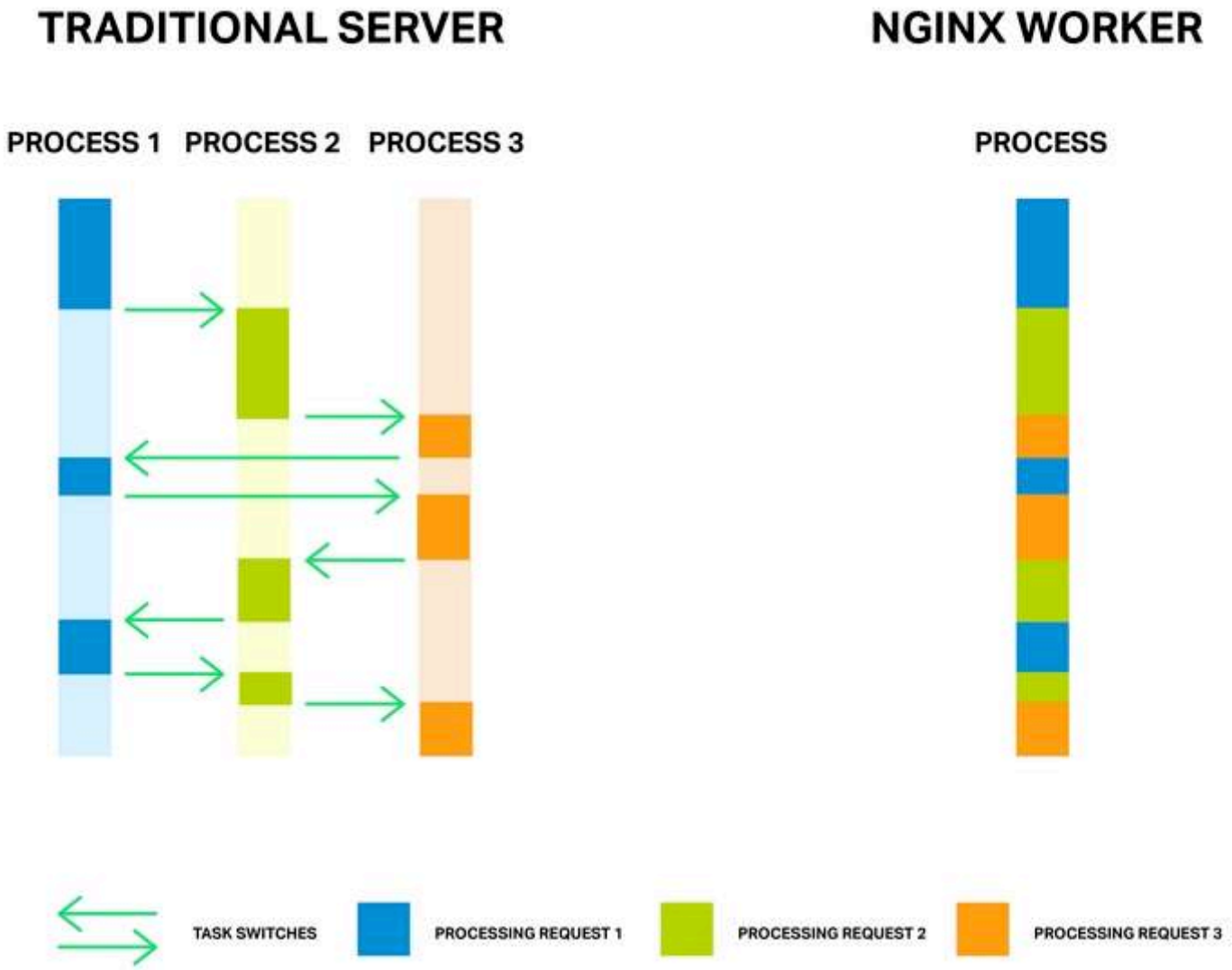
一个进程/线程处理一个连接/请求

阻塞模型、依赖OS调度实现并发

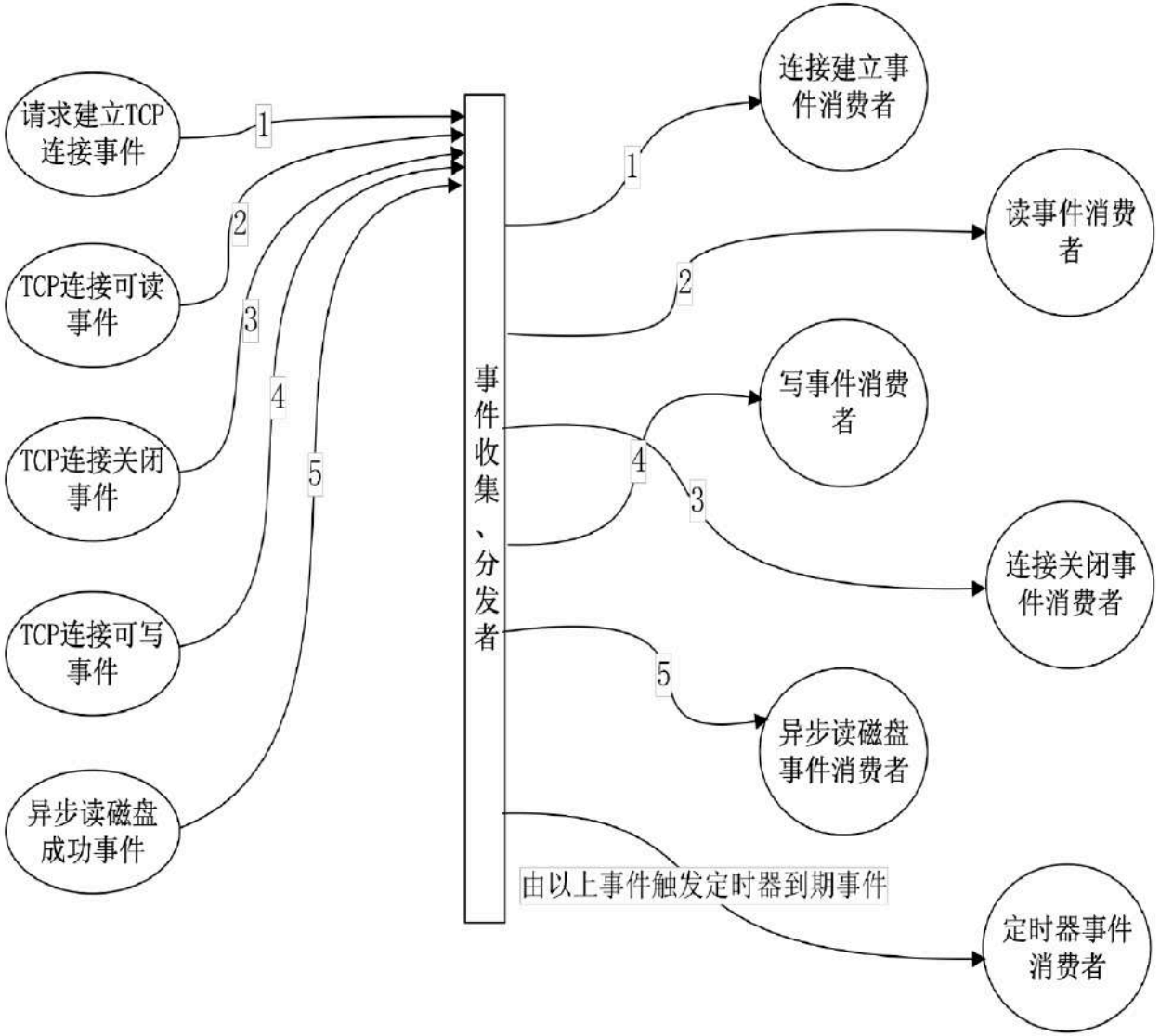
NGINX：

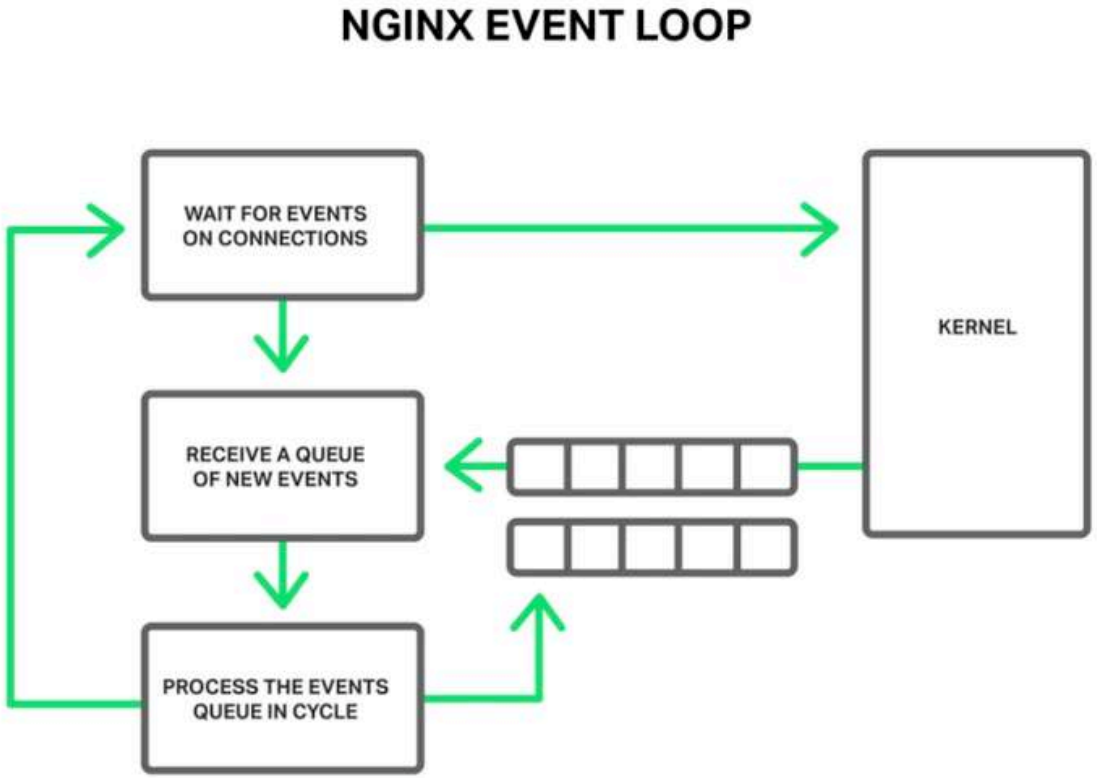
一个进程处理多个连接/请求

非阻塞模型、减少OS进程切换



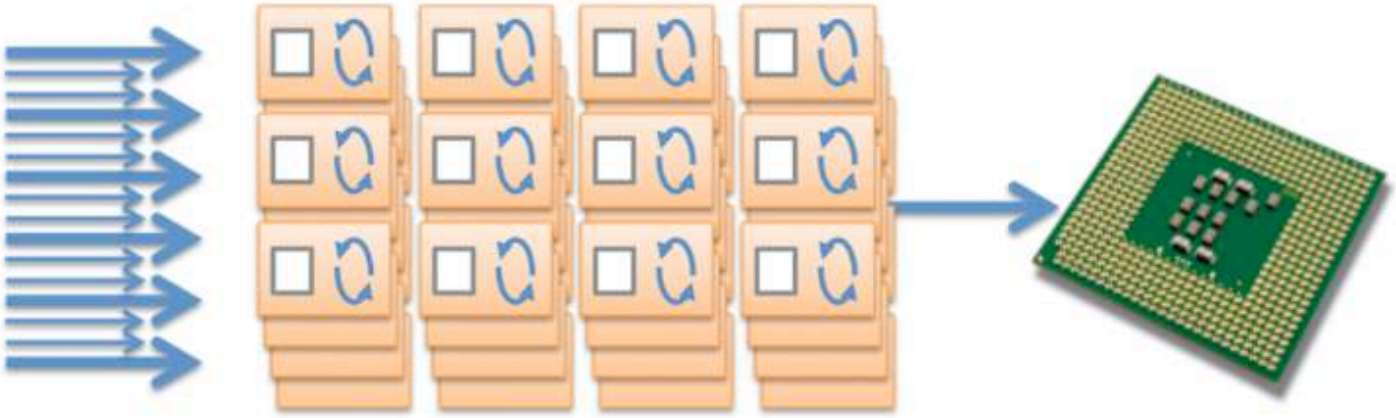
事件驱动模型


请求建立TCP连接事件
TCP连接可读事件
TCP连接关闭事件
TCP连接可写事件
异步读磁盘成功事件
事件收集、分发者
连接建立事件消费者
读事件消费者
写事件消费者
连接关闭事件消费者
异步读磁盘事件消费者
由以上事件触发定时器到期事件
定时器事件消费者
1
2
3
4
5


NGINX EVENT LOOP
WAIT FOR EVENTS ON CONNECTIONS
RECEIVE A QUEUE OF NEW EVENTS
PROCESS THE EVENTS QUEUE IN CYCLE
KERNEL


BLOCKING OPERATION
WAITING

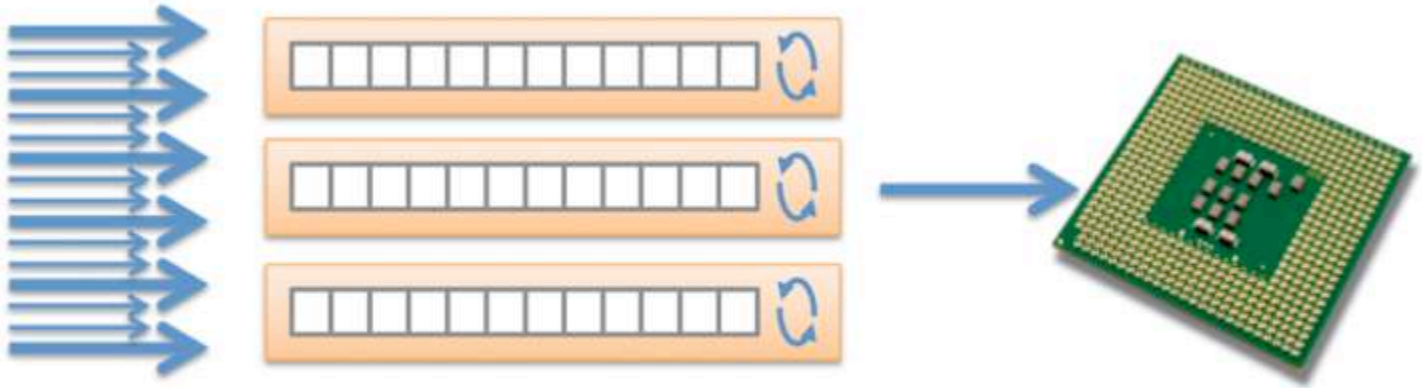
事件驱动模型



Hundreds of concurrent connections...

require hundreds of heavyweight threads or processes...

competing for limited CPU and memory

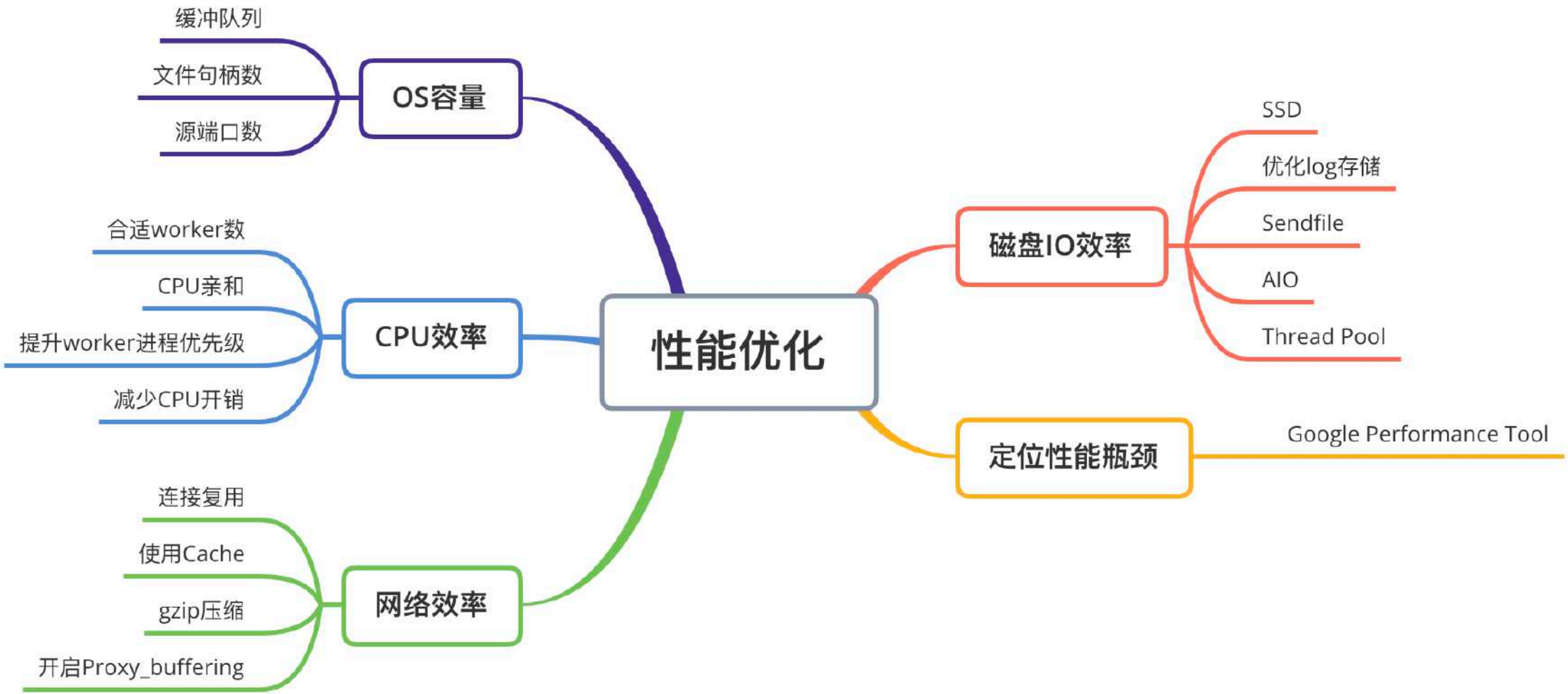


Hundreds of concurrent connections...

handled by a small number of multiplexing processes...

typically one process per core

性能优化方法论


性能优化
OS容量
缓冲队列
文件句柄数
源端口数
CPU效率
合适worker数
CPU亲和
提升worker进程优先级
减少CPU开销
网络效率
连接复用
使用Cache
gzip压缩
开启Proxy_buffering
磁盘IO效率
SSD
优化log存储
Sendfile
AIO
Thread Pool
定位性能瓶颈
Google Performance Tool

诀窍一：别让OS限制了NGINX的性能

缓冲队列

max_syn_backlog，默认128

net.core.somaxconn ，默认128

net.core.netdev_max_backlog，默认1000

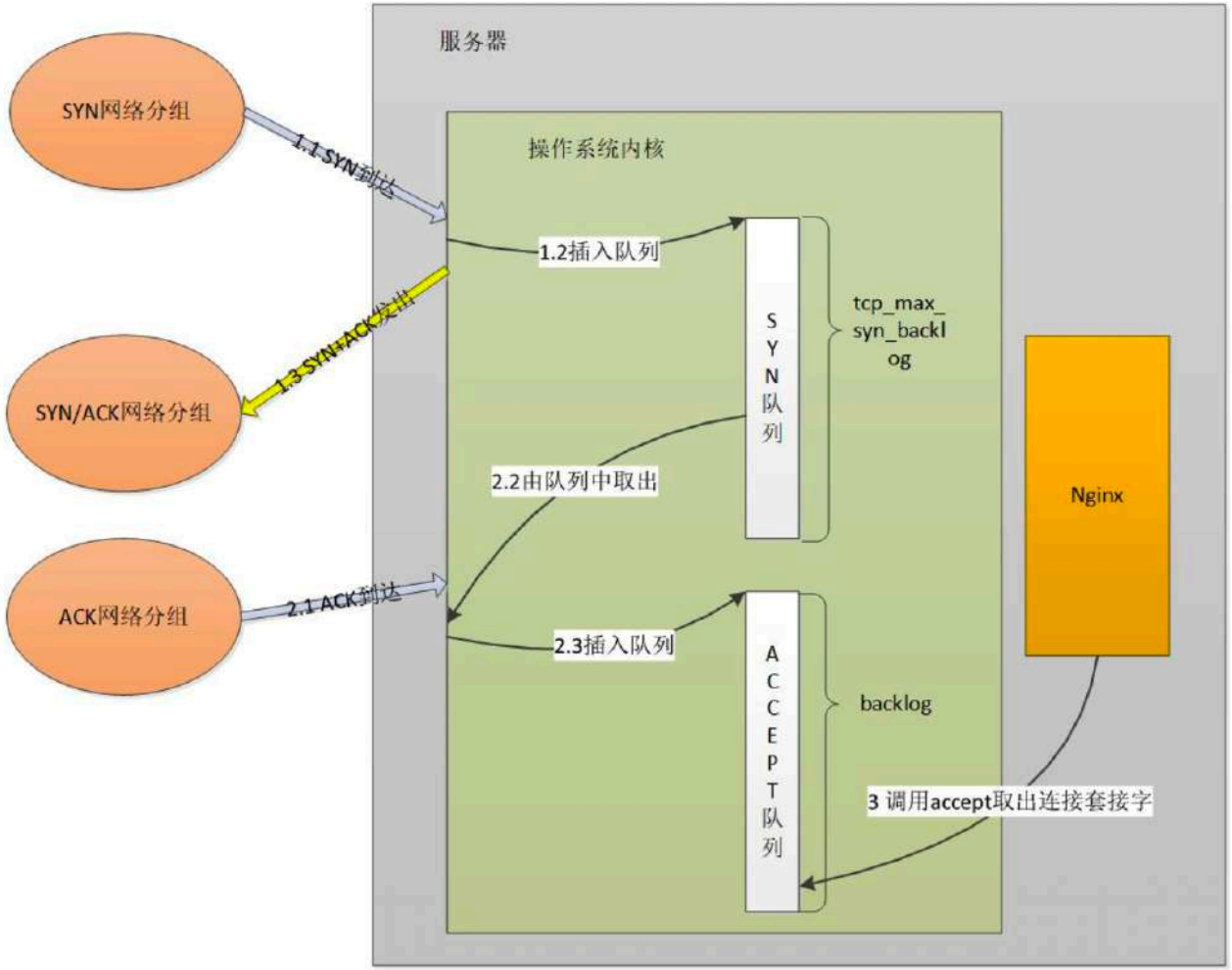
文件句柄数

sys.fs.file-max, 系统全局文件句柄数，默认379012

nofile，用户的全局文件句柄数，默认1024

可用端口数

net.ipv4.ip_local_port_range 默认32768 65535



诀窍二：提升CPU使用效率

合适的worker进程数

worker进程数 = CPU 核数

CPU亲和

每个worker进程绑定一个CPU核，提升缓存命中率

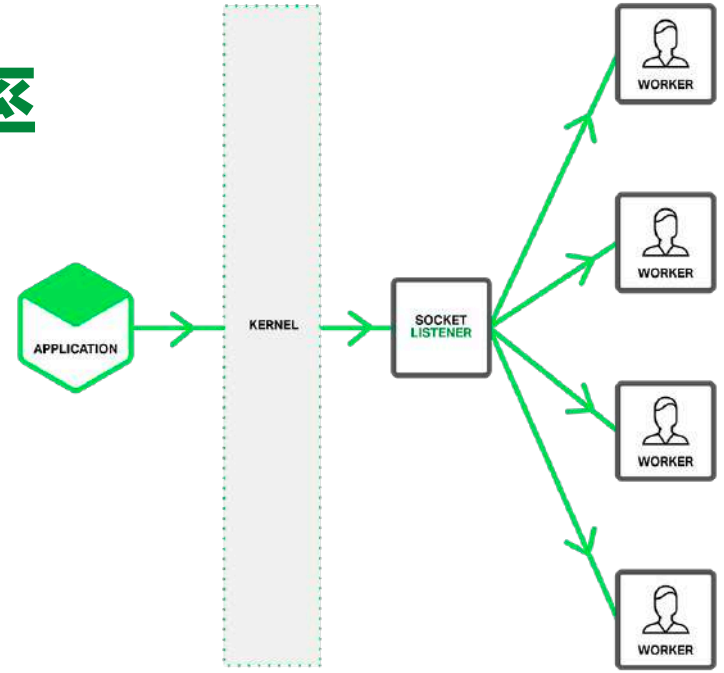
增加worker进程的CPU使用时间

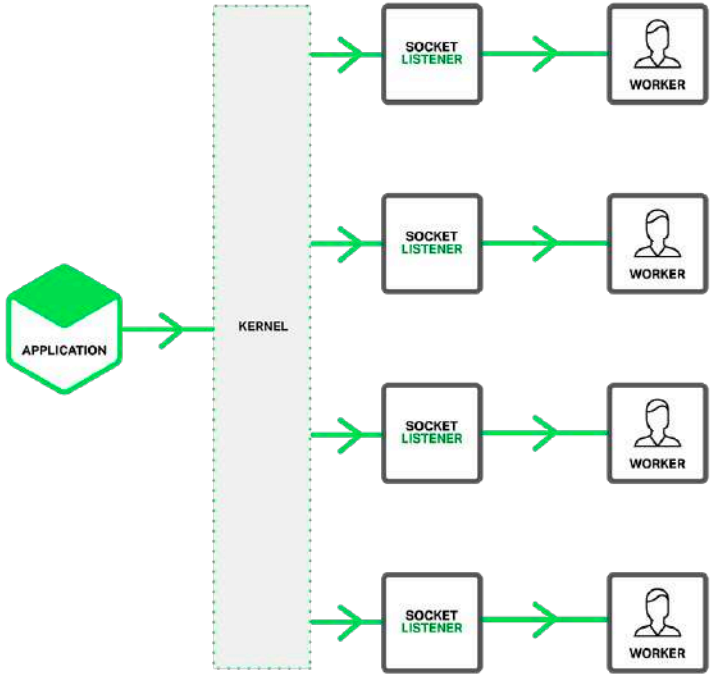
提升worker进程的优先级

减少CPU开销

multi_accept，会导致worker间负载不均
accept_mutex解决惊群问题

reuseport，弊端是reload会RST





诀窍三：提升网络效率

连接复用

减少upstream建连

使用Cache

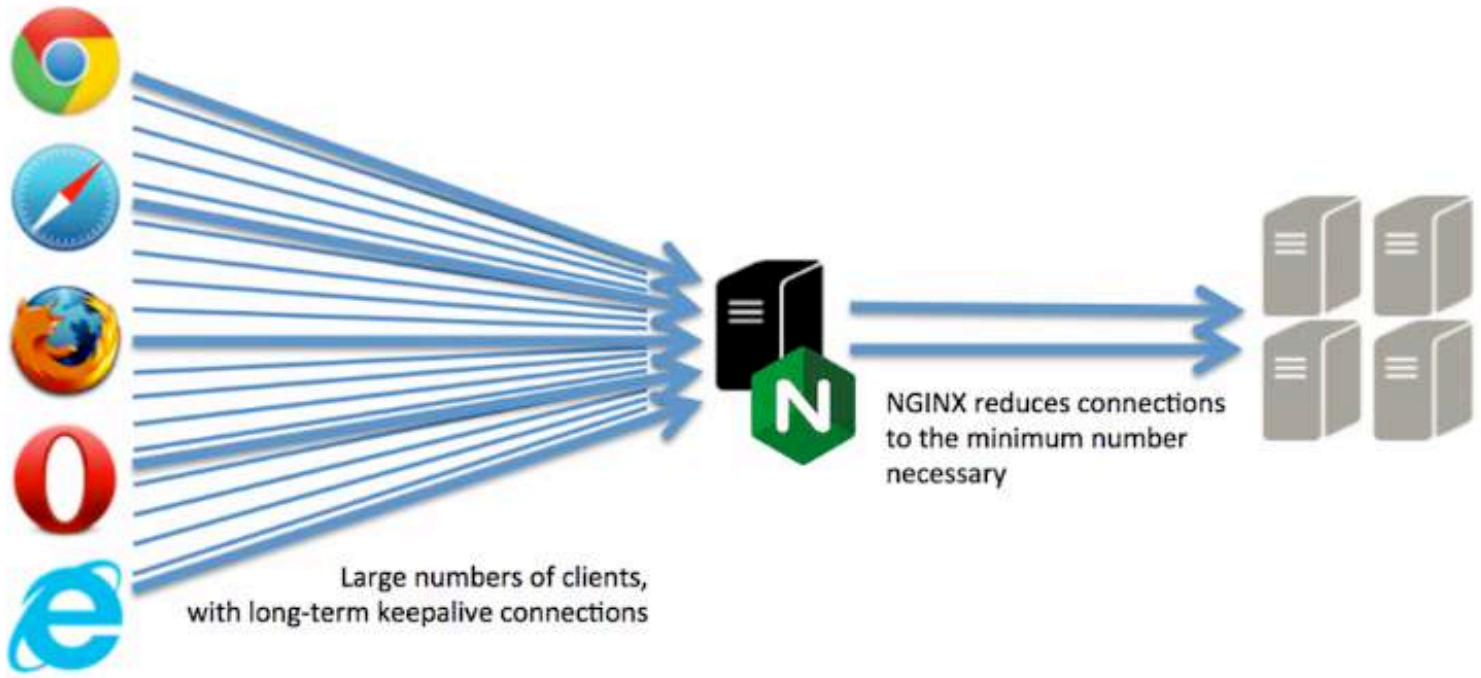
超时时间对业务的影响

gzip压缩

会增加cpu开销，需平衡

开启proxy_buffering

谨慎设置proxy_buffer 大小，避免磁盘io读写



诀窍四：提升磁盘IO效率

更换SSD！

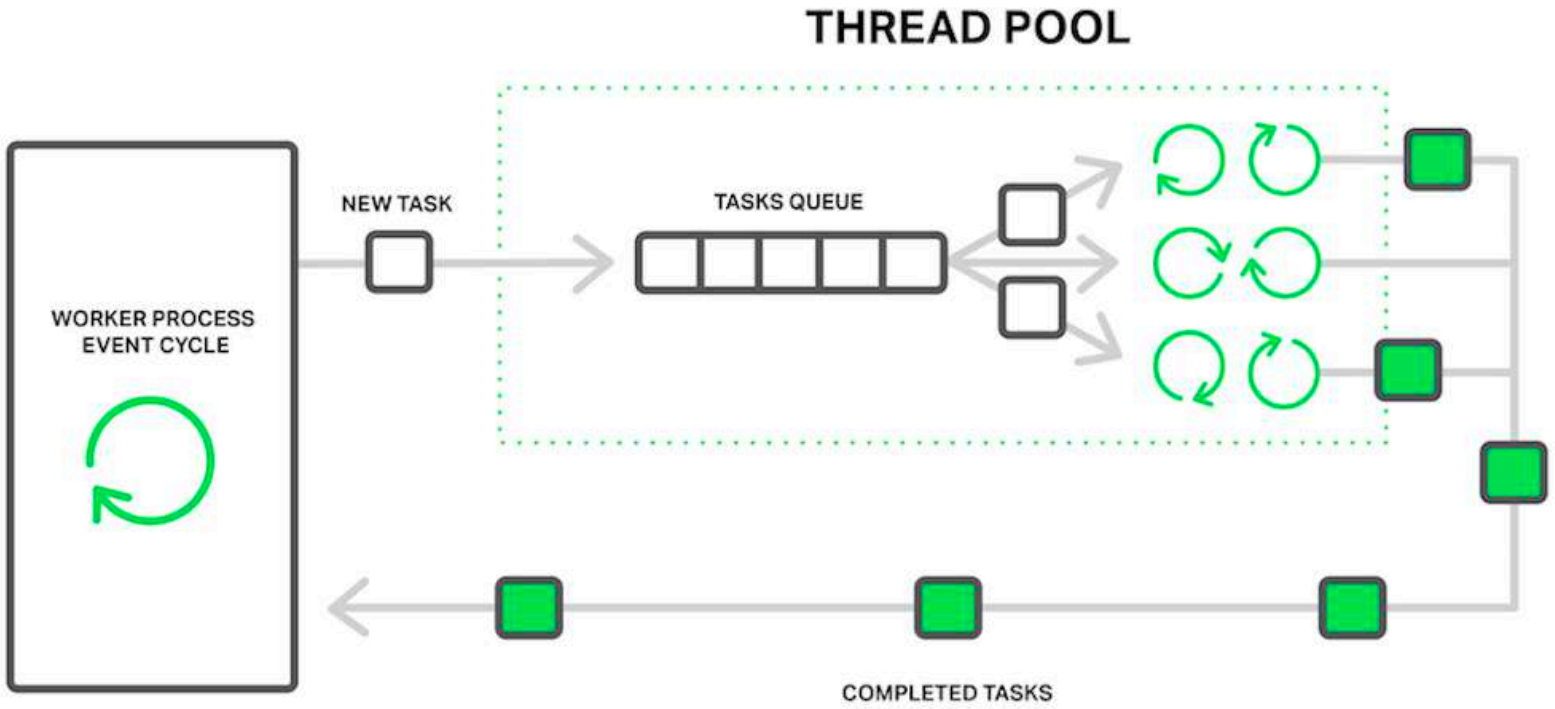
提升IOPS

access/error logging

减少频繁磁盘写入

Sendfile零拷贝

AIO / Thread pools



诀窍五：定位性能瓶颈

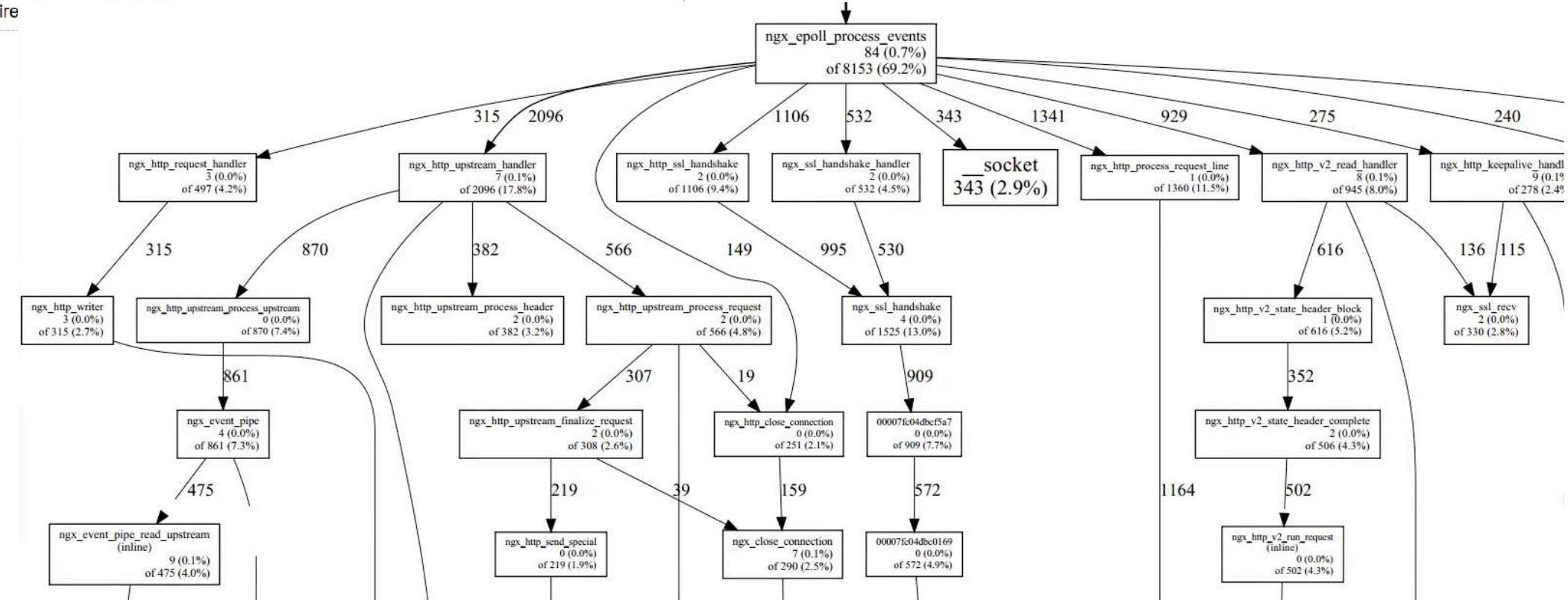
Module ngx_google_perftools_module

Example Configuration
Directives
google_perftools_profiles

The `ngx_google_perftools_module` module (0.6.29) enables profiling of nginx worker processes using Google Performance Tools. The module is intended for nginx developers.

This module is not built by default, it should be enabled with the `--with-google_perftools_module` configuration parameter.

This module require



实践是检验真理的唯一标准

反向代理

Web Server

客户端：wrk软件

配置2C4G
已实测可以打出4.5w RPS的能力
wrk -t4 -c800 -d120s http://10.1.10.146

反向代理：性能优化对象

配置2C4G
初始化nginx conf默认配置，逐步进行优化
已优化了linux os关键参数
通过dashboard实时观测RPS指标

Web端：NGINX

配置2C4G
已初步优化，支撑2.5W+ rps的能力
不会是性能瓶颈

NGINX

默认配置性能表现 - 6K RPS

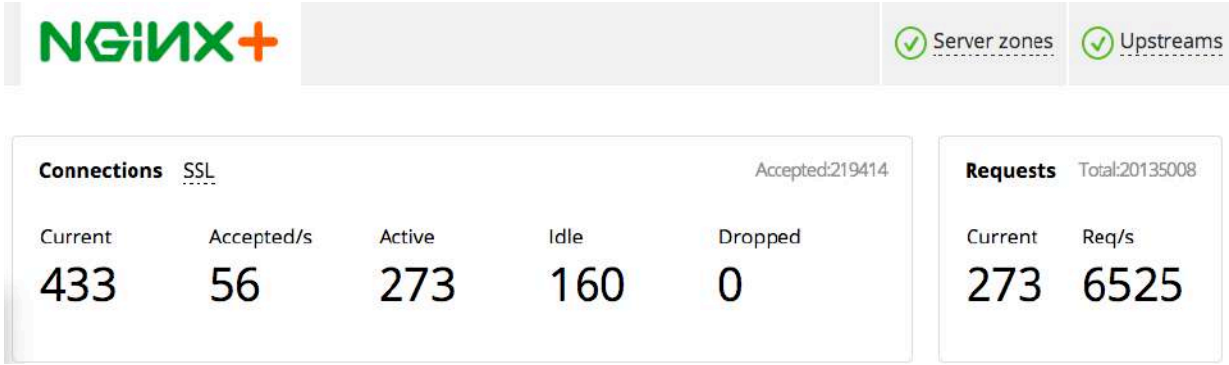
```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;

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;

    sendfile        on;
    #tcp_nopush     on;
    keepalive_timeout  65;
    #gzip  on;
    include /etc/nginx/conf.d/*.conf;
}
```



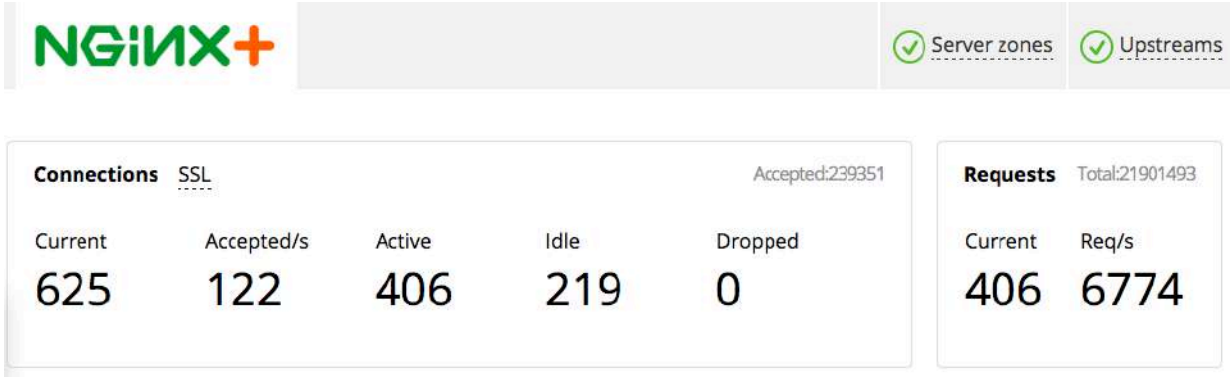

```
[root@localhost /]# wrk -t4 -c800 -d120s http://10.1.10.146
Running 2m test @ http://10.1.10.146
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   109.19ms  184.09ms   2.00s    95.82%
    Req/Sec     1.55k   492.78     3.60k    62.43%
  738668 requests in 2.00m, 607.91MB read
  Socket errors: connect 0, read 3408, write 0, timeout 4531
  Non-2xx or 3xx responses: 1
Requests/sec:   6152.86
Transfer/sec:      5.06MB
```

NGINX

优化连接数限制 – 6K RPS, Socket errors减少

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;

events {
    worker_connections  100000;
}
```

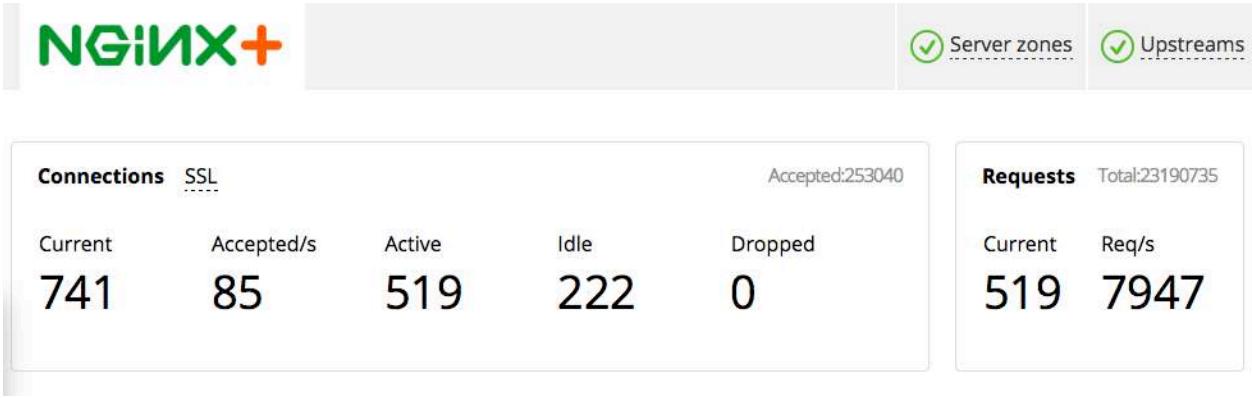



```
[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146
Running 2m test @ http://10.1.10.146
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   119.72ms  179.98ms   2.00s    95.11%
    Req/Sec     1.52k   527.92     6.03k    64.26%
  728057 requests in 2.00m, 599.17MB read
  Socket errors: connect 0, read 141, write 0, timeout 3602
Requests/sec:   6062.12
Transfer/sec:      4.99MB
```

优化worker数量 - 8K RPS，30%性能提升

```
user  nginx;
worker_processes  auto;
error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;

events {
    worker_connections  100000;
}
```



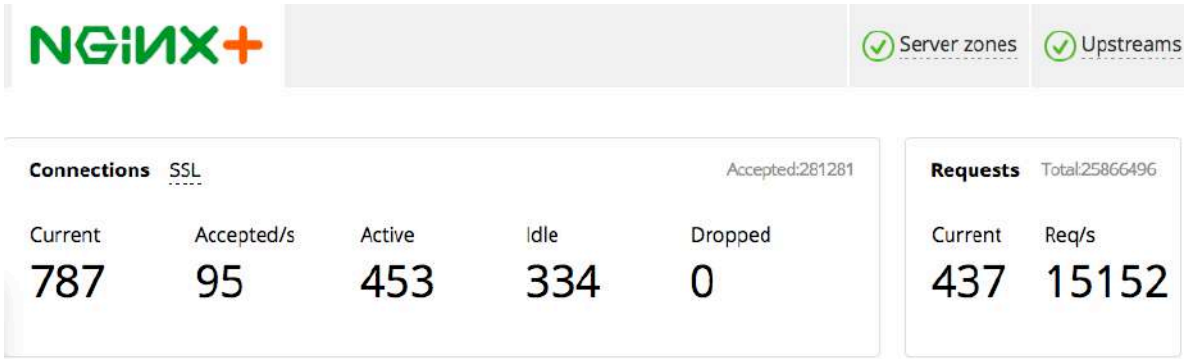

```
[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146
Running 2m test @ http://10.1.10.146
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   266.91ms  388.22ms   2.00s    84.04%
    Req/Sec     1.89k   409.68     3.96k    72.41%
  902374 requests in 2.00m, 742.63MB read
  Socket errors: connect 0, read 0, write 0, timeout 769
Requests/sec:   7513.67
Transfer/sec:      6.18MB
```

NGINX

配置连接复用 - 15K RPS，100%性能提升

```
upstream webserver {
   zone   backend 64k;
   #ip_hash;
   server 10.1.10.145:80;
   #server 10.1.10.150:80;
   keepalive 128;
}

server {
    listen 80 default;
    server_name  localhost;
    location / {
       proxy_pass http://webserver;
       proxy_http_version 1.1;
       proxy_set_header Connection "";
    }
}
```

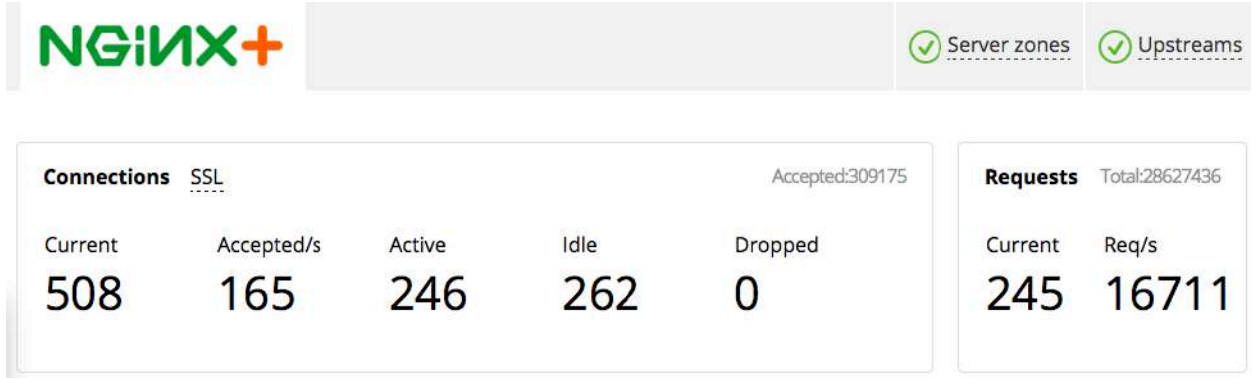



```
[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146
Running 2m test @ http://10.1.10.146
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   169.83ms  258.98ms   1.98s    83.91%
    Req/Sec     3.82k   731.30    10.81k    74.54%
  1826296 requests in 2.00m, 1.47GB read
  Socket errors: connect 0, read 0, write 0, timeout 38
Requests/sec:  15207.40
Transfer/sec:     12.52MB
```

配置CPU亲和及worker优先级
- 16K RPS，5%性能提升

```
user  nginx;
worker_processes  2;
worker_cpu_affinity 0101 1010;
error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;
worker_priority -20;

events {
    worker_connections  100000;
}
```




```
[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146
Running 2m test @ http://10.1.10.146
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   152.98ms  235.47ms   2.00s    83.66%
    Req/Sec     3.94k   759.08     7.39k    75.00%
  1881759 requests in 2.00m, 1.51GB read
  Socket errors: connect 0, read 0, write 0, timeout 310
Requests/sec:  15669.69
Transfer/sec:     12.90MB
```

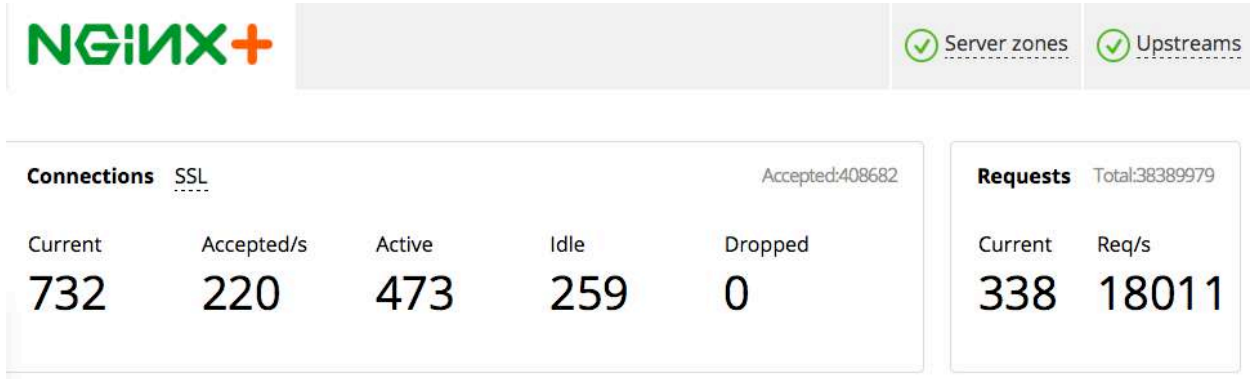
配置日志缓存 - 18K RPS，10%性能提升

```
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
buffer=1m;

    sendfile        on;
    #tcp_nopush     on;
    keepalive_timeout  65;
    #gzip  on;
    include /etc/nginx/conf.d/*.conf;
}
```



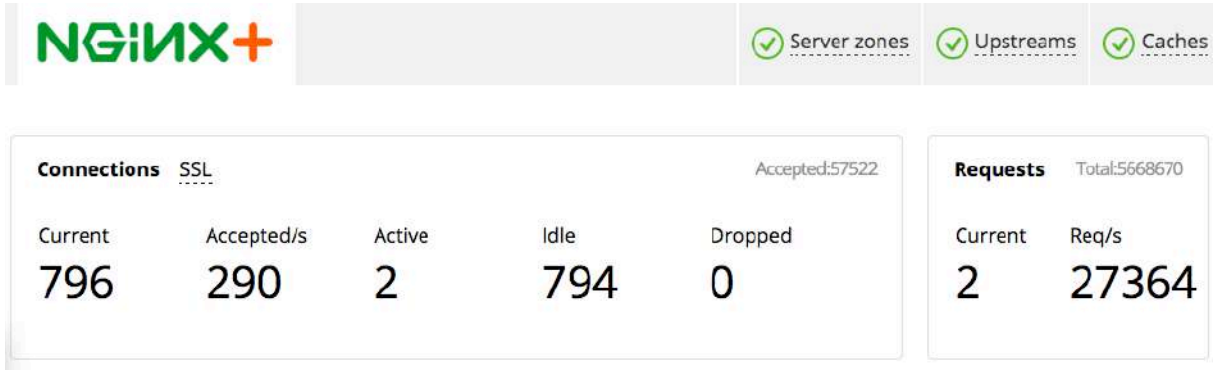

```
[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146
Running 2m test @ http://10.1.10.146
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   125.81ms  285.55ms   2.00s    90.16%
    Req/Sec     4.56k     0.90k   11.21k    72.98%
  2175234 requests in 2.00m, 1.75GB read
  Socket errors: connect 0, read 0, write 0, timeout 7993
Requests/sec:  18111.96
Transfer/sec:     14.91MB
```

关闭access log，19k，5%提升

配置Cache - 27K RPS，50%性能提升

```
proxy_cache_path /tmp/cache
keys_zone=mycache:10m inactive=60m;


server {
    listen 80 default;
    server_name  localhost;
    location / {
        proxy_pass http://webserver;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_cache_key
$host:$server_port$request_uri;
        proxy_cache_valid 200 304 1h;
        proxy_cache mycache;
    }
}
```




```
[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146/
Running 2m test @ http://10.1.10.146/
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    66.61ms  112.04ms   1.95s    88.35%
    Req/Sec     6.71k     1.70k   21.59k    81.55%
  3166292 requests in 2.00m, 2.54GB read
  Socket errors: connect 0, read 0, write 0, timeout 9
Requests/sec:  26370.78
Transfer/sec:     21.70MB
```

还能继续增长吗？CPU才使用80+%，加大电流

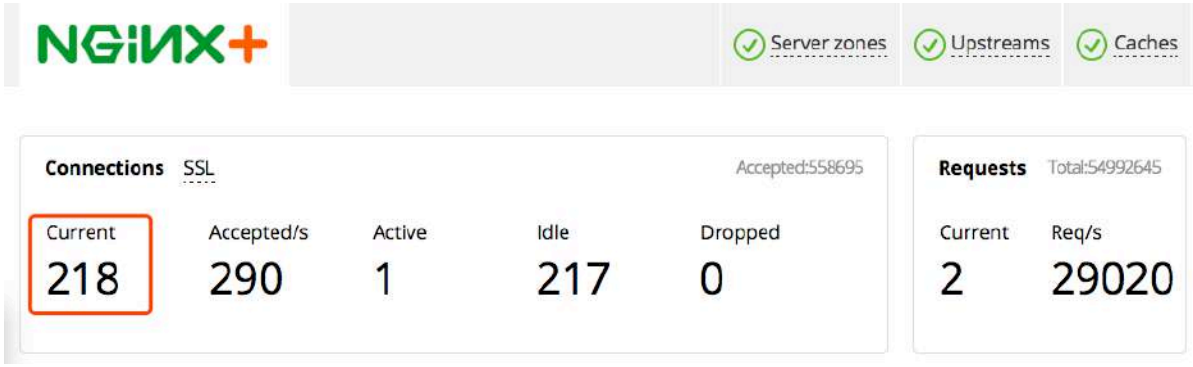
```
[[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146/
Running 2m test @ http://10.1.10.146/
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    66.61ms  112.04ms   1.95s    88.35%
    Req/Sec     6.71k     1.70k   21.59k    81.55%
  3166292 requests in 2.00m, 2.54GB read
  Socket errors: connect 0, read 0, write 0, timeout 9
Requests/sec:  26370.78
Transfer/sec:     21.70MB
```

```
[[root@localhost wrk]# wrk -t8 -c1800 -d120s http://10.1.10.146/
Running 2m test @ http://10.1.10.146/
  8 threads and 1800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   141.62ms  208.10ms   2.00s    86.09%
    Req/Sec     3.65k     1.43k   19.87k    72.37%
  3408019 requests in 2.00m, 2.74GB read
  Socket errors: connect 0, read 0, write 0, timeout 112
Requests/sec:  28380.12
Transfer/sec:     23.36MB
```

RPS小幅度增长，延迟却翻倍了！！网络带宽瓶颈了?

试试gzip压缩 - 29K RPS，5%性能提升

```
http {
    ……
    gzip on;
    gzip_min_length  500;
    gzip_buffers     4 256k;
    gzip_http_version 1.1;
    gzip_comp_level 1;
    gzip_types     text/plain application/javascript
application/x-javascript text/javascript text/css
application/xml application/xml+rss;
    gzip_vary on;
    gzip_proxied    expired no-cache no-store
private auth;
    gzip_disable   "MSIE [1-6]\.";
    ……
}
```

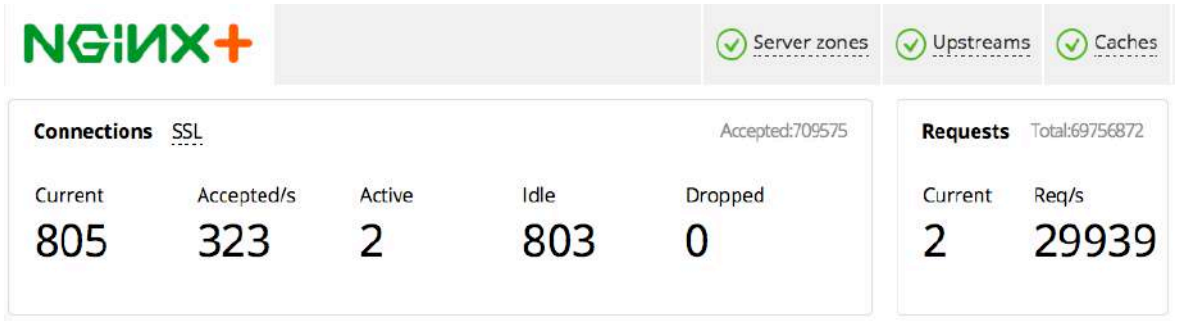



```
[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146/
 -H "Accept-Encoding: gzip"
Running 2m test @ http://10.1.10.146/
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   255.11ms  491.35ms   2.00s    84.35%
    Req/Sec     6.55k     1.97k   24.26k    80.40%
  3102377 requests in 2.00m, 1.96GB read
  Socket errors: connect 0, read 0, write 0, timeout 8821
Requests/sec:  25832.05
Transfer/sec:     16.75MB
```

CPU 100%，响应延迟剧增

优化CPU开销 - 30K RPS，5 %性能提升，延迟大幅优化

```
events {
    multi_accept on;
    accept_mutex on;
    accept_mutex_delay 1ms;
    worker_connections 100000;
}
```




```
[root@localhost wrk]# wrk -t4 -c800 -d120s http://10.1.10.146/]
 -H "Accept-Encoding: gzip"
Running 2m test @ http://10.1.10.146/
  4 threads and 800 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    28.92ms   10.00ms 117.70ms   74.91%
    Req/Sec     6.80k     0.86k   13.59k    78.65%
  3239397 requests in 2.00m, 2.05GB read
Requests/sec:  26977.71
```

性能优化实践总结

合适worker 进程数
CPU亲和
提升worker进程优先级
优化连接数限制
连接复用
日志缓存
cache
gzip
CPU开销优化

6K RPS

109 ms

wrk大量报错

30K RPS

29 ms

无任何报错

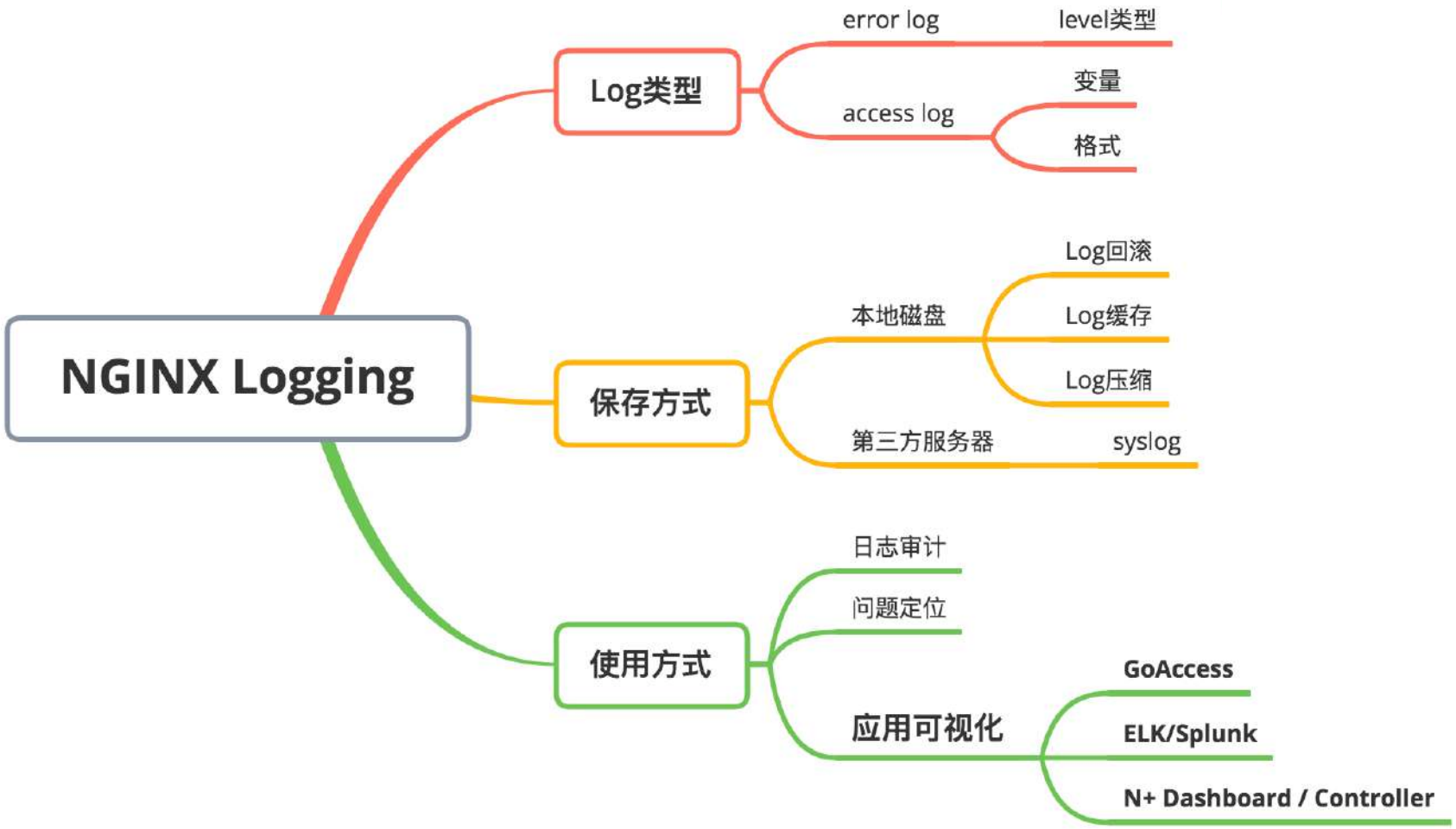
还有空间吗？一定有！Just do IT!

NGINX从入门到精通进阶系列培训

应用篇：全面洞察NGINX应用状态

廖健雄 j.liao@f5.com

全面洞察NGINX应用状态



变量 – 洞察NGINX应用状态的钥匙

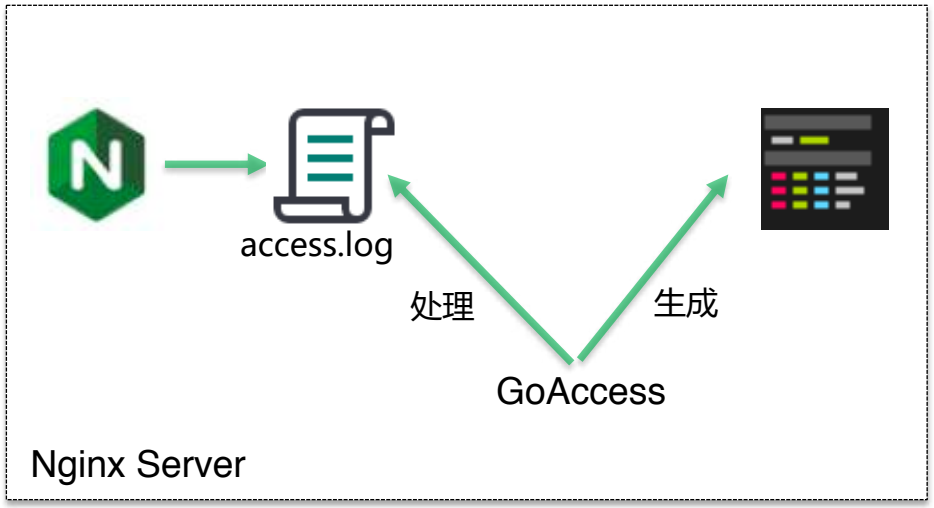
客户端信息	IP	$remote_addr $http_x_forward_for
	地址位置	$geoip_*
	浏览器类型	$modern_browser $ancient_browser $msie $http_user_agent
	协议	$https $http2
	ssl信息	$ssl_cipher $ssl_ciphers $ssl_client_*
客户端行为	请求行信息	$request $request_*
	请求header任何字段	$http_*
	上下文页面	$http_referer
	是否复用ssl session	$ssl_session_id $ssl_session_reused
	使用的ssl协议	$ssl_protocol

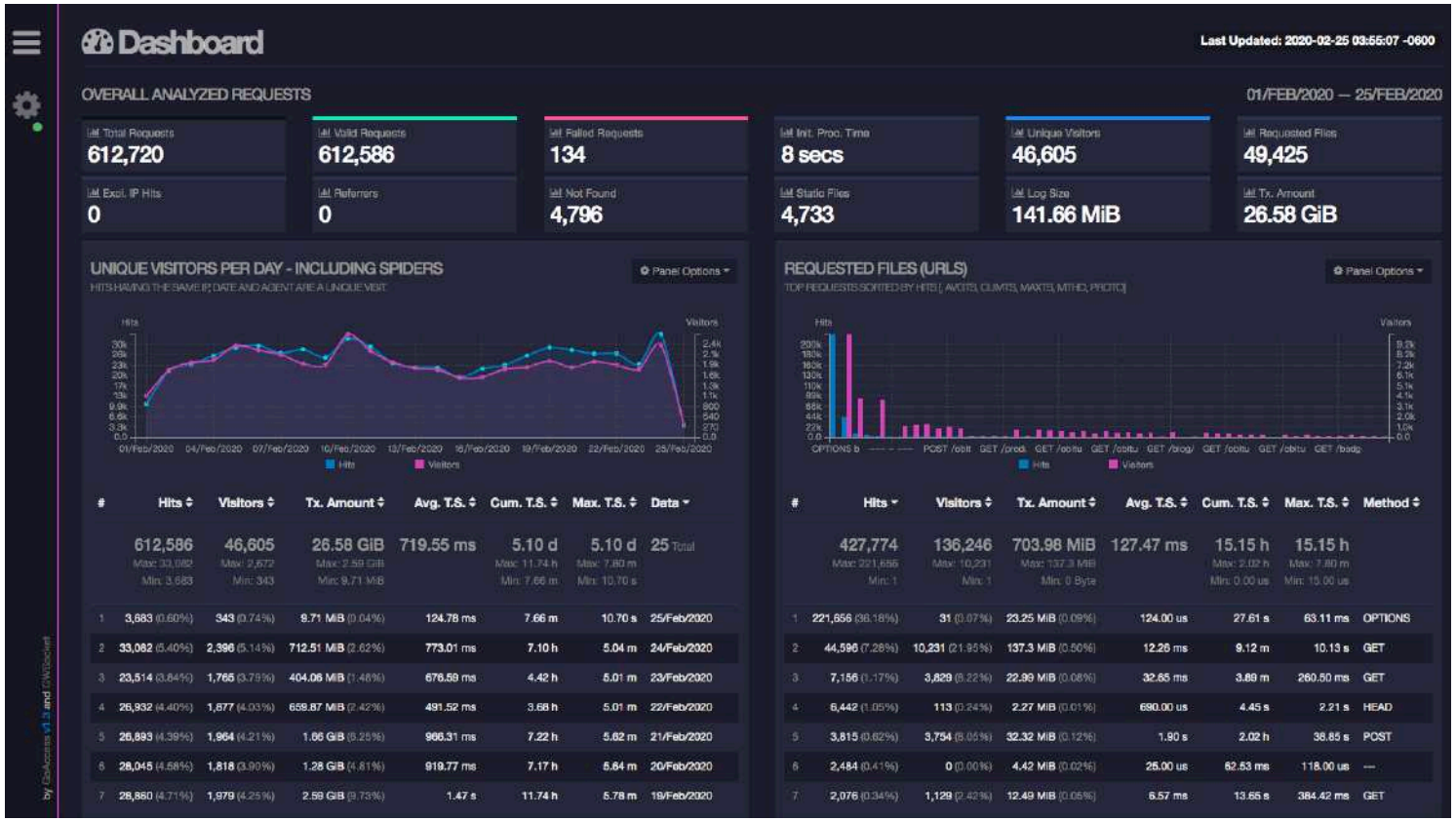
NGINX信息	nginx 版本	$nginx_version
	处理请求的worker进程id	$pid
	当前时间	$local_time
NGINX运行状态	当前连接总数	$connections_active
	分别处于reading、writing和waiting状态的连接数量	$connections_reading $connections_writing $connections_waiting

应用侧信息	应用域名	$host
	服务器信息	$upstream_addr
	响应的header字段	$upstream_http_*
应用侧状态	响应状态码	$status
	接收和发送的数据包大小	$upstream_bytes_sent $upstream_bytes_received
	响应延迟情况	$upstream_connect_time $upstream_first_byte_time $upstream_session_time $upstream_response_time

应用可视化之GoAccess

GoAccess 是一款开源的且具有交互视图界面的**实时 Web 日志分析工具**，能为系统管理员提供**快速**且有价值的 HTTP 统计，并以在线可视化服务器的方式呈现。





GoAccess优劣分析

```
[root@rp1 nginx]# /usr/local/bin/goaccess /var/log/nginx/access.
log -o /usr/share/nginx/html/goaccess.html  --real-time-html
/var/log/nginx/access.log
Parsed 10 lines producing the following errors:

Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'
Token 'Mozilla/5.0' doesn't match specifier '%s'

Format Errors - Verify your log/date/time format
```

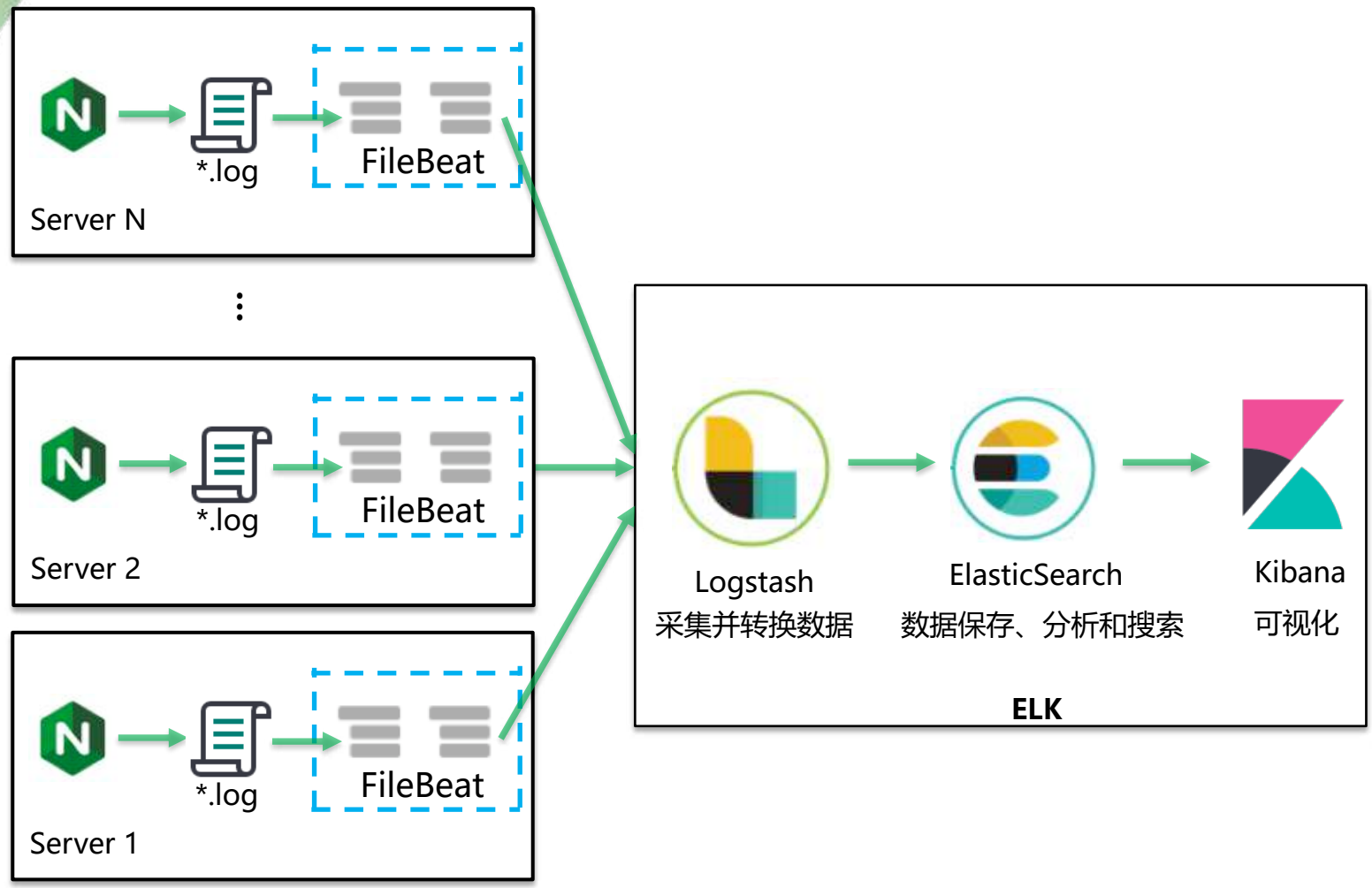
优势

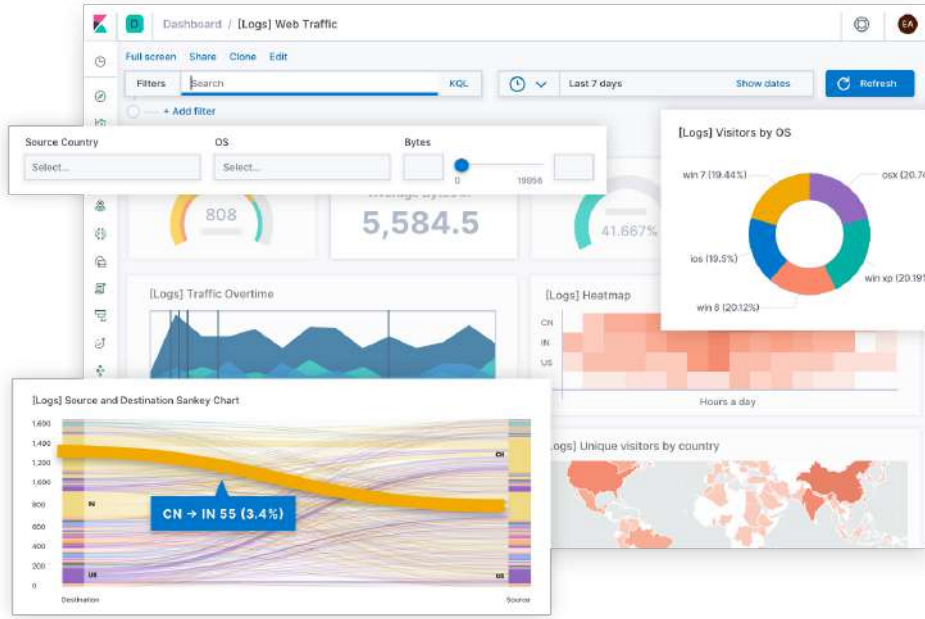
简单易用

劣势

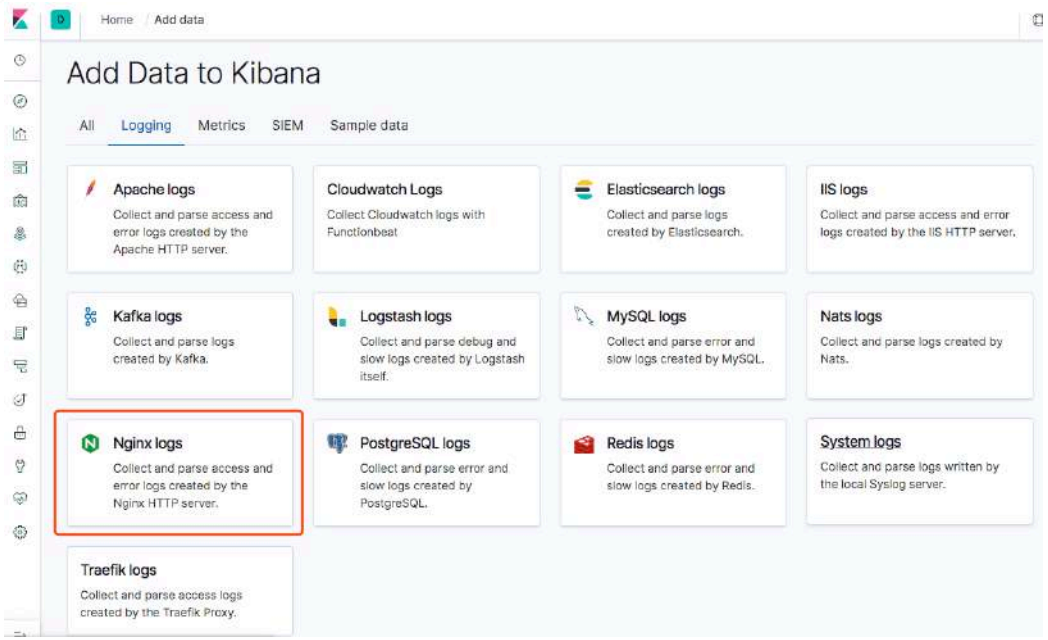
功能比较单一
花费大量精力实现格式匹配
单机版，不适用于分布式部署的场景

应用可视化之ELK


Server N
*.log
FileBeat
Server 2
*.log
FileBeat
Server 1
*.log
FileBeat
Logstash
采集并转换数据
ElasticSearch
数据保存、分析和搜索
Kibana
可视化
ELK


Dashboard / [Logs] Web Traffic
[Logs] Visitors by OS
[Logs] Traffic Overtime
[Logs] Heatmap
[Logs] Source and Destination Sankey Chart
CN → IN 55 (3.4%)
[Logs] Unique visitors by country

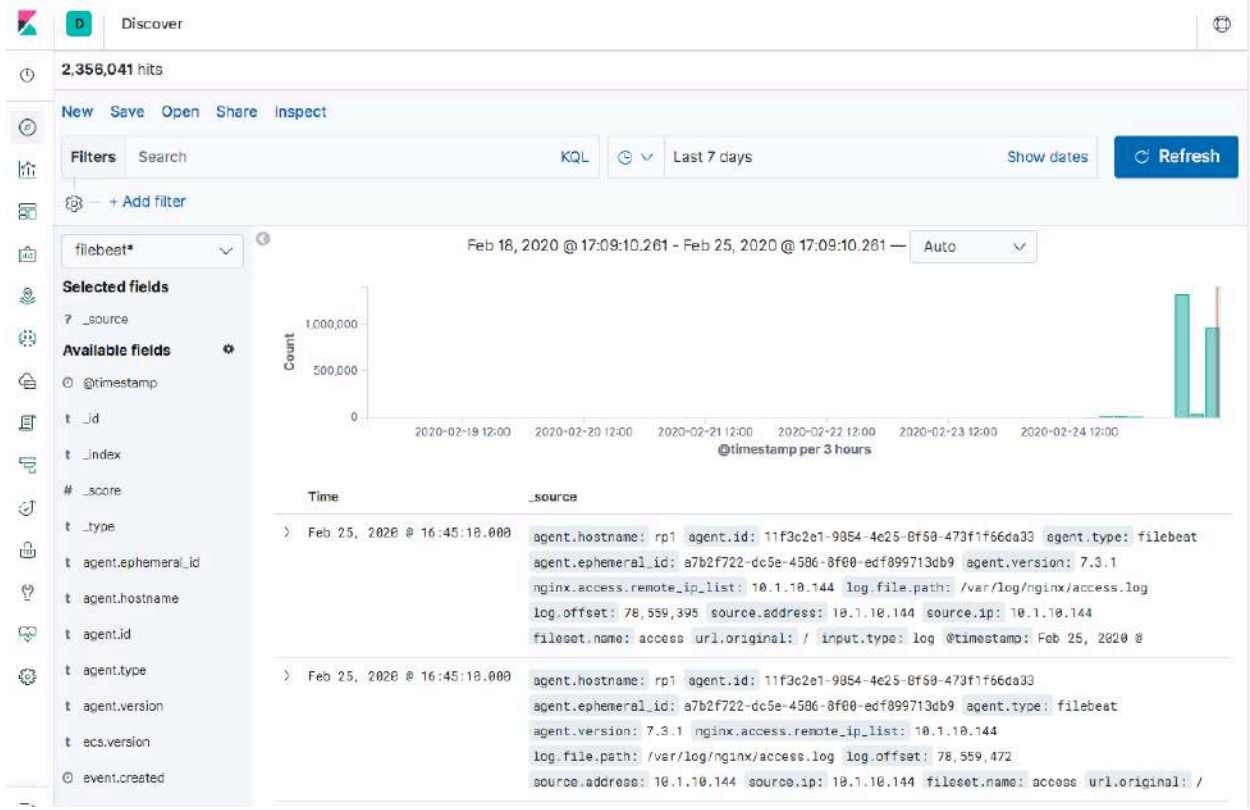
应用可视化之ELK



1. 安装filebeat
2. 配置filebeat

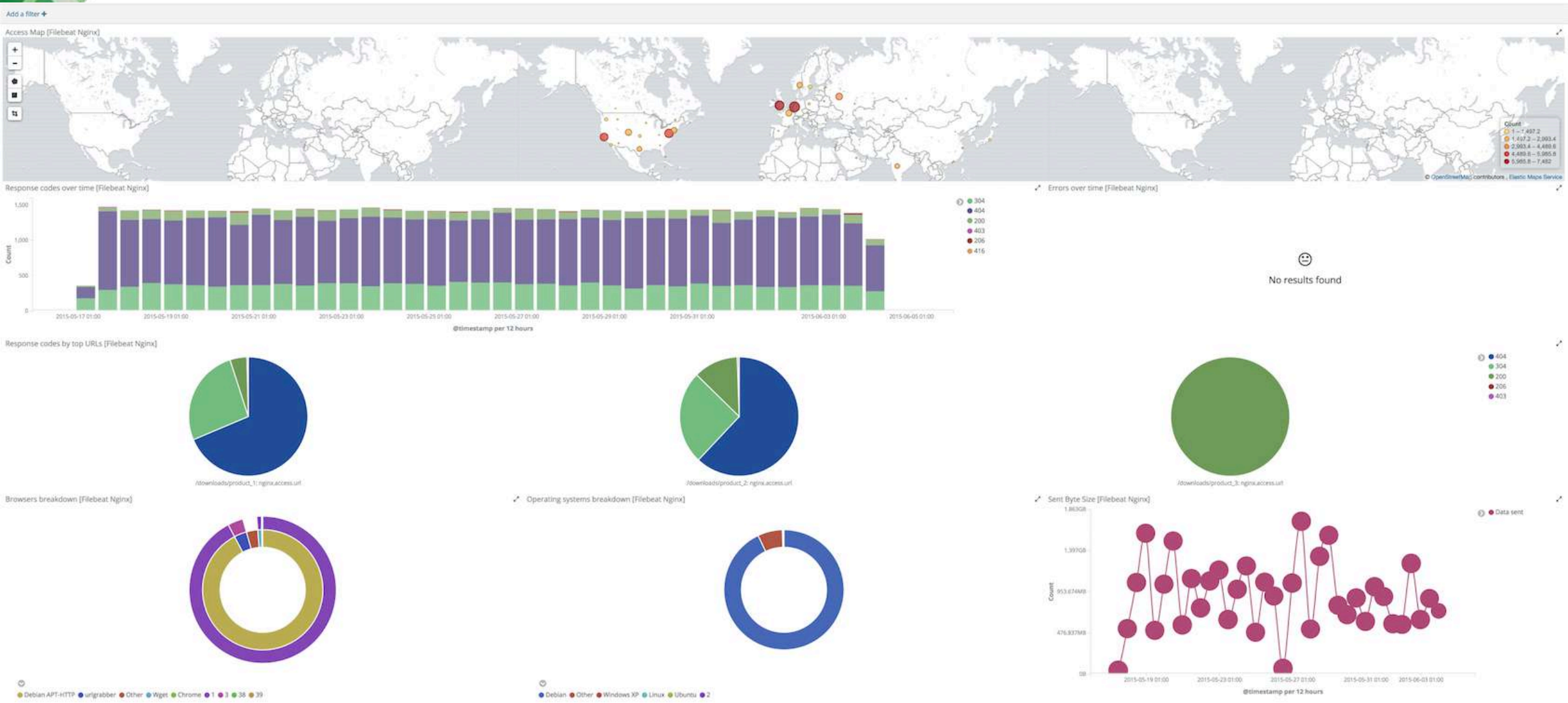
```
output.elasticsearch:
  hosts: ["<es_url>"]
  username: "elastic"
  password: "<password>"
setup.kibana:
  host: "<kibana_url>"
```

3. 启动filebeat
4. 发起业务请求

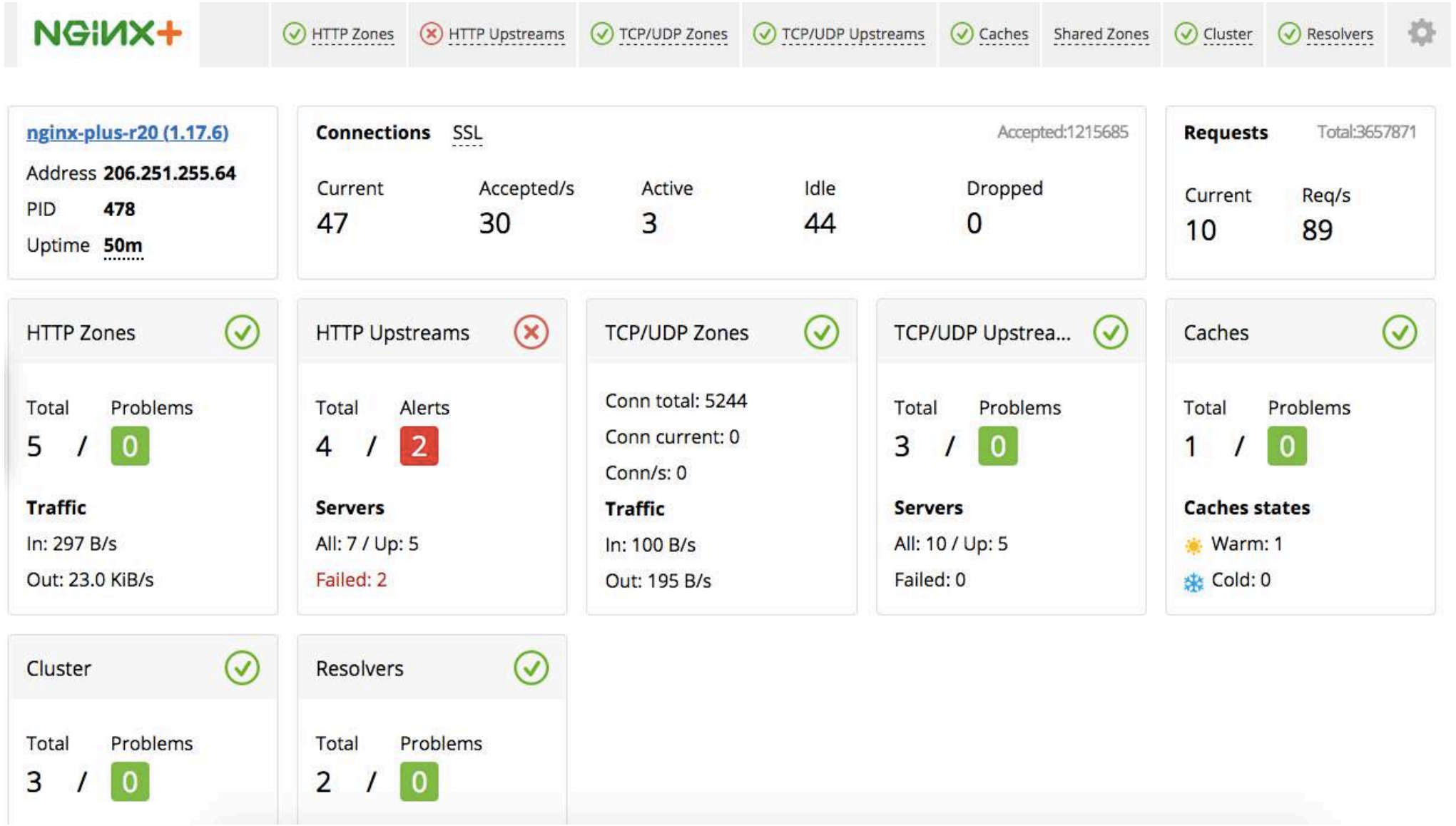


NGINX

应用可视化之ELK效果展示



应用可视化之NGINX Plus Dashboard



https://docs.nginx.com/nginx/admin-guide/monitoring/live-activity-monitoring/

应用可视化之NGINX Plus Dashboard

Server Zones

Zone	Requests			Responses						Traffic			
	Current	Total	Req/s	1xx	2xx	3xx	4xx	5xx	Total	Sent/s	Rcvd/s	Sent	Rcvd
backend	0	23824	0	0	0	0	0	23824	23824	0	0	7.13 MiB	1.93 MiB
frontend.test	1	12684682	16	0	12635410	11	1612	47648	12684681	35.3 KiB	6.45 KiB	13.8 GiB	3.09 GiB

Location Zones

Zone	Requests		Responses						Traffic			
	Total	Req/s	1xx	2xx	3xx	4xx	5xx	Total	Sent/s	Rcvd/s	Sent	Rcvd
dashb	2778658	13	0	2777024	11	1623	0	2778646	8.90 KiB	6.25 KiB	4.25 GiB	2.00 GiB
synced-backends	6	0	0	6	0	0	0	6	0	0	1.58 KiB	570 B
synced-backends2	3	0	0	3	0	0	0	3	0	0	810 B	288 B
tests	0	0	0	0	0	0	0	0	0	0	0	0

HTTP Upstreams

Show upstreams list

Failed only

backend

Zone: 40 %

Show all

Server			Requests		Responses			Conns		Traffic				Server checks		Health monitors				Response time	
Name	DT	W	Total	Req/s	...	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers	Response
192.168.128.20:8081	0ms	1	3	0		0	0	0	∞	0	0	291 B	726 B	0	0	2848366	0	0	passed	0ms	0ms
192.168.128.20:8080	0ms	1	3	0		0	0	0	∞	0	0	291 B	726 B	0	0	2848420	0	0	passed	0ms	0ms

backend2

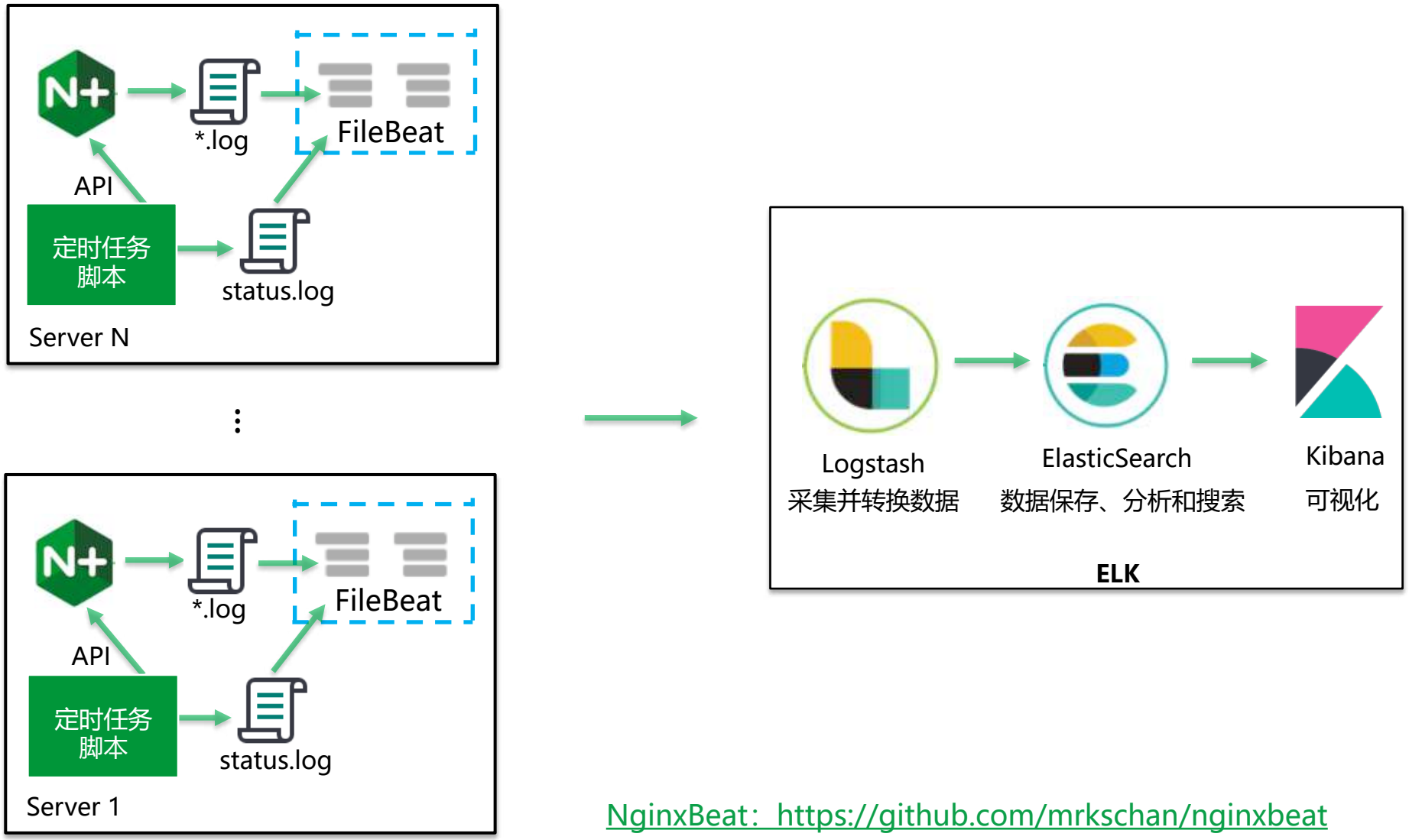
Zone: 40 %

Show all

Server			Requests		Responses			Conns		Traffic				Server checks		Health monitors				Response time	
Name	DT	W	Total	Req/s	...	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers	Response
192.168.128.20:8081	0ms	1	2	0		0	0	0	∞	0	0	198 B	484 B	0	0	2848366	0	0	passed	0ms	0ms
192.168.128.20:8081	0ms	1	1	0		0	0	0	∞	0	0	99.0 B	242 B	0	0	2848366	0	0	passed	0ms	0ms

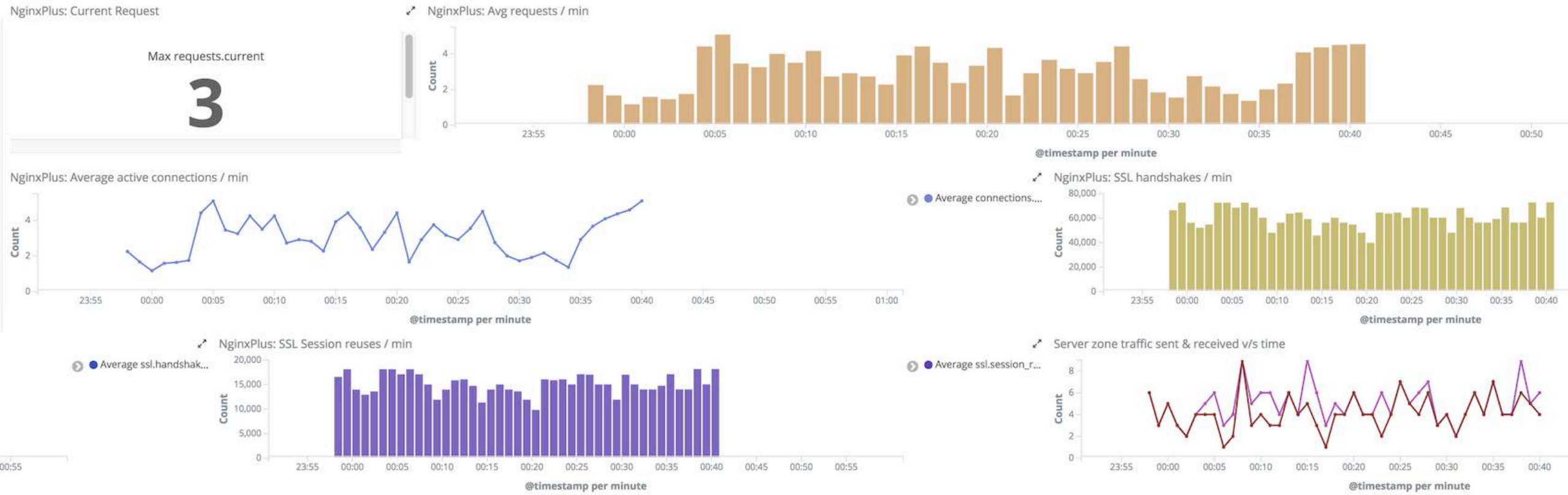
NGINX

应用可视化之NGINX Plus + ELK

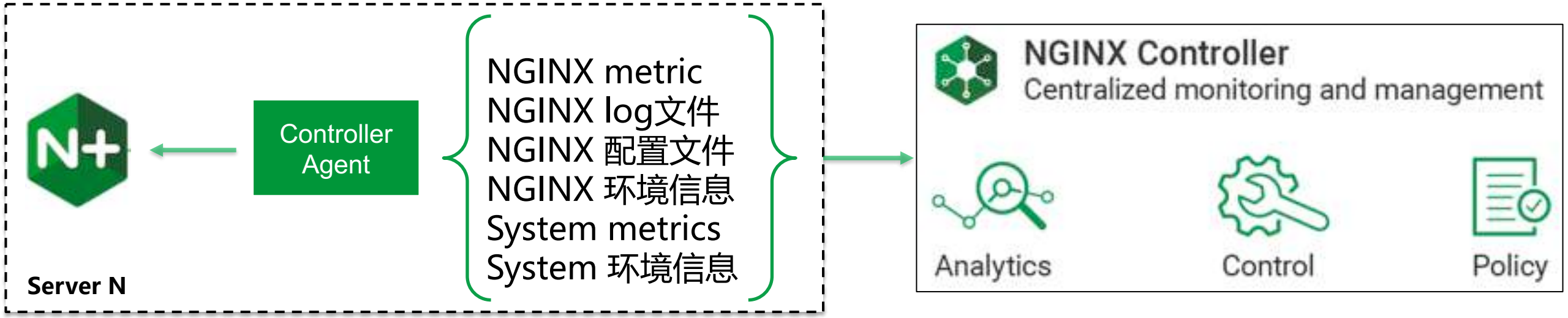


NginxBeat：https://github.com/mrkschan/nginxbeat

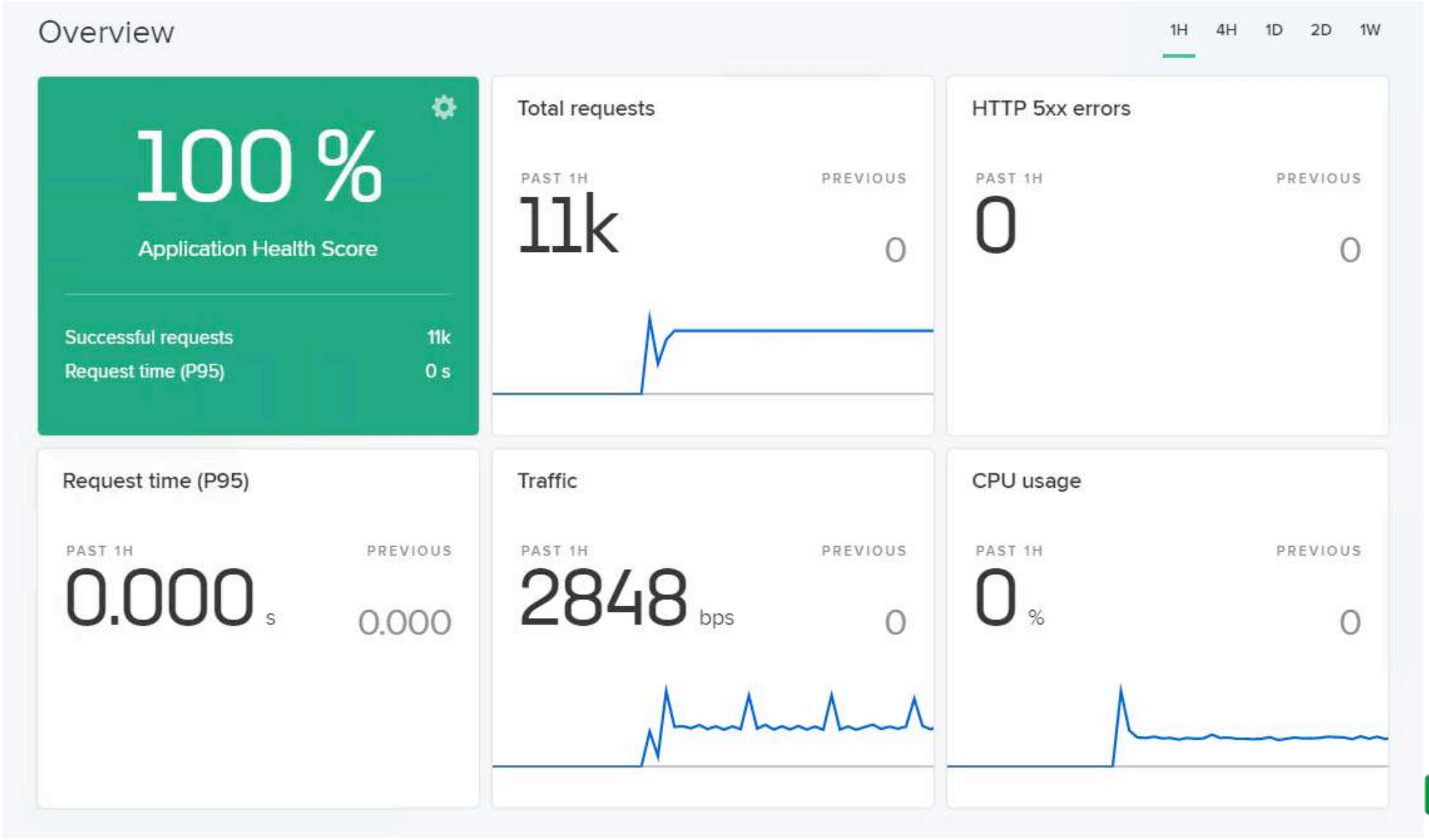
应用可视化之NGINX PLUS + ELK 效果展示



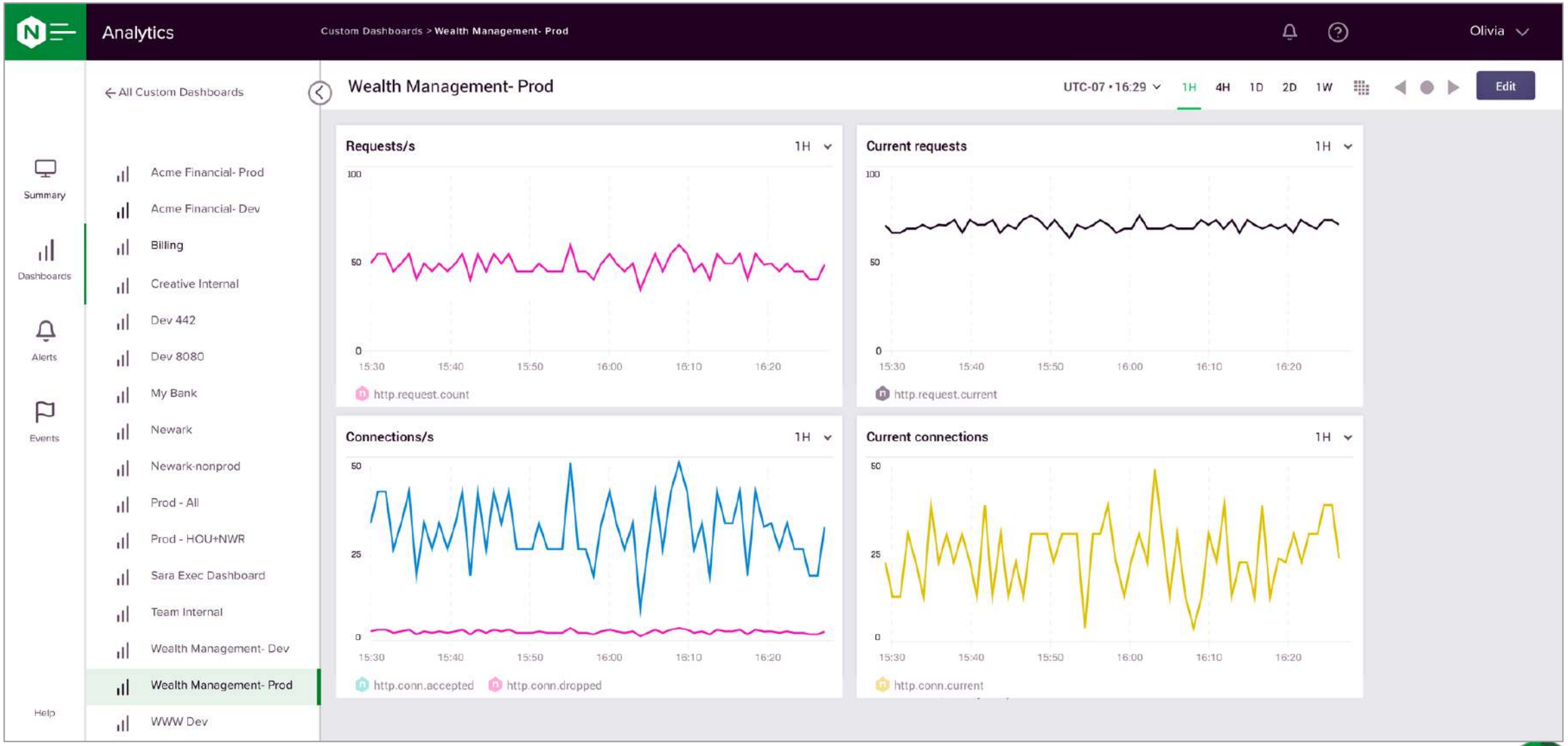
应用可视化之NGINX Plus + NGINX Controller



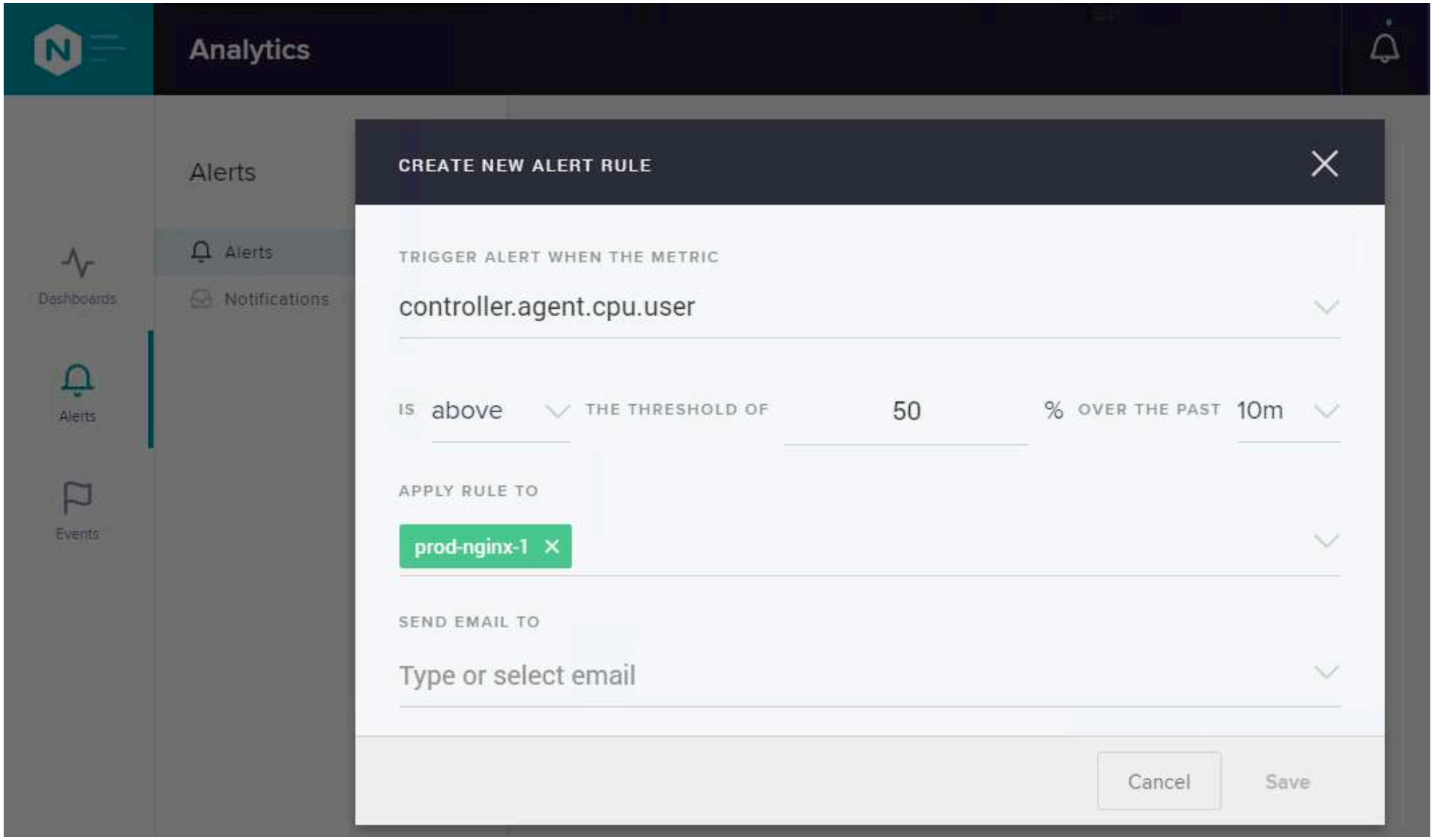
应用可视化之NGINX Plus + NGINX Controller



应用可视化之NGINX Plus + NGINX Controller



应用可视化之NGINX Plus + NGINX Controller



关注我们

F5 官方微信
（新闻，技术文章）

F5社区
（答疑，吐槽，分享，互动）

加入F5社区：关注“F5社区”微信公众号，注册成为社员，随时参加meet up活动，代码共享，讨论，答疑等。只要你有想法，有创造，那么就快来大展身手吧，让我们在社区里尽情分享，交流，吐槽和互动，在这个自由的国度里，发现闪亮的自己。让我们一起来见证“一群有才能的人在一起做有梦想的事！

NGINX技术群

操作步骤：

1. 扫描二维码并在“入群信息”栏填写姓名
2. 点击下方“我要入群”
3. 长按识别二维码进入群聊


f5
N
Thank You