GeekBrains

Мониторинг распределённых приложений и сервисов

Система мониторинга на основе Prometheus

На этом уроке

Мы узнаем, как установить, сконфигурировать и начать работать с Prometheus, а также познакомимся с инструментом визуализации — Grafana.

Оглавление

Prometheus

- Архитектура
- Конфигурация Prometheus
  - Service Discovery
  - Config Reload
- Metrics
- PromQL
- Exporters
- Alerting
- Security

Grafana

- Запуск и базовая настройка
- Интерфейс
- Дашборды и панели
- Группы и доступы

Практическое задание

Глоссарий

Дополнительные материалы

Используемые источники

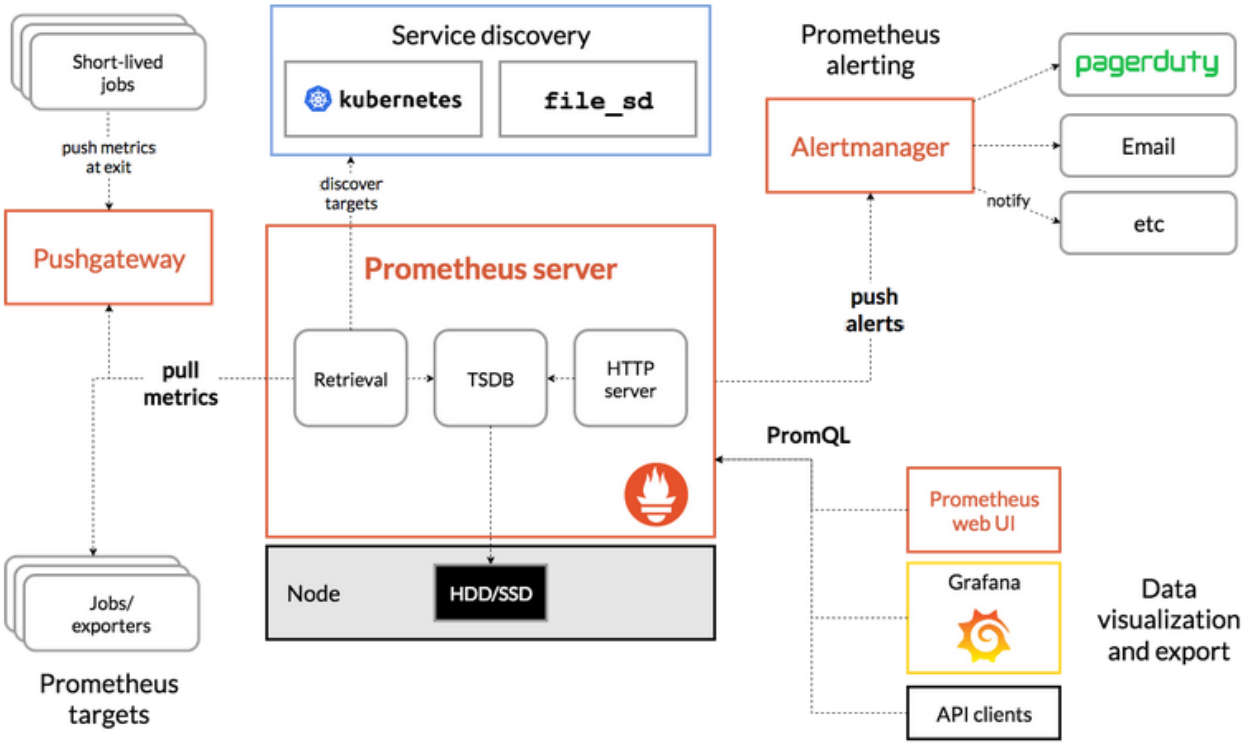
Prometheus

Архитектура

Prometheus — open-source-проект, написанный на Go, для мониторинга различных систем и сервисов. Отличительная особенность — наличие собственной TSDB (Time Series Database), собственного языка запросов PromQL и использование pull-модели для сбора метрик, т. е., Prometheus сам ходит по источникам данных. Преимущество такой модели, что всё настраивается в одном месте,

непосредственно на Prometheus-сервере. Архитектурно мониторинговое решение на основе Prometheus выглядит следующим образом:

1. Центральную часть занимает непосредственно Prometheus.
2. В его конфигурации прописаны jobs, т. е., цели до источников данных (exporters).
3. Для визуализации данных в основном используется Grafana.
4. В качестве инструмента для алертинга поставляется Alertmanager.



Для запуска Prometheus в docker воспользуйтесь командой:

```
docker run -d -p 9090:9090 prom/prometheus:v2.25.0
```

Для прокидывания своей конфигурации в контейнер, необходимо добавить в команду запуска следующие параметры, предварительно заменив /path/to на реальный путь до файла:

```
docker run -d -p 9090:9090 -v
/path/to/prometheus.yml:/etc/prometheus/prometheus.yml -v
/path/to/prometheus.rules.yml:/etc/prometheus/prometheus.rules.yml -v
/path/to/alert.rules.yml:/etc/prometheus/alert.rules.yml prom/prometheus:v2.25.0
```

Конфигурация Prometheus

Основная конфигурация представляет из себя yaml-файл, где в нескольких секциях прописываются все необходимые параметры. Вот пример базовой конфигурации, с комментариями к параметрам:

```
# prometheus global config
global:
  scrape_interval:     15s    # как часто очищать цели
  evaluation_interval: 15s    # смотрим, обновились ли правила
  # scrape_timeout is set to the global default (10s).

alerting:                     # настройки для связанного Alertmanager
  alertmanagers:
  - static_configs:
    - targets:
      - alertmanager:9093

rule_files:                   # список файлов с правилами и алертами
  - "prometheus.rules.yml"
  - "alert.rules.yml"

scrape_configs:               # список целей с конфигурацией
  # metrics_path defaults to '/metrics'
  # scheme defaults to 'http'.
  - job_name: prometheus
    scrape_interval: 5s
    scrape_timeout: 2s
    honor_labels: true
    static_configs:
      - targets: ['prometheus:9090']

  - job_name: node-exporter
    scrape_interval: 5s
    scrape_timeout: 2s
    honor_labels: true
    static_configs:
      - targets: ['node:9100']
```

Prometheus поддерживает два типа правил: Recording rules и Alerting rules. Recording rules позволяют на основе полученных метрик проводить вычисления и записывать вычисленное значение как новую метрику, что позволяет экономить ресурсы за счёт того, что не приходится вычислять значение каждый раз при запросе.

Пример Recording rule, которое описывает процент занятой памяти:

```
groups:
 - name: recordingRules-1
   rules:
     - record: node_memory_MemUsage_percent
```

```
                    expr:   100   -   (100   *   node_memory_MemFree_bytes   /
node_memory_MemTotal_bytes)
```

Alerting rules — это аналог триггеров в Zabbix, они срабатывают при наступлении заданных условий, после чего создается alert, который обрабатывается Alertmanager. В дополнение к условиям срабатывания можно указать labels и annotations. Пример Alerting rule, срабатывающий при получении значения 0 для метрики up, с фильтром по джобе `{job="node_exporter"},` в течение трёх минут:

```
groups:
- name: Hardware alerts
  rules:
  - alert: Node down
    expr: up{job="node_exporter"} == 0
    for: 3m
    labels:
      severity: warning
    annotations:
      title: Node {{ $labels.instance }} is down
       description: Failed to scrape {{ $labels.job }} on {{ $labels.instance }}
for more than 3 minutes. Node seems down.
```

Также для Prometheus, помимо локального, можно настроить взаимодействие с удалённым хранилищем. Это делается также в конфигурационном файле добавлением секций remote_write, remote_read. В качестве такой хранилки может выступать как другой Prometheus (в терминологии самого Prometheus это называется Federation), так и сторонние базы данных, такие как Victoria Metrics и Thanos.

Внимание! *Несмотря на то, что Prometheus предоставляет встроенную TSDB, для больших проектов рекомендуется использовать стороннюю TSDB.*

Prometheus в k8s

Для разворачивания Prometheus в Kubernetes используются prometheus operator, с помощью которого деплоится и управляется непосредственно сам Prometheus и необходимые компоненты.

Обычно для мониторинга kubernetes используются следующие сервисы:

- **metrics-server** — метрики использования ресурсов;
- **cAdvisor** — метрики Docker-демона, мониторинг контейнеров;
- **kube-state-metrics** — состояние k8s в целом;
- **Node-exporter** — метрики непосредственно с нод в k8s.

Service Discovery

Service Discovery — это автоматическое обнаружение целей, удобный способ конфигурирования объектов для Prometheus. В этом случае подразумевается, что вы не вручную задаёте каждый инстанс (объект мониторинга) который желаете замониторить, а указываете своему серверу, откуда он может получить эти объекты (Consul, OpenStack, local files…). Пример для консула, запущенного локально:

```
scrape_configs:
- job_name: 'overwritten-default'

  consul_sd_configs:
  - server:   '127.0.0.1:5361'
    services: ['auth', 'api', 'load-balancer', 'postgres']
```

Config Reload

Prometheus может перечитать свою конфигурацию в процессе работы, и если новая конфигурация составлена неправильно, она не будет применена. Перечитывание конфигурации запускается отправкой SIGHUP процессу Prometheus или отправкой HTTP-запроса по адресу /-/reload (работает, только если Prometheus запущен с параметром --web.enable-lifecycle).

Для перечитывания конфигурации при работе в Kubernetes используется prometheus-config-reloader.

Metrics

Prometheus, для клиентских библиотек, предлагает четыре основных типа метрики. Сам Prometheus server сейчас объединяет все данные в нетипизированные временные ряды. Рассмотрим эти типы:

1. **Counter (счётчик)** — метрика, которая может только увеличиваться или сбрасываться до нуля (например, при перезагрузке). Примеры счётчиков: время работы, количество запросов, отправленных или принятых пакетов и т. д.
   Внимание! *На практике использовать счётчики в сыром виде не имеет смысла, в то же время очень интересна скорость роста счётчиков, например за минуту, час, день...*
2. **Gauge (шкала)** — метрика, значение которой может уменьшаться и увеличиваться, типичный пример — количество используемой памяти, загрузка CPU, и т. д.
3. **Histogram (гистограмма)** — комбинация различных счётчиков, обычно метрики этого типа отслеживают продолжительность событий.
4. **Summary (сводка)** — этот тип метрики похож на гистограмму, но у него есть отличие в вычислении квантилей.

PromQL

PromQL — язык запросов, предоставляемый Prometheus для выборки и агрегирования данных временных рядов в реальном времени. Язык выражений Prometheus может быть одним из трёх типов:

- **Instant vector** — набор временных рядов, с единственным экземпляром для каждого временного ряда (имеют одинаковый таймстемп);
- **Range vector** — набор временных рядов, содержащих несколько экземпляров данных для каждого временного ряда;
- **Scalar** — число с плавающей точкой.

Операторы:

Арифметические	Сравнения	Агрегирующие
• + (addition) • - (subtraction) • * (multiplication) • / (division) • % (modulo) • ^ (power/exponentiation)	• == (equal) • != (not-equal) • > (greater-than) • < (less-than) • >= (greater-or-equal) • <= (less-or-equal)	• sum (calculate sum over dimensions) • min (select minimum over dimensions) • max (select maximum over dimensions) • avg (calculate the average over dimensions) • group (all values in the resulting vector are 1) • stddev (calculate population standard deviation over dimensions) • stdvar (calculate population standard variance over dimensions) • count (count number of elements in the vector) • count_values (count number of elements with the same value) • bottomk (smallest k elements by sample value) • topk (largest k elements by sample value) • quantile (calculate φ-quantile (0 ≤ φ ≤ 1) over dimensions)

Также поддерживаются логические операторы and, or, unless.

Помимо операторов, Prometheus поддерживает множество функций, например abs(), delta(), rate()...

```
#возвращает все временные ряды для выбранной метрики
```

```
http_requests_total
#возвращает временные ряды, подпадающими под указанные фильтры
http_requests_total{job="apiserver", handler="/api/comments"}
#возвращает кол-во запросов в секунду, в среднем на протяжении 5 минут
rate(http_requests_total{job="api-server"}[5m])
# возвращает неиспользуемую память в МБ
(instance_memory_limit_bytes - instance_memory_usage_bytes) / 1024 / 1024
```

Exporters

Exporters — это сущности, собирающие метрики с различных сервисов или устройств и выставляющие их на заданный endpoint. Обычно никаких настроек производить не надо, достаточно просто запустить нужный экспортер и указать таргет для Prometheus. На официальном сайте представлено множество экспортеров для различных сервисов, если чего-то там нет, можно попробовать найти это на GitHub.

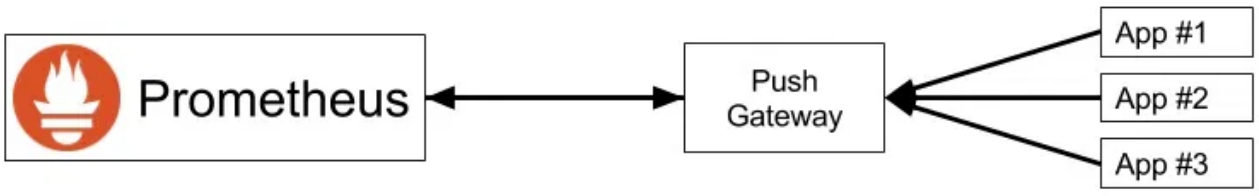
Запустим node exporter для сбора метрик с нашей машины

```
docker run -d \
  --name node-exporter \
  --net="host" \
  --pid="host" \
  -v "/:/host:ro,rslave" \
  quay.io/prometheus/node-exporter:v1.1.0 \
  --path.rootfs=/host
```

О визуализации получаемых метрик с экспортеров расскажем ниже.

Pushgateway

Prometheus Pushgateway — сервис-посредник, позволяющий собирать метрики с целей, которые нельзя очистить (scrape). Один из примеров использования Pushgateway — когда цели имеют малое время жизни (когда интервал очистки целей больше времени жизни этих целей). В этом случае объекты мониторинга отправляют данные на сервис Pushgateway, откуда их забирает Prometheus.



Пример запуска:

```
docker run -d -p 9091:9091 prom/pushgateway:v1.4.0
```

Alerting

Алертинг в Prometheus разделён на две части.

1. Alerting rules — записанные на Prometheus server отправляются в Alertmanager.
2. Alertmanager управляет алертами, на основе конфигурации может не отправлять оповещения, отправлять с задержкой или моментально. Пример конфигурации для отправления оповещений сразу в два канала связи:

```
route:
  receiver: 'default-receiver'
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  group_by: [cluster, alertname]


  routes:
  - receiver: 'telegram'
    group_wait: 10s
    match_re:
      alertname: ".*"
    continue: true

  - receiver: 'rocketchat'
    group_wait: 10s
    match_re:
      alertname: ".*"
    continue: true

receivers:
  - name: 'default-receiver'
    email_configs:
    - to: 'example_email_group@example.org'

  - name: telegram
    webhook_configs:
    - send_resolved: true
      url: http://127.0.0.1:9087/alert/-chat_id

  - name: rocketchat
    webhook_configs:
    - send_resolved: true
      url: http://127.0.0.1:9087/alert/'${WEBHOOK_URL}'
```

Здесь приведена часть конфигурации для Alertmanager, где мы указываем блок route, задающий правила маршрутизации для всех алертов. Здесь можно задать несколько правил для отправления нотификаций с помощью различных receivers.

Alertmanager поддерживает несколько встроенных интеграций, в остальных случаях используются webhooks (список интеграций, который собираются добавить в будущем).

В связи с тем что Alertmanager не очень наглядный инструмент для отображения алертов, то рядом можно поднять Karma или Alerta в качестве наглядной визуализации сработавших алертов.

Security

По умолчанию все пользователи имеют доступ к Prometheus и Alertmanager, т. к. из коробки они не поддерживают никакую аутентификацию, но в официальной документации привёден пример, как можно отгородиться basic auth с помощью Nginx.

Хранение данных

Prometheus может хранить данные как в встроенной TSDB, так и записывать в удалённое хранилище. Для этого используется директива remote-write.

Внимание! *При использовании параметра remote-write увеличивается использование оперативной памяти примерно на 25%*

ClickHouse — колоночная аналитическая СУБД для онлайн-обработки аналитических запросов (OLAP). Подключение к ClickHouse производится с помощью адаптера.

Thanos обеспечивает global query view, высокую доступность, резервное копирование данных (все эти функции опциональны). Thanos берёт данные, которые сохранил Prometheus на локальный диск, и копирует их в S3, в GCS либо в другой object storage. Также Thanos поддерживает PromQL и Prometheus Quering API.

VictoriaMetrics — это быстрое, экономичное и масштабируемое решение для мониторинга. Сейчас может использоваться как отдельная система мониторинга, включающая мониторинг агентов, базу данных, систему алертинга и бэкап данных. Применимо к Prometheus поддерживает PromQL, Prometheus Quering API, обеспечивает global query view. Хорошо оптимизирована по потребляемым ресурсам.

Grafana

Grafana — это платформа для визуализации, мониторинга и анализа данных. Grafana позволяет создавать дашборды с набором панелей (графики, диаграммы, таблицы и т. д.), каждая из которых может отображать различные данные. Начиная с 2015, Grafana поддерживает Prometheus как источник данных.

Запуск и базовая настройка

Запустить Grafana можно несколькими способами, но, как обычно, для простоты используем Docker

```
docker run -d -p 3000:3000 --name grafana grafana/grafana
```

Теперь, когда у нас запущены Prometheus, Grafana и node-exporter, мы можем посмотреть на красивые графики о состоянии нашей системы. Для этого нужно сделать ещё несколько шагов:

1. Добавить Datasource в Grafana (Configuration -> Data Sources -> Add data source - Prometheus select -> URL http://<ip_вашей_машины>:9090 -> Save and test.
2. Скачать дашборд для Node exporter со страницы https://grafana.com/grafana/dashboards/1860.
3. Импортировать его в Grafana: Create -> Import -> Upload JSON file -> select file -> Prometheus Prometheus.

Внимание! *Найти необходимые дашборды можно по адресу https://grafana.com/grafana/dashboards, также они могут находиться в репозитории вместе с приложением/экспортёром.*

Интерфейс

Разберёмся с интерфейсом Grafana:

1. По умолчанию включена Basic Auth и при запуске вас попросят сменить дефолтный пароль.
2. После входа попадаем на домашнюю страницу, которую можно поменять.
   Внимание! *Есть как минимум 2 способа сменить домашний дашборд: 1) Выбрать существующий дашборд, нажать на звёздочку (make as favorite) и в настройках выбрать этот дашборд (Configuration->Home Dashboard-><Your Dashboard>; 2) Задать путь до вашего дашборда, например, с помощью переменной "**GF_AUTH_DEFAULT_HOME_DASHBOARD_PATH**" или в файле **/etc/grafana/grafana.ini** задать параметр default home dashboard path. Ещё можно при запуске сервиса передать аргумент **--homepath**.*

3. С левой стороны мы видим панель управления

	Переход на домашний дашборд
	Поиск дашбордов по папкам, имени, тегам
	Создание дашбордов и папок, также можно импортировать дашборды
	Управление дашбордами
	Эта вкладка убирает всё лишнее и позволяет опробовать свои запросы
	Управление алертами
	Конфигурирование: добавление источников данных, управление пользователями и командами, подключение плагинов, настройка предпочтений и управление API ключами
	Администрирование: создание и удаление пользователей и организаций, просмотр конфигурации и статистики
	Управление аккаунтом

Дашборды и панели

Дашборды можно создавать, экспортировать, импортировать и редактировать. Создать пустой дашборд очень просто, достаточно нажать Create->Dashboard, после чего вам предложат создать панель или конвертнуть её в строку в новом дашборде.

Панели имеют несколько встроенных типов визуализации, в том числе Graph, Text, Table и другие, но одними встроенными типами дело не ограничивается и всегда можно подключить к вашей Grafana сторонние плагины, которые могут добавить дополнительные типы. В каждом типе есть свои настройки для отображения данных, а также некоторые из них поддерживают алертинг (редко используемая в Grafana функция).

Внимание! *Ещё одна особенность многих панелей — трансформация данных (Transform), которая позволяет преобразовывать результаты запросов перед визуализацией.*

Дашборды можно шаблонизировать с помощью переменных, задаваемых к каждому дашборду отдельно. Например, чтобы выбрать все доступные ноды из Prometheus, можно добавить переменную, где дадим ей имя (node), выберем тип (query), укажем источник данных (Prometheus) и зададим сам запрос: ***label_values(node_uname_info{job="$job"}, instance),*** здесь мы можем видеть, что фильтрация по *job* задается также с помощью переменной (она также должна быть определена).

В настройках к дашбордам также можно добавить теги, слинковать с другими дашбордами или какими-нибудь внешними ресурсами.

Группы и доступы

Доступ в Grafana производится на основе организаций. В каждой организации может быть свой набор дашбордов и источников данных.

Доступ к данным в Grafana осуществляется на основе следующих правил:

1. В Grafana существуют так называемые организации, которые имеют свой набор источников данных, дашбордов, а также заданных пользователей.
2. Внутри каждой организации настраивается роли для пользователей (Admin, Edit, View).
3. При необходимости можно переопределить права на отдельный дашборд в организации.

Внимание! *Без особых усилий можно настроить аутентификацию по LDAP и замапить (сопоставить) пользователей или пользовательские группы к определенной организации.*

Практическое задание

1. Prometheus:
    1.1. Запустить Prometheus.
    1.2. Настроить мониторинг для инстанса, на котором запущен Prometheus.
    1.3. Настроить Alertmanager на отправку оповещений в Telegram.
    1.4. *Настроить basic auth для Alertmanager и Prometheus.
    1.5. *Запустить и настроить Karama и/или Alerta для визуализации алертов.
2. Grafana:
    2.1. Запустить Grafana.
    2.2. Добавить дашборд node-exporter и источник данных из первого задания.
    2.3. * Настроить ограниченный доступ для нового пользователя GeekBrain-User:
        2.3.1. Роль Viewer в организации GeekBrain-Ogr.
        2.3.2. Роль Editor для дашборда Node-exporter в организации GeekBrain-Ogr.

Глоссарий

Экспортер — часть программного обеспечения, которое получает существующие метрики от сторонней системы и экспортирует их в формат, понятный серверу Prometheus.

Квантиль — значение, которое не превышает фиксированную вероятность.

Дополнительные материалы

1. VictoriaMetrics · The Aspiring Monitoring Solution.
2. Highly available Prometheus setup with long term storage capabilities.
3. Introduction to PromQL, the Prometheus Query Language.
4. samber/awesome-prometheus-alerts: 🚨 Collection of Prometheus alerting rules.
5. Blog - Metric Types in Prometheus and PromQL.
6. Типы метрик.
7. Как работает гистограмма Prometheus? / Блог компании OTUS / Хабр.
8. Alerta monitoring system — alerta 7.5 documentation.
9. prymitive/karma: Alert dashboard for Prometheus Alertmanager.
10. Мониторинг с помощью Prometheus. Мониторинг приложений и серверов… | by Igor Olemskoi | Southbridge.
11. Выбираем хранилище данных для Prometheus: Thanos vs VictoriaMetrics.

Используемые источники

1. Overview.
2. prometheus-operator/prometheus-operator: Prometheus Operator creates/configures/manages Prometheus clusters atop Kubernetes.
3. The 4 Types Of Prometheus Metrics – Tom Gregory.
4. discovery.
5. Grafana documentation.
6. Grafana Dashboards - discover and share dashboards for Grafana.

7. Plugins - extend and customize your Grafana | Grafana Labs