Урок 4. Мониторинг с Prometheus + Grafana

На этом занятии мы разберём ещё один интересный инструмент для мониторинга приложений — и не просто приложений, а целого набора систем и серверов. В связи с инструментом визуализации Grafana это даёт отличный результат. У вас может быть единый дашборд, на котором в реальном времени отображаются различные метрики производительности ваших систем и серверов.

Prometheus — это база данных временных рядов. Такие базы специально разработаны для обработки данных, связанных со временем.

В реляционных базах есть таблицы. Эти таблицы содержат столбцы и строки, каждая из которых определяет запись в вашей таблице. Базы данных временных рядов работают иначе. Данные хранятся в «коллекциях», но эти коллекции имеют общий знаменатель: они агрегируются с течением времени. Это означает, что для каждой точки, которую вы сохраните, у вас есть временная метка. Например, если мы хотим хранить время загрузки стартовой страницы, то коллекция будет содержать два атрибута — временную метку и время загрузки.

С учётом того, что Prometheus может мониторить различные системы, будь то веб-приложения, базы данных, сервера или виртуальные машины, мы сможем измерять производительность всей инфраструктуры проекта.

Prometheus извлекает метрики через HTTP-вызовы к определённым конечным точкам, указанным в его конфигурации.

Например, настроим наше flask-приложение так, чтобы оно передавало метрики на некоторый URL. По этому URL Prometheus с определёнными интервалами и будет извлекать данные. Проще говоря, Prometheus встраивает роут /metrics в приложение Flask, определяет триггеры before_request, after_request, цепляет статистику по запросам, а при обращении на /metrics эта статистика отдаётся.

Что касается баз данных и серверов, у Prometheus есть готовые экспортеры, которые будут производить сбор метрик. В любом случае Prometheus сам инициализирует сбор данных. Но сейчас мы поговорим о Flask.

Пример кода взят из репозитория библиотеки.

```
import time
import random

from flask import Flask
from prometheus_flask_exporter import PrometheusMetrics
```

```
app = Flask(__name__)
metrics = PrometheusMetrics(app)


@app.route('/one')
def first_route():
    time.sleep(random.random() * 0.2)
    return 'ok'


@app.route('/two')
def the_second():
    time.sleep(random.random() * 0.4)
    return 'ok'


@app.route('/three')
def test_3rd():
    time.sleep(random.random() * 0.6)
    return 'ok'


@app.route('/four')
def fourth_one():
    time.sleep(random.random() * 0.8)
    return 'ok'


@app.route('/error')
def oops():
    return ':(', 500


if __name__ == '__main__':
    app.run('0.0.0.0', 5000, threaded=True)
```

Перед запуском Prometheus давайте настроим его. Для этого нужно создать yml-файл и описать конфиги.

```
global:
  scrape_interval:     3s

  external_labels:
    monitor: 'flask-app'
```

```
scrape_configs:
  - job_name: 'prometheus'

    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'flask'

    dns_sd_configs:
      - names: ['app']
        port: 5000
        type: A
        refresh_interval: 5s
```

Полный список переменных для конфигурации можно посмотреть в документации: https://prometheus.io/docs/prometheus/latest/configuration/configuration/.

Далее поговорим о Grafana — это платформа для создания информационных панелей, аналитики и мониторинга, которая настраивается для подключения к различным источникам, таким как Prometheus. Основные плюсы: возможность работать на одной панели мониторинга с несколькими источниками данных, приятный дизайн и простое подключение к источникам. Однако есть и минусы — не самая очевидная настройка визуализации панели.

Сперва давайте поднимем докер — контейнеры, в которых мы опишем наше flask-приложение, Prometheus, Grafana и небольшой генератор, который будет спамить запросы на наш app.

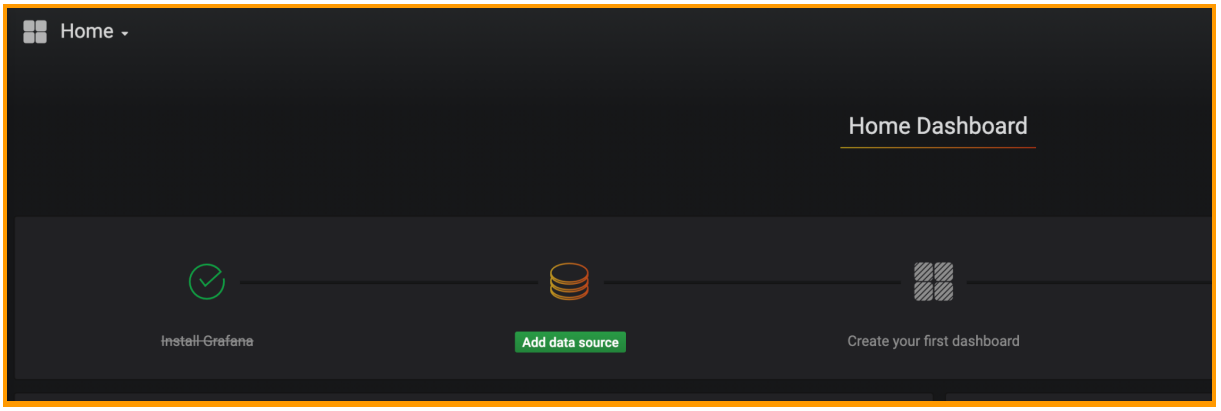
```
import time
import random
import threading

import requests

endpoints = ('one', 'two', 'three', 'four', 'error')


def run():
   while True:
       try:
           target = random.choice(endpoints)
           requests.get("http://app:5000/%s" % target, timeout=1)

       except:
```

```
            pass


if __name__ == '__main__':
    for _ in range(4):
        thread = threading.Thread(target=run)
        thread.daemon = True
        thread.start()

    while True:
        time.sleep(1)
```

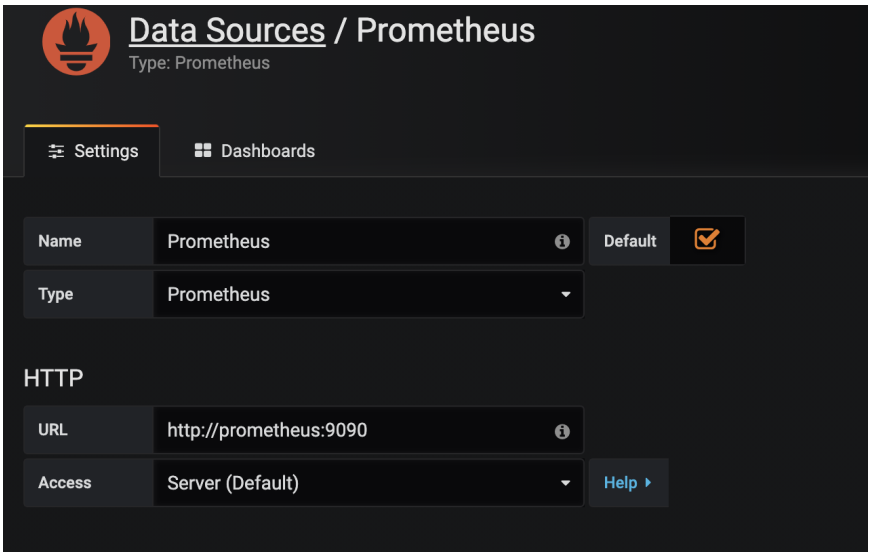
Далее описываем конфиг докера.

```
version: '2'
services:

 app:
   build:
     context: app
   stop_signal: SIGKILL
   ports:
     - 5000:5000

 generator:
   build:
     context: generator
   stop_signal: SIGKILL

 prometheus:
   image: prom/prometheus:v2.2.1
   volumes:
     - ./prometheus/config.yml:/etc/prometheus/prometheus.yml
   ports:
     - 9090:9090

 grafana:
   image: grafana/grafana:5.1.0
   ports:
     - 3000:3000
```

Заходим на http://localhost:3000, видим страницу входа в Grafana (по умолчанию логин и пароль — admin). Попадаем на стартовую страницу.



Добавляем источник — Prometheus.



Далее импортируем пример example.json для построение дашборда.

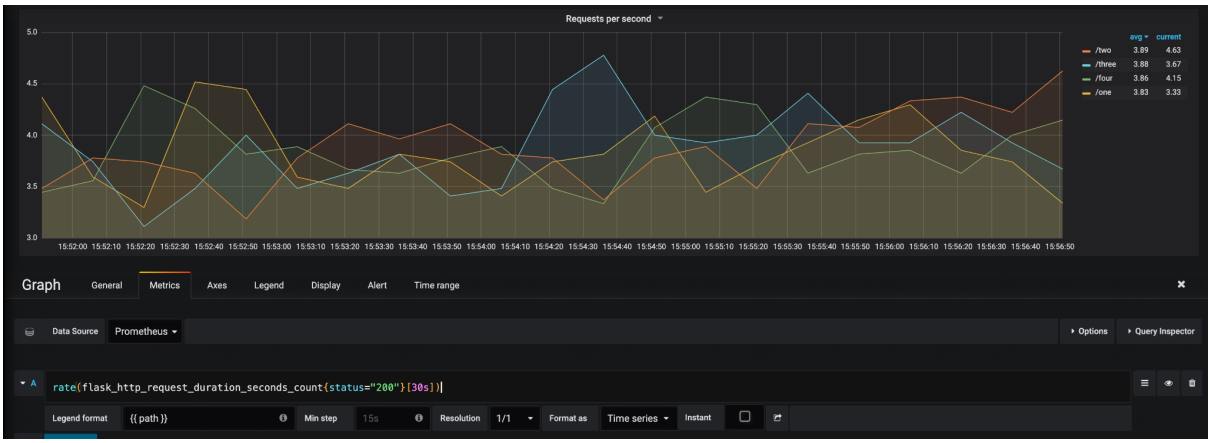
Конфиги для Grafana так же можно было определить в проект в yaml-файлах и передать их в конфиг докера. В таком случае при поднятии контейнера мы бы сразу

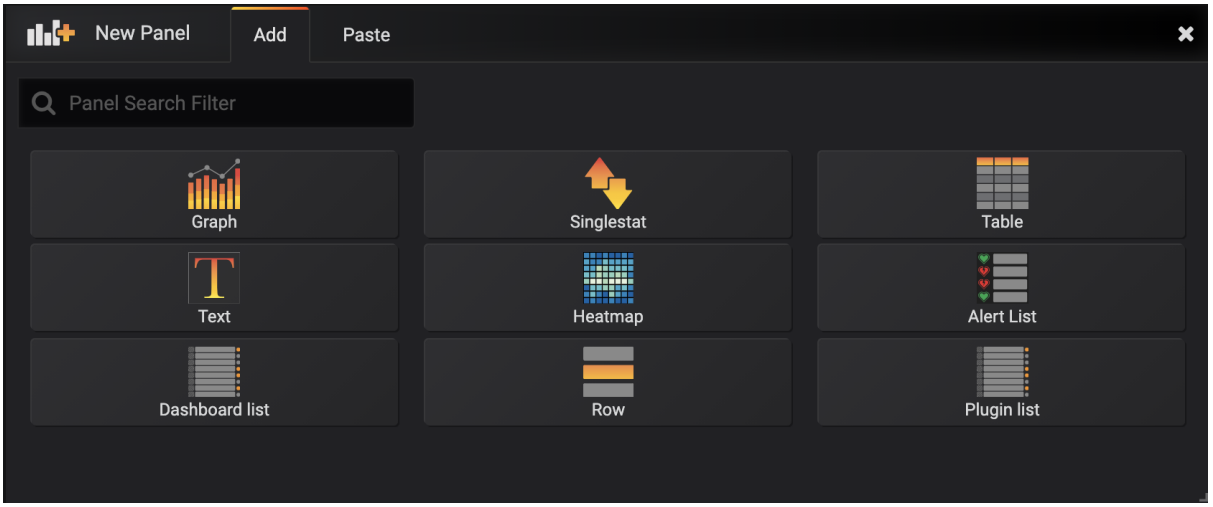
увидели наш дашборд. Но сейчас мы не будем усложнять работу с этими инструментами. Ссылки, как писать конфиги для Grafana, будут под видео.

Конфиги Grafana: https://grafana.com/docs/grafana/latest/administration/configuration/.

Конфиг Grafana + Prometheus: https://grafana.com/docs/grafana/latest/datasources/prometheus/.

Аналогично настройке источника мы можем в интерфейсе Grafana настраивать графики различного типа.





Давайте посмотрим на магический json-файл, который построил нам такой дашборд.

Наш дашборд состоит из панелей.

Итак, подведём итог модуля. Мы разобрались, какие типы профилировщиков существуют, познакомились с инструментами профилирования, узнали, как мониторить вызовы с помощью Pycharm, библиотеки flask_profiler и werkzeug profiler. Мы также поработали с системой мониторинга ошибок Sentry и со связкой инструментов Prometheus + Grafana для замеров полезных метрик с их последующей визуализацией. Помимо инструментов, которые мы использовали в этом модуле, есть большое количество альтернатив, поэтому экспериментируйте.