

Мониторинг

Golang Developer. Professional



Проверить, идет ли запись

Меня хорошо видно & слышно?






Хохлов Александр

Архитектор платформенных решений в ГК Иннотех

- **25+** лет в Информационных технологиях, последние 10 лет - архитектура и все что вокруг нее
- Занимаюсь вопросами проектирования, разработки (db/back/front/mobile), инфраструктуры, управления и развития технических команд.
- Преподаватель курсов:
 - Golang Developer. Basic
 - Golang Developer. Professional
 - System Design
 - Highload Architecture

 @khorost

 alexander.khokhlov@gmail.com

Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в учебной группе **#канал
группы**



Задаем вопрос
в чат или голосом



Вопросы вижу в чате,
могу ответить не сразу

Карта курса



СТАРТ

Начало работы с Go

Concurrency в Go

Работа с сетью и БД

Стандартные библиотеки и практики

Микросервисы

Проектная работа

ФИНИШ



Маршрут вебинара

Observability и Operability

Метрики и мониторинг

Мониторинг ресурсов, приложений и БД

Prometheus

Observability и Operability

Observability

Observability (наблюдаемость) - мера того, насколько по выходным данным можно восстановить информацию о состоянии системы.

Примеры:

- логирование (zap, logrus -> fluentd -> kibana)
- мониторинг (zabbix, prometheus)
- алертинг (chatops, pagerduty, opsgenie)
- трейсинг (jaeger, zipkin, opentracing, datadog apm, signoz.io)
- профилирование (pprof)
- сбор ошибок и аварий (sentry, hawk-tracker.ru)



Operability

Operability (работоспособность) - мера того, насколько приложение умеет сообщать о своем состоянии здоровья, а инфраструктура управлять этим состоянием.

Примеры:

- простейшие хелсчеки
- liveness и readiness в Kubernetes



Docker Health Check

```
FROM nginx:latest

# Copy the web app content into the container
COPY ./index.html /usr/share/nginx/html/index.html

# Define the health check
HEALTHCHECK --interval=30s --timeout=5s --retries=3 \
  CMD curl --fail http://localhost || exit 1
```

Ключом **--restart** можно управлять восстановлением работоспособности контейнера

```
version: '3'
services:
  web:
    image: nginx
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost"]
      interval: 30s
      retries: 3
      start_period: 5s
      timeout: 10s
```



Зачем нужен мониторинг?

- Отладка, решение текущих проблем
- [SRE мир](#):
 - SLA (Service Level Agreement)
 - SLO (Service Level Objective)
 - SLI (Service Level Indicator)
- Отправка уведомлений
- Технические и бизнесовые A/B эксперименты
- Анализ трендов, прогнозирование

[SRE \(Site Reliability Engineering\) book](#)



Вопросы



если есть вопросы



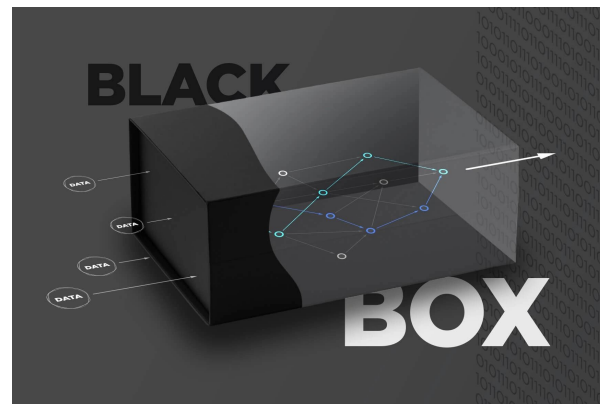
если вопросов нет



Метрики и мониторинг

Виды мониторинга

- Количественный / Событийный (y/n)
- Whitebox / Blackbox
- Push / Pull



Push vs Pull

Push - агент, работающий на окружении (например, сайдкар), подключается к серверу мониторинга и отправляет данные.

Особенности:

- мониторинг специфических/одноразовых задач
- может работать за NAT
- не нужно открывать никакие URL'ы/порты на стороне приложения
- из приложения нужно конфигурировать подключение

Примеры: ``graphite``, ``statsd``

Pull - сервис мониторинга сам опрашивает инфраструктуры/сервисы и агрегирует статистику.

Особенности:

- не нужно подключаться к агенту на стороне приложения
- нужно открывать URL или порт, который будет доступен сервису мониторинга
- более отказоустойчивый
- не требует авторизации /верификации источника

Примеры: ``datadog-agent``, ``prometheus``



Load Average

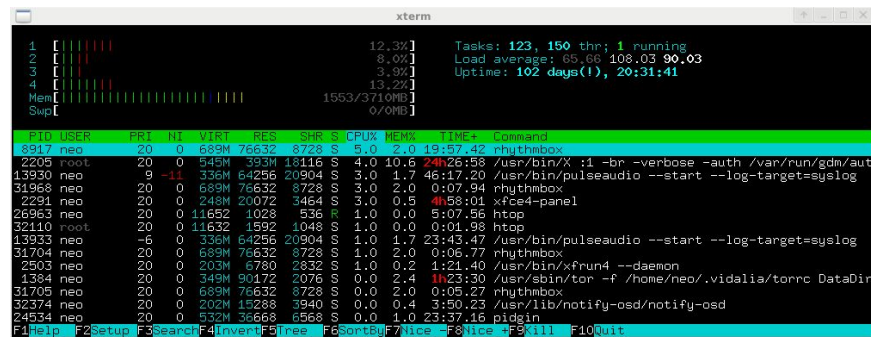
LA - сложная метрика, ее можно интерпретировать как количество процессов (поточков) в ОС, находящихся в ожидании какого-либо ресурса (чаще всего CPU или диск).

Нормальной считается загрузка когда LA ~ числу ядер процессора.

Как посмотреть:

- top
- iostat, dstat

```
$ top
Processes: 420 total, 3 running, 417 sleeping, 1860 threads
Load Avg: 2.47, 2.53, 2.38
```



CPU

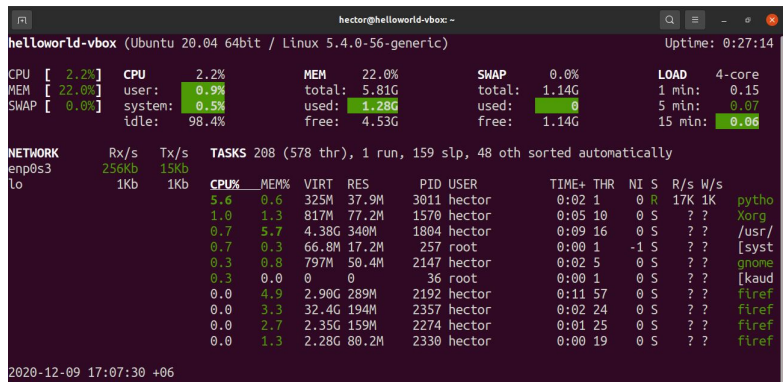
- User (`usr`, `us`) - процессор выполняет код программ. Высокое значение может быть признаком неоптимальных алгоритмов.
- System (`sys`, `sy`) - процессор выполняет код ядра. Высокое значение может означать большое кол-во операций ввода/вывода или сетевых пакетов.
- Wait (`wai`, `wa`) - процессор находится в ожидании ввода/вывода. Высокое значение означает недостаточную мощность дисковой системы.
- Idle (`id`) - процессор бездействует.
- Steal time (st) - переподписка в VM

Как посмотреть: top, htop

```
$ top
```

```
Processes: 419 total, 2 running, 417 sleeping, 1868 threads
```

```
CPU usage: 5.11% user, 4.39% sys, 90.48% idle
```



The screenshot shows the htop terminal window on a Linux system. The top section displays system statistics: CPU usage is 2.2% (user: 0.9%, system: 0.5%, idle: 98.4%), MEM usage is 22.0% (total: 5.81G, used: 1.28G, free: 4.53G), SWAP usage is 0.0%, and LOAD is 4-core (1 min: 0.15, 5 min: 0.07, 15 min: 0.06). The bottom section shows a list of running processes, sorted by CPU usage. The processes listed include python, Xorg, /usr/, [syst], gnome, [kaid], firef, and firef.

| NAME | STATE | CPU% | MEM% | VIRT | RES | PID | USER | TIME+ | THR | NI | S | R/s | W/s | COMMAND |
|--------|-------|------|------|-------|-------|------|--------|-------|-----|----|---|-----|-----|---------|
| python | S | 5.6 | 0.6 | 325M | 37.9M | 3011 | hector | 0:02 | 1 | 0 | R | 17K | 1K | python |
| Xorg | S | 1.0 | 1.3 | 817M | 77.2M | 1570 | hector | 0:05 | 10 | 0 | S | ? | ? | Xorg |
| /usr/ | S | 0.7 | 5.7 | 4.38G | 340M | 1804 | hector | 0:09 | 16 | 0 | S | ? | ? | /usr/ |
| [syst] | S | 0.7 | 0.3 | 66.8M | 17.2M | 257 | root | 0:00 | 1 | -1 | S | ? | ? | [syst] |
| gnome | S | 0.3 | 0.8 | 797M | 50.4M | 2147 | hector | 0:02 | 5 | 0 | S | ? | ? | gnome |
| [kaid] | S | 0.3 | 0.0 | 0 | 0 | 36 | root | 0:00 | 1 | 0 | S | ? | ? | [kaid] |
| firef | S | 0.0 | 4.9 | 2.90G | 289M | 2192 | hector | 0:11 | 57 | 0 | S | ? | ? | firef |
| firef | S | 0.0 | 3.3 | 32.4G | 194M | 2357 | hector | 0:02 | 24 | 0 | S | ? | ? | firef |
| firef | S | 0.0 | 2.7 | 2.35G | 159M | 2274 | hector | 0:01 | 25 | 0 | S | ? | ? | firef |
| firef | S | 0.0 | 1.3 | 2.28G | 80.2M | 2330 | hector | 0:00 | 19 | 0 | S | ? | ? | firef |

Memory

- Resident (`Res`/`RSS`) - память, занятая данными программ (как правило кучей). Высокое значение может говорить об утечках памяти в программах.
- Shared (`Shr`) - память, разделяемая между разными процессами (как правило сегменты кода).
- Cached - дисковый кеш операционной системы, в нагруженных системах (СУБД) состоянии занимает все свободное место.
- Free - незанятая память.

Как посмотреть:

- top
- free

```
$ top
SharedLibs: 249M resident, 42M data, 35M linkedit.
MemRegions: 164838 total, 2815M resident, 134M private, 1737M shared.
PhysMem: 12G used (2571M wired), 4457M unused.
```

```
crasic@crasic-desktop: ~
File Edit View Terminal Help
top - 11:09:28 up 2 days, 2:32, 2 users, load average: 0.56, 0.49, 0.37
Tasks: 141 total, 1 running, 140 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.0%us, 0.7%sy, 0.0%ni, 97.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 5812 crasic    20   0  451m 159m 35m  S   1.7  15.9   70:53.46 firefox-bin
3081 crasic    20   0  269m  83m  56m  S   0.0   8.3   4:08.88 soffice.bin
 785 root       20   0  95664 53m  12m  S   0.3   5.3  156:52.14 Xorg
2965 crasic    20   0  95868 19m  11m  S   0.0   2.0   1:07.87 evince
1241 crasic    20   0  95312 17m  1m  S   0.0   1.7   1:25.96 nautilus
4645 crasic    20   0  144m  17m  10m  S   0.0   1.7   0:55.24 gnome-screensho
1242 crasic    20   0  49596 14m  10m  S   0.0   1.5   2:06.67 gnome-panel
1427 crasic    20   0  31488 14m  8384  S   0.0   1.5   0:00.45 python
1282 crasic    20   0  87280 14m  10m  S   0.0   1.4   0:11.55 clock-applet
1270 crasic    20   0  47724 13m  10m  S   0.0   1.3   5:30.32 wnck-applet
1288 crasic    20   0  55212 12m  10m  S   0.0   1.3   0:04.14 indicator-apple
1219 crasic    20   0  116m  12m  10m  S   0.0   1.3   2:46.50 metacity
1237 crasic    20   0  66920 12m  9976  S   0.0   1.3   0:31.41 nm-applet
5981 crasic    20   0  53000 12m  9.8m  S   0.0   1.3   0:00.82 gnome-terminal
1289 crasic    20   0  54936 12m  9828  S   0.0   1.2   0:00.52 indicator-apple
1271 crasic    20   0  46560 12m  9616  S   0.0   1.2   0:00.68 trashapplet
1462 crasic    20   0  43456 11m  9152  S   0.0   1.1   0:08.44 update-notifier
1333 crasic    20   0  43536 11m  8452  S   0.0   1.1   0:05.41 notify-osd
1420 crasic    20   0  67376 9.8m  7640  S   0.0   1.0   0:00.23 evolution-alarm
```

IO

- `%util` - процент времени в течение которого диск занят вводом/выводом.
- `r/s`, `w/s` - число запросов на чтение/запись в секунду.
- `rKb/s`, `wKb/s` - объем данных прочитанных/записанных в секунду.
- `await`, `r_await`, `w_await` - среднее время в мс. ожидания+выполнения запроса ввода/вывода. latency диска.
- `avgqu-sz` - средняя длина очереди запросов к диску.

Как посмотреть:

- `iostat -x 1`
- `dstat`

Проблемы:

- `%util` ~ 100% - вам не хватает мощности диска
- `%util` сильно отличается у разных дисков в RAID - неисправен диск?

```
$ iostat
```

| Device: | r/s | w/s | rsec/s | wsec/s | avgqu-sz | await | %util |
|---------|------|------|--------|--------|----------|-------|-------|
| sda | 0.26 | 0.81 | 14.00 | 21.68 | 0.05 | 36.55 | 2.28 |
| sda2 | 1.89 | 0.26 | 0.81 | 14.00 | 0.07 | 45.85 | 1.12 |

```
[administrator@Akdemir: ~]$ iostat -N
Linux 4.1.0-2-amd64 (Akdemir) 12/01/2015 _x86_64_ (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.11    0.00    0.20    1.80    0.00   97.89

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda                1.14         29.31         1.66     57300     3245
sdb                1.06         26.84         0.70     52476     1360
md0                 0.11          1.82         0.00      3556         9
md1                 2.18         48.45         2.12    94707    4136
md2                 0.11          1.71         0.00      3340         0
vg_main-root        0.72         20.61         0.23    40297     444
vg_swap-swap        0.05          1.23         0.00      2396         0
vg_main-usr         0.72         10.19         0.21    19925     408
vg_main-home        0.06          1.41         0.03       2761         56
vg_main-var         0.58         15.74         1.65    30764    3228
vg_tmp-tmp          0.11          1.13         0.24      2213      460
```

Troubleshooting

Алгоритм:

- идентифицировать проблему
- найти причину (?)
- решить проблему

Расширенный алгоритм: управление инцидентами.

Гуглить по фразам "incident management" и "blameless postmortem".

Инструменты:

- `top`, `htop` - умеют сортировать процессы по CPU, RES
- `iostat` - умеет сортировать процессы по использованию диска
- `iftop` - трафик, по хостам
- `atop` - записывает лог, позволяет исследовать ситуацию *post hoc*



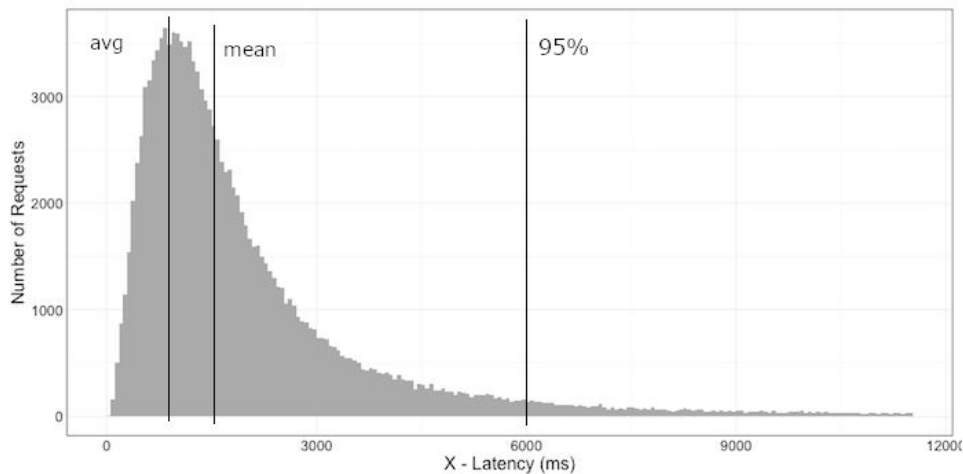
Мониторинг Web/API серверов

- RPS (request per second)
- Response time
- Задержка между компонентами приложения (latency)
- Код ответа (HTTP status 200/500/5xx/4xx)
- Разделение по методам API

Для детального анализа: трейсинг, например, <https://opentracing.io>



Распределение значений

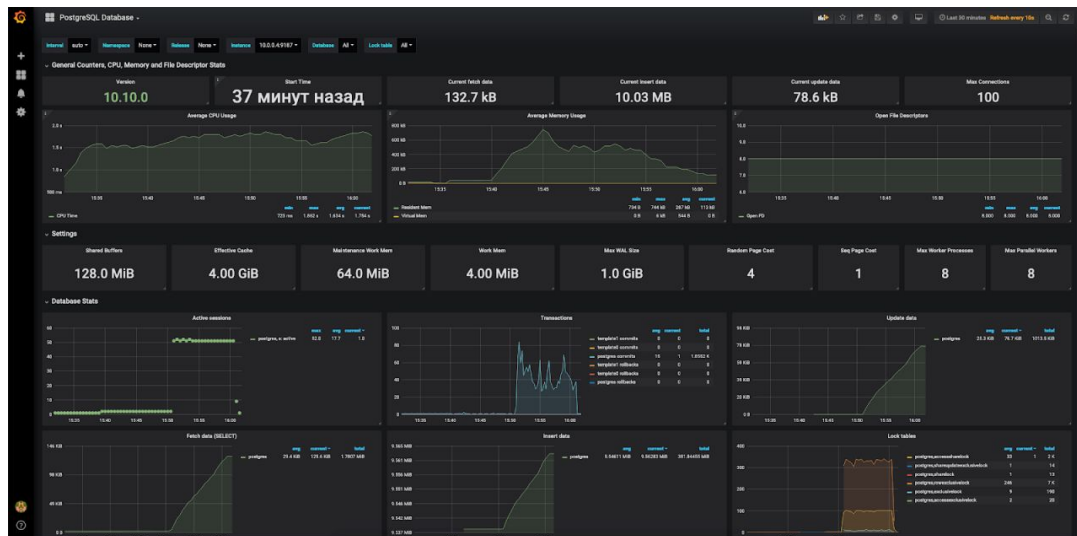


Среднее значение (`avg`, `mean`) или даже медиана (`median`) не отражают всей картины!

Полезно измерять *процентили* (percentile): время в которое укладываются 95% или, например, 99% запросов.

Мониторинг баз данных

- TPS (transactions per second)
- QPS (queries per second)
- IO usage
- CPU usage
- Replication Lag
- Wait Events
- Active connections



Основные группы метрик

- Latency - время задержки
- Traffic - количество запросов и объем трафика
- Errors - количество и характер ошибок
- Saturation - утилизация ресурсов

Вопросы



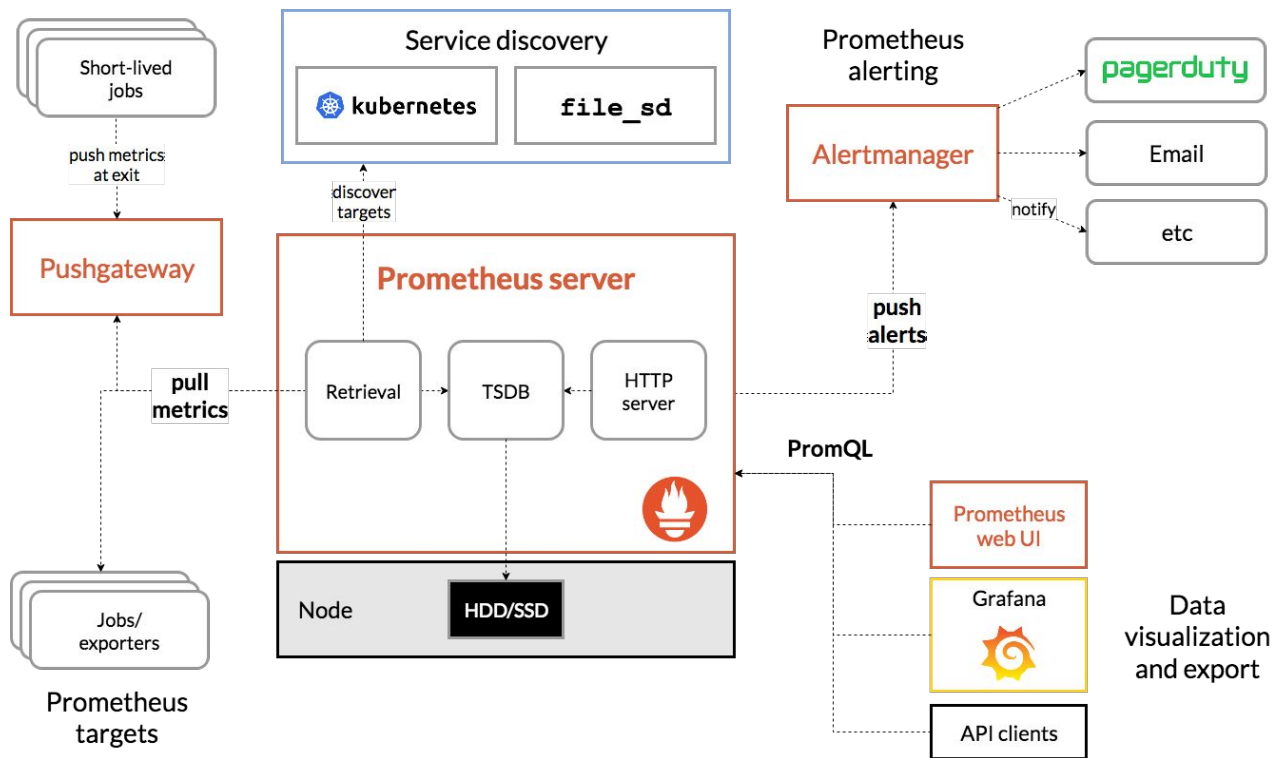
если есть вопросы



если вопросов нет

Prometheus

Prometheus



Установка и запуск сервера

```
docker run \  
  -p 9090:9090 \  
  -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml \  
  prom/prometheus
```

Настройка `/tmp/prometheus.yml`

```
global:  
  scrape_interval: 15s # как часто опрашивать exporter-ы  
  
scrape_configs:  
  - job_name: 'prometheus'  
    static_configs:  
      - targets: ['localhost:9090']  
  - job_name: 'app'  
    static_configs:  
      - targets: ['localhost:9100', 'localhost:9102', 'localhost:9103', 'localhost:9187']
```



Prometheus - запуск

С настройками по умолчанию Prometheus будет доступен на порту 9090:

<http://127.0.0.1:9090/>

Prometheus - мониторинг

- мониторинг сервера:
 - https://github.com/prometheus/collectd_exporter (collectd + collectd-exporter)
 - https://github.com/prometheus/node_exporter
- мониторинг базы: postgres-exporter
 - https://github.com/wrouesnel/postgres_exporter
- визуализация:
 - <https://grafana.com/docs/grafana/latest/installation/docker/>

Prometheus - протокол

Простой способ исследовать: `wget -O - http://localhost:9103/metrics`

```
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 1.036096e+06

collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="blocked"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="paging"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="running"} 1
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="sleeping"} 57
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="stopped"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="zombies"} 0

go_gc_duration_seconds{quantile="0"} 4.0147e-05
go_gc_duration_seconds{quantile="0.25"} 6.950600000000001e-05
go_gc_duration_seconds{quantile="0.5"} 0.000108126
go_gc_duration_seconds{quantile="0.75"} 0.001107202
go_gc_duration_seconds{quantile="1"} 0.039212351
go_gc_duration_seconds_sum 0.49406203400000004
go_gc_duration_seconds_count 282
```



Prometheus - типы метрик

- `Counter` - монотонно возрастающее число, например, число запросов
- `Gauge` - текущее значение, например, потребление памяти
- `Histogram` - распределение значений по бакетам (сколько раз значение попало в интервал)
- `Summary` - похоже на `histogram`, но по квантилям
- Векторные типы для подсчета данных по меткам

Документация: https://prometheus.io/docs/concepts/metric_types/

Отличная документация в godoc:

https://godoc.org/github.com/prometheus/client_golang/prometheus



Мониторинг Go HTTP сервисов

```
import (  
    "log"  
    "net/http"  
    "github.com/prometheus/client_golang/prometheus/promhttp"  
    metrics "github.com/slok/go-http-metrics/metrics/prometheus"  
    "github.com/slok/go-http-metrics/middleware"  
)  
  
func myHandler(w http.ResponseWriter, r *http.Request) {  
    w.WriteHeader(http.StatusOK)  
    w.Write([]byte("hello world!"))  
}  
  
func main() {  
    // middleware для мониторинг  
    mdlw := middleware.New(middleware.Config{  
        Recorder: metrics.NewRecorder(metrics.Config{}),  
    })  
    h := mdlw.Handler("", http.HandlerFunc(myHandler))  
    // HTTP exporter для prometheus  
    go http.ListenAndServe(":9102", promhttp.Handler())  
    // Ваш основной HTTP сервис  
    if err := http.ListenAndServe(":8080", h); err != nil {  
        log.Panicf("error while serving: %s", err)  
    }  
}
```



Мониторинг Go HTTP сервисов

```
import "github.com/prometheus/client_golang/prometheus"

var regCounter = prometheus.NewCounter(prometheus.CounterOpts{
    Name: "business_registration",
    Help: "Client registration event",
})

func init() {
    prometheus.MustRegister(regCounter)
}

func myHandler(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("Hello, world!"))
    regCounter.Inc()
}
```



LIVE

Вопросы



если есть вопросы



если вопросов нет

Итоги занятия

Рефлексия

Список материалов для изучения

1. [Калькулятор SLA: 99.9% аптайм / shootnick.ru](#)
2. [Installation | Prometheus](#)
3. [Monitoring Distributed Systems - sre golden signals](#)
4. [Monitoring and Observability. During lunch with a few friends in late... | by Cindy Sridharan | Medium](#)
5. [Человеческим языком про метрики 3: перцентили для чайников](#)



Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то, что узнали на вебинаре?

Приглашаем на следующий вебинар

Тестирование микросервисов



Ссылка на вебинар будет
в ЛК за 15 минут



Материалы
к занятию в ЛК —
можно изучать



Обязательный материал
обозначен красной
лентой

Заполните, пожалуйста, опрос о занятии

Мы читаем все ваши сообщения
и берем их в работу 🖋️❤️

**You are our
OTUS heroes**

