Занятие 10: Мониторинг метрик

0. Prometheus и Node-extractor

На сайте с документацией описаны все варианты установки.
Так как связка этого ПО уже входит в официальные репозитории Ubuntu, можно установить её стандартным менеджером пакетов, хоть это и будут более старые версии, по сравнению со скачиванием с официального сайта.
Здесь и далее используется виртуалка с Ubuntu 20.04 с именем ***prometheus-machine***.

Установка ПО
```
sudo -i
apt update
apt install prometheus
```

Проверка установки. Оба компонента должны быть активны.
```
sudo systemctl status prometheus
sudo systemctl status prometheus-node-exporter
```

Открываем портал Prometheus, порт по-умолчанию: 9090: *http://prometheus-machine:9090/targets*

Prometheus сам собирает и публикует свои метрики: *http://prometheus-machine:9090/metrics*.
В дополнение к этому, установленный Node-exporter публикует снятые с узла метрики на порту 9100: *http://prometheus-machine:9100/metrics*

Node-exporter

Если вдруг вам понадобится Node-exporter отдельно, его можно взять с Гитхаба, либо взять новую версию с официального сайта: *https://prometheus.io/docs/guides/node-exporter/#tarball-installation*

Скачать и распаковать дистрибутив
```
wget https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-1.3.1.linux-amd64.tar.gz
tar xvfz node_exporter-1.3.1.linux-amd64.tar.gz
cd node_exporter-1.3.1.linux-amd64/
```

Скопировать бинарник в /usr/sbin
```
sudo cp node_exporter /usr/sbin/
```

Создать соответствующий сервис
```
sudo nano /etc/systemd/system/node_exporter.service
```

[Unit]
Description=Node Exporter

[Service]
ExecStart=/usr/sbin/node_exporter
Restart=Always

[Install]
WantedBy=default.target
=======

Запустить сервис и добавить в автозапуск
```
sudo systemctl enable node_exporter.service
sudo systemctl start node_exporter
```

1. Конфигурация Prometheus

Настроек Prometheus по умолчанию вполне достаточно, чтобы следить за всем происходящим на локальной машине.

```
nano /etc/prometheus/prometheus.yml
-------
    global:
      scrape_interval: 15s       — интервал сбора метрик (по умолчанию — 15 секунд);
      evaluation_interval: 15s   — интервал сверки с правилами (по умолчанию — 15 секунд);
      rule_files:                — файлы правил (речь о них пойдёт ниже)

    scrape_configs:              — базовые настройки сбора метрик на сервере
      - job_name: "prometheus"   — имя задачи;
      - scrape_interval: "15s"   — интервал сбора метрик (здесь — каждые 15 секунд);

    target_groups:               — список сервисов, для которых нужно собирать метрики.
        - targets:
           - ["localhost:9090", "localhost:9100", ...]
=======
```

После изменения конфигурации нужно перезагрузить *Prometheus*:
```
sudo systemctl restart prometheus
```

Кроме того, существует альтернативный, более гибкий способ добавления источников метрик динамически механизмами service discovery. Применяйте любой, но лучше их не комбинировать.

Также в файле конфигурации прописываются ссылки на файлы с описанием правил. Правила помогают предварительно вычислять часто используемые или требующие значительных затрат ресурсов показатели и сохранять их в виде новых временных рядов.
Работа с правилами описана в *https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/*

В общем виде синтаксис правил можно представить так:

<имя временного ряда>{метки} = ***<параметр для записи>***

Примеры:
```
    job:http_inprogress_requests:sum = sum(http_inprogress_requests) by (job)
    new_time_series{label_to_change="new_value",label_to_drop=""} = old_time_series
```

В виде файла правил можно организовать их в группы:
```
groups:
  - name: example
    rules:
    - record: code:prometheus_http_requests_total:sum
      expr: sum by (code) (prometheus_http_requests_total)
```

Prometheus вычисляет с правила с определённой периодичностью, указанной в конфигурационном файле в параметре evaluation_interval, пересчитывает значение и сохраняет его под новым именем с текущей временной меткой.

3. Подключение новых источников и метрик. Экспортер для питона.

Описание: *https://github.com/prometheus/client_python*

Ставим его на виртуальную машину, где есть Python и pip.

@pyt

```
pip install prometheus-client
```

Создаем демо-проект

```
nano prom-test.py
-------
from prometheus_client import start_http_server, Summary
import random
import time

# Create a metric to track time spent and requests made.
REQUEST_TIME = Summary('request_processing_seconds', 'Time spent processing request')

# Decorate function with metric.
@REQUEST_TIME.time()
def process_request(t):
    """A dummy function that takes some time."""
    time.sleep(t)

if __name__ == '__main__':
    # Start up the server to expose the metrics.
    start_http_server(8000)
    # Generate some requests.
    while True:
        process_request(random.random())
=======
```

Запускаем процесс в фоновом режиме

```
python prom-test.py &
```

* Открываем страницу браузера к виртуалке с питоном, чтобы убедиться в том, что метрики доступны: *http://192.168.3.97:8000/*

Подключение к Prometheus:

@prometheus-machine

```
nano /etc/prometheus/prometheus.yml
-------
...
  - job_name: test_py
    static_configs:
      - targets: ['192.168.3.97:8000']
=======
```

После изменения конфига Prometheus нужно перезагрузить:

```
sudo systemctl restart prometheus
```

* Результат: на http://prometheus-machine:9090/targets появился новый источник данных.

Посмотрите список доступных метрик на странице http://prometheus-machine:9090/graph, выберите метрику *request_processing_seconds_count* и нажмите кнопку *Execute*.

В текстовом поле *Console* появилось выражение:

```
request_processing_seconds_count{instance="192.168.3.97:8000",job="test_py"}
```

Также на странице появился график данной метрики.

Добавим новую метрику, вычислим количество вызовов в секунду (*rate*):
*Add Graph -> **rate(request_processing_seconds_count[1m])** -> Execute*

Эту же метрику можно добавить в виде правила в конфигурацию Prometheus.

@prometheus-machine

```
nano /etc/prometheus/prometheus.yml
-------
... rule-files:
    - test-py.rules
...
=======
```

Теперь сами правила:

```
sudo nano /etc/prometheus/test-py.rules
-------
groups:
  - name: requests_from_python
    rules:
    - record: request_processing_seconds_count:rate
      expr: rate(request_processing_seconds_count[1m])
      labels:
        class: May24
=======
```

Теперь в списке доступных метрик на http://prometheus-machine:9090/graph доступен выбор вычисленной метрики:
***request_processing_seconds_count:rate** -> Execute*

4. AlertManager

Это программа из того же пакета что и Prometheus, она позволяет сортировать алерты и отправлять оповещения реже, чем просто «каждый раз при срабатывании».

Инструкций по настройке оповещений много в интернете:

https://losst.ru/nastrojka-alertmanager-prometheus
https://itnext.io/prometheus-with-alertmanager-f2a1f7efabd6
https://blog.ruanbekker.com/blog/2019/05/17/install-alertmanager-to-alert-based-on-metrics-from-prometheus/

Настройка алертинг-правил Prometheus

Первым делом необходимо настроить правила алертинга, чтобы программа генерировала алерты при возникновении тех или иных событий (документация).
Для описания правил надо создать отдельный файл в папке с конфигурацией prometheus, например:

```
sudo nano /etc/prometheus/alerting.rules
-------
groups:
  - name: имя_группы
    rules:
...
=======
```

Все правила разбиты на группы. У каждой группы есть имя (name) и список правил (rules).
Давайте рассмотрим простое правило «отправлять алерт, если хост недоступен»:

```
-------
groups:
  - name: class13
    rules:
    - alert: PrometheusTargetMissing
      expr: up == 0
      for: 0m
      labels:
        severity: critical
      annotations:
        summary: "Service node_exporter unavailable on the instance {{ $labels.instance }}"
        description: "Possibly node_exporter is down\n VALUE = {{ $value }}\n LABELS = {{ $labels }}"
=======
```

В описании можно использовать переменные.
Вот они:

{{$value}} - содержит значение переменной, участвующей в выражении;
{{$labels.instance}} - IP адрес и порт экспортёра, с которым возникла проблема;
{{$labels.job}} - имя задачи из конфигурационного файла prometheus;

Описание механизма шаблонов здесь: *https://prometheus.io/docs/visualization/consoles/*
Больше полезных правил вы можете найти здесь: *https://awesome-prometheus-alerts.grep.to/rules.html*

Осталось добавить созданный файл с правилами в основной файл конфигурации Prometheus.
Для этого добавим в него такие строчки:

```
sudo nano /etc/prometheus/prometheus.yml
-------
...
rule_files:
  - "alerting.rules"
...
=======
```

Затем нужно проверить созданные правила, для этого выполните команду:

```
promtool check rules /etc/prometheus/alerting.rules
```

Если всё прошло успешно, команда выведет слово SUCCESS и количество найденных правил.

После изменения конфига Prometheus нужно перезагрузить:
```
sudo systemctl restart prometheus
```

* Результат: На странице http://prometheus-machine:9090/alerts появилась запись об алерте.

Установка AlertManager

Установить *Alertmanager* можно из официальных репозиториев, но так вы получите программу старой версии и без веб-интерфейса.
```
sudo apt install prometheus-alertmanager -y
```

После завершения установки веб-интерфейс можно развернуть командой:
```
/usr/share/prometheus/alertmanager/generate-ui.sh
```

Также можно установить программу с официального сайта или скачать с гитхаба. Как и другие программы этой экосистемы, *Alertmanager* написан на Golang, поэтому состоит из одного исполняемого файла и нескольких конфигурационных файлов. Он не зависит от установленных в системе библиотек.
Сначала скачайте архив с исполняемыми файлами alertmanager с Гитхаба, например:
```
wget \
https://github.com/prometheus/alertmanager/releases/download/v0.21.0/alertmanager
-0.21.0.linux-amd64.tar.gz
```

#Дальше распакуйте полученный архив:
```
tar -xvf alertmanager-0.21.0.linux-amd64.tar.gz
```

И скопируйте файлы *alertmanager* и *amtool* в папку /usr/local/bin:
```
sudo cp alertmanager-0.21.0.linux-amd64/alertmanager /usr/local/bin/
sudo cp alertmanager-0.21.0.linux-amd64/amtool /usr/local/bin/
```

Далее надо скопировать конфигурационный файл *alertmanager.yml* в */etc/prometheus*:
```
sudo cp alertmanager-0.21.0.linux-amd64/alertmanager.yml /etc/prometheus
```

Дайте пользователю *prometheus* права на конфигурационный файл:
```
sudo chown -R prometheus:prometheus /etc/prometheus/alertmanager.yml
```

Осталось создать *systemd* сервис, с помощью которого вы сможете запускать программу.
```
sudo systemctl edit --full --force prometheus-alertmanager
-------
[Unit]
Description=Alertmanager Service
After=network.target

[Service]
EnvironmentFile=-/etc/default/alertmanager
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/alertmanager $ARGS
ExecReload=/bin/kill -HUP $MAINPID
Restart=on-failure

[Install]
WantedBy=multi-user.target
=======
```

Теперь можно запустить сервис:
```
sudo systemctl start prometheus-alertmanager
```

* Результат: на странице http://prometheus-machine:9093/ доступен Alertmanager UI

Настройка отправки алертов в Prometheus

```
sudo nano /etc/prometheus/alertmanager.yml
-------
alerting:
  alertmanagers:
  - static_configs:
    - targets: ['localhost:9093']
=======
```

```
sudo systemctl restart prometheus
```

Настройка оповещений в AlertManager

Надо указать как группировать алерты, когда и куда отправлять уведомления.

Вот основное содержание конфигурационного файла:

```
sudo nano /etc/prometheus/alertmanager.yml
-------
global:
route:
  group_by: ['по каким параметрам группировать правила']
  group_wait: время_ожидания_перед_отправкой_уведомления_для_группы
  group_interval: время_отправки_повторного_сообщения_для_группы
  repeat_interval: время_до_отправки_повторного_сообщения
  receiver: 'имя_способа_отправки_сообщений'
receivers:
  - name: 'имя_способа_отправки_сообщений'
    конфигурация
=======
```

Пример конфигурации с отправкой уведомлений на email:

```
sudo nano /etc/prometheus/alertmanager.yml
-------
global:
    route:
      group_by: ['alertname']
      group_wait: 30s
      group_interval: 10m
      repeat_interval: 60m
      receiver: 'email'
    receivers:
    - name: 'email'
      email_configs:
      - to: 'адрес_электронной_почты_получателя'
        from: 'адрес_электронной_почты_отправителя'
        smarthost: 'SMTP_хост:порт'
        auth_username: 'имя_пользователя'
        auth_identity: 'имя_пользователя'
        auth_password: 'пароль'
=======
```

5. Тестовый алерт и его проба

Осталось только перезапустить Alertmanager:

```
sudo systemctl restart prometheus-alertmanager
```

Для тестирования алерта на отслеживаемом сервере можно отключить node_exporter:

```
sudo systemctl stop prometheus-node-exporter
```

Подождем пока Prometheus снова опросит цели и в AlertManager появится этот алерт:
http://prometheus-machine:9093/

Добавим выдуманный алерт превышения порога «количества запросов в минуту»:

```
sudo nano /etc/prometheus/alerting.rules
-------
groups:
```

```
  - name: class13
    rules:
    - alert: PythonTestOverload
      expr: job:request_processing_seconds_count:rate <= 2
      for: 0m
      labels:
        severity: major
      annotations:
        summary: "Service {{ $labels.job }} overload at instance {{ $labels.instance }}"
        description: "Rate of service {{ $labels.job }} exceeds threshold"
=======
```

Перезапуск

```
sudo systemctl restart prometheus
sudo systemctl restart prometheus-alertmanager
```

Результат можно проверить на страницах *http://prometheus-machine:9090/alerts* и *http://prometheus-machine:9093/*

6. Grafana

Установка Grafana несколько сложнее, чем Prometheus и Node Exporter, но, к счастью, на сайте Grafana есть отличная документация: *https://grafana.com/grafana/download?platform=linux*

Под Ubuntu - скачайте deb-пакет и установите его:

```
sudo apt-get install -y adduser libfontconfig1
wget https://dl.grafana.com/oss/release/grafana_6.4.3_amd64.deb
sudo dpkg -i grafana_6.4.3_amd64.deb
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

А ещё можно скачать с гитхаба: *https://github.com/grafana/grafana*

После установки станет доступен UI, порт по-умолчанию: 3000, пользователь и пароль: admin/admin.
http://prometheus-machine:3000/

Подключение к Prometheus

@Grafana UI
С домашнего дашборда:
!Add data source -> Prometheus:
Name: ***Prometheus-python***
URL: ***http://localhost:9090***
!Save&Test

* Результат: сообщение “*Data source is working*”

Пробный дашборд для node-extractor

Для примера используем готовый дашборд, созданный пользователем *cordobatyc*, расположенный по следующей ссылке: *https://grafana.com/grafana/dashboards/10795*

@Grafana UI
!Дашборды -> !Manage -> !Import
Grafana.com dashboard: ***10795***
!Load
Prometheus: ***Prometheus-python***

!Import

После установки здесь не хватает одного компонента (по центру дашборда). Необходимый для этого плагин для отображения круговых диаграмм можно найти по ссылке:
https://grafana.com/grafana/plugins/grafana-piechart-panel

Установка плагина описана на его странице:
```
sudo grafana-cli plugins install grafana-piechart-panel
```

Для завершения установки плагина перезапустите Grafana:
```
sudo systemctl restart grafana-server
```

7. Тестовый дашборд для питона

Пример взят из https://rtfm.co.ua/grafana-sozdanie-dashboard

@Grafana UI
Добавить (+) -> Dashboard -> Add Query
Settings -> General:
Name: ***PythonTest Dashboard***
Tags: ***class8***

Допустим, у нас есть несколько сред с разными версиями питона. Соответственно – в дашборде хочется выводить статистику и того, и другого. Для этого добавим переменную, с помощью которой сможем переключаться между ними (необходимая документация).

@Grafana UI
Settings -> Variables -> Add Variable:
Name: ***python_version_major***
Type: ***Query***
Label – имя, как оно будет отображаться в дашборде для выбора
Data source: ***Prometheus***
Refresh: ***On Dashboard Load*** *(т.е. при загрузке дашборда)*
Query: это, собственно, сам запрос, который вернёт нам значения, и из которого будем получать список окружений, фильтруем вывод по метке job="test_py". Запрос получается таким:
label_values(python_info{job="test_py"}, major)
!Add

Panel -> Edit
General:
Metric: ***request_processing_seconds_count:rate***
Visualization -> ***SingleStat***
Show: ***Current***
Coloring: ***Value***
Thresholds: ***1,2***
Spark Lines: ***Show***

Сохранить.