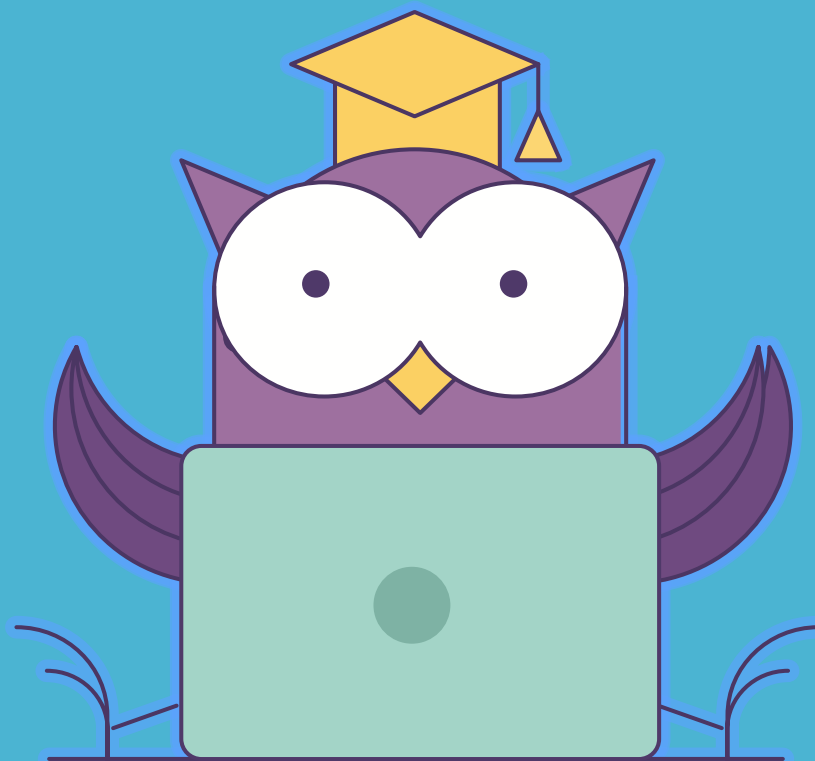




ОНЛАЙН-ОБРАЗОВАНИЕ

# Как меня слышно и видно?



## > Напишите в чат

+ если все хорошо

- если есть проблемы со звуком или с видео

!проверить запись!

# Мониторинг

Дмитрий Смаль



- Мониторинг Linux-серверов
- Системные метрики: LA, CPU, MEM, IO
- Мониторинг Web-серверов
- Мониторинг Баз-данных
- Prometheus

- Отладка, решение текущих проблем
- Рассылка уведомлений
- Технические A/B эксперименты
- Анализ трендов, прогнозирование

SRE book:

<https://landing.google.com/sre/sre-book/chapters/monitoring-distributed-systems/>

- Количественный / Событийный
- Whitebox / Blackbox
- Push / Pull

Ya ?

*Push* - агент, работающий на машине (в виртуалке/контейнере/программе) подключается к сервер мониторинга и отправляет данные. Плюсы:

- мониторинг одноразовых задач
- может работать за NAT
- может быть более производительным (за счет UDP)

Пример: `Graphite`

*Pull* - сервер мониторинга сам опрашивает агентов, собирая статистику.

Плюсы:

- проще получать данные по-запросу
- более отказоустойчивый
- не требует авторизации / верификации источника

Пример: `Prometheus`

- LA (Load Average) - длина очереди процессов в планировщике
- CPU (User/System/Wait) - время проводимое процессором в различных режимах
- Memory (RSS/Shared/Cached/Free) - распределение памяти в системе
- IO stats
- Network stats



LA - сложная метрика, ее можно интерпретировать как количество процессов(поток) 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

- 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.

Как посмотреть:

- `top`, `htop`

- Resident ( `Res` ) - память, занятая данными программ (как правило кучи). 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 - не занятая память.

Как посмотреть:

- `top`
- `free`

- `%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?

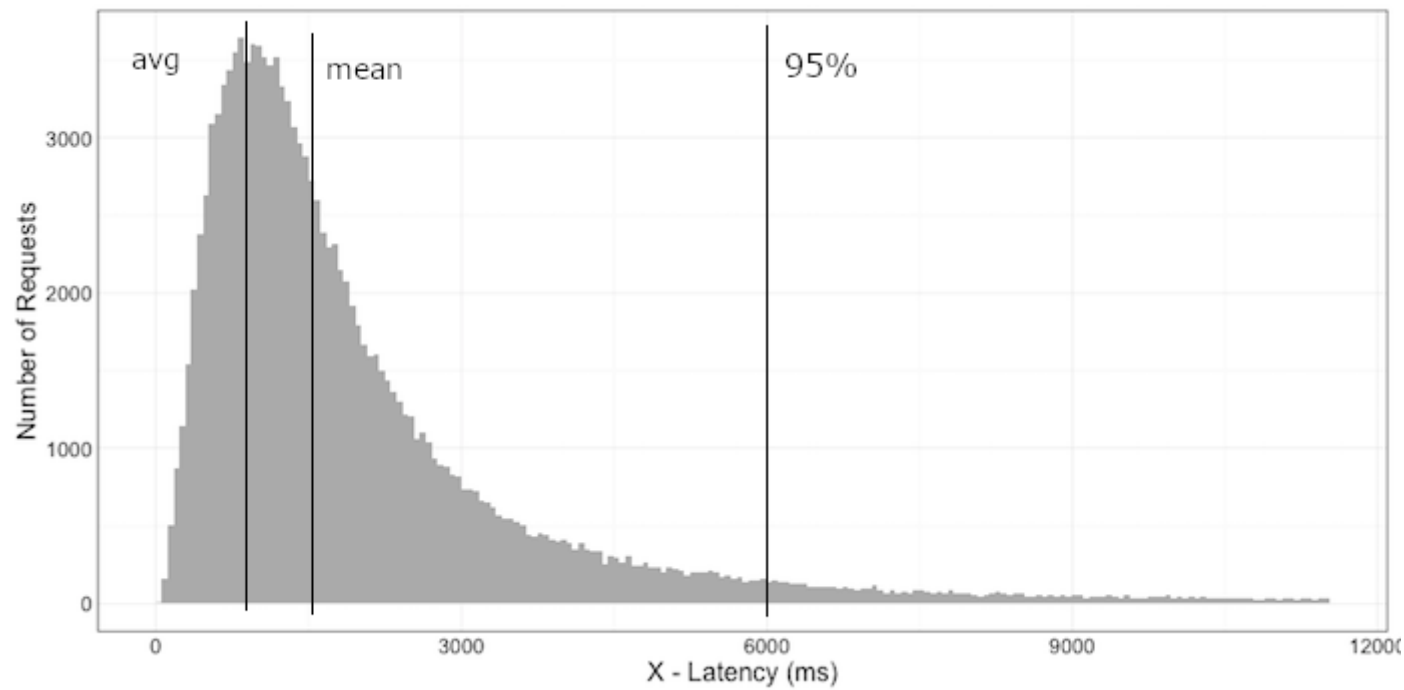
Алгоритм админа =)

- идентификация проблемы
- найти причину (?)
- решить проблему

Инструменты:

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

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

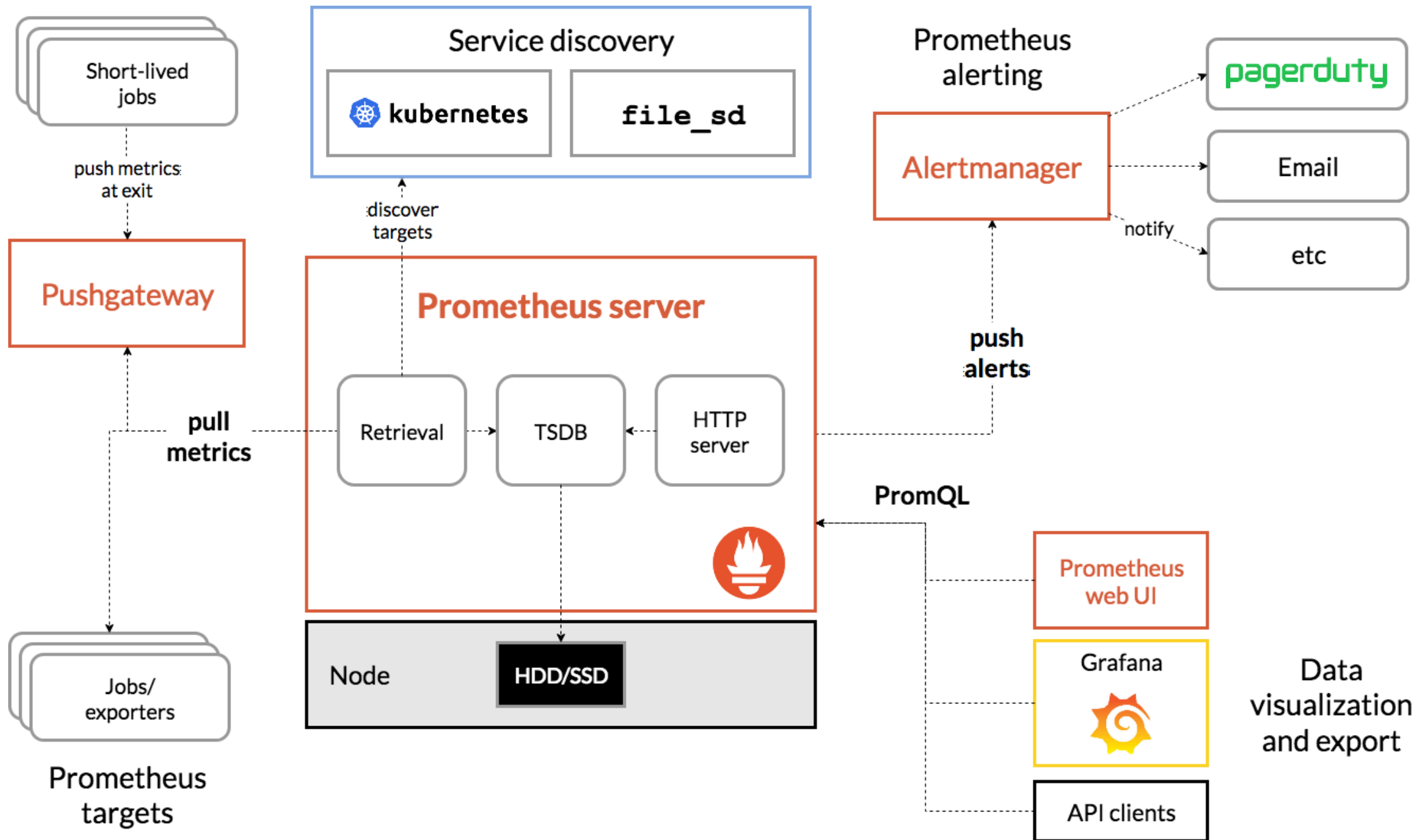


Среднее значение ( `avg` ) или даже медиана ( `mean` ) не отображают реальной картины! Необходимо измерять *перцентили*, например время в которое укладываются `98%` запросов.

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



- Latency - время ответа
- Traffic - количество запросов / объем
- Errors - количество ошибок
- Saturation - утилизация ресурсов



## Установка

```
$ apt get install prometheus
```

## Настройка `/etc/prometheus/prometheus.yml`

```
global:
  scrape_interval: 15s # как часто опрашивать exporter-ы

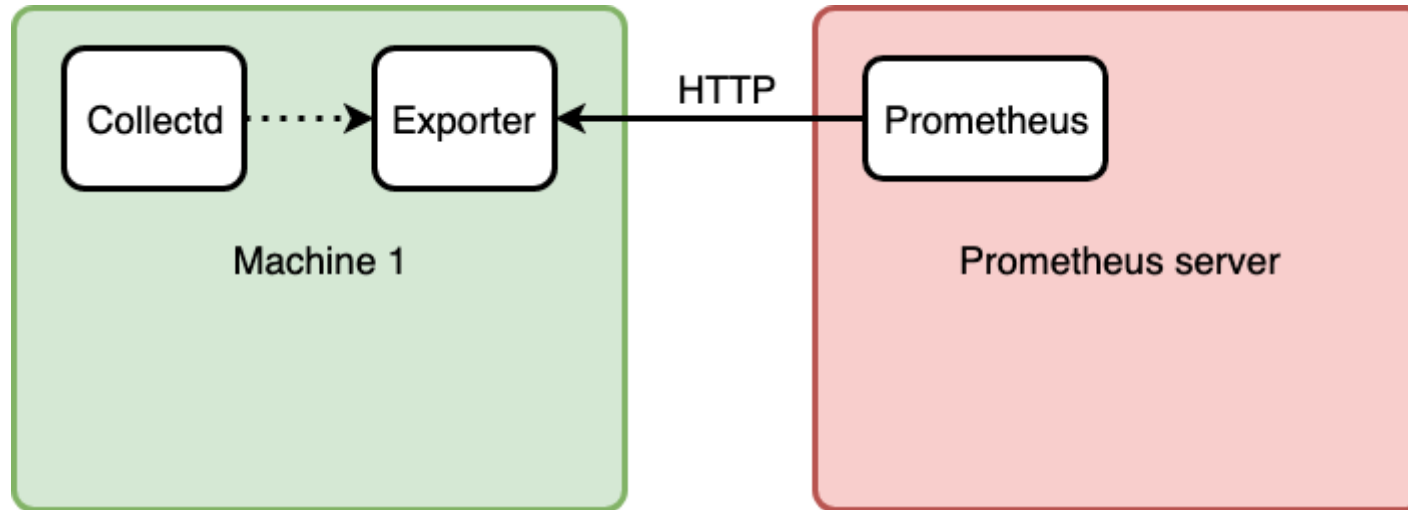
storage:
  tsdb:
    path: /var/lib/prometheus # где хранить данные
    retention:
      time: 15d # как долго хранить данные

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'app'
    static_configs:
      - targets: ['localhost:9100', 'localhost:9102', 'localhost:9103', 'localhost:9187']
```

## Запуск

```
$ service prometheus start
```

Prometheus будет доступен на порту 9090: <http://84.201.152.233:9090/>



```
$ apt-get install collectd

# В /etc/collectd/collectd.conf
LoadPlugin network
<Plugin network>
    Server "127.0.0.1" "25826"
</Plugin>

# запускаем collectd
$ service start collectd

# скачиваем collectd-exporter
$ wget https://github.com/prometheus/collectd_exporter/releases/download/v0.4.0/collectd_exporter-0.4.0
$ tar -zxvf collectd_exporter-0.4.0.linux-amd64.tar.gz

# запускаем exporter
$ ./collectd_exporter --collectd.listen-address="localhost:25826" \
    --web.listen-address="localhost:9103"
```

## Установить postgres-exporter

```
$ go get github.com/wrouesnel/postgres_exporter
$ cd ${GOPATH-$HOME/go}/src/github.com/wrouesnel/postgres_exporter
$ go run mage.go binary
```

## Запустить с указанием connection string

```
$ export DATA_SOURCE_NAME="postgresql://login:password@hostname:port/dbname"
$ ./postgres_exporter
```

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-centrall1.internal",processes="blocked"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-centrall1.internal",processes="paging"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-centrall1.internal",processes="running"} 1
collectd_processes_ps_state{instance="mialinx-test-ub.ru-centrall1.internal",processes="sleeping"} 57
collectd_processes_ps_state{instance="mialinx-test-ub.ru-centrall1.internal",processes="stopped"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-centrall1.internal",processes="zombies"} 0

go_gc_duration_seconds{quantile="0"} 4.0147e-05
go_gc_duration_seconds{quantile="0.25"} 6.9506000000000001e-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
```



- `Counter` - монотонно возрастающее число, например число запросов
- `Gauge` - текущее значение, например потребление памяти
- `Histogram` - распределение значения, по bucket
- `Summary` - похоже на `histogram`, но по квантилям

```
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)  
    }  
}
```

```
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()
}
```

Заполните пожалуйста опрос

<https://otus.ru/polls/5662/>



Спасибо за внимание!

