Мониторинг приложения и инфраструктуры

Проект microservices и проверка ДЗ

Создайте новую ветку в вашем `microservices`-репозитории для выполнения данного ДЗ. Это второе задание про мониторинг, ветку назовите **monitoring-2**.

Проверка данного ДЗ будет производиться через Pull Request ветки с ДЗ.

После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить и закрыть Pull Request.

План

- Мониторинг Docker контейнеров
- Визуализация метрик
- Сбор метрик работы приложения и бизнес метрик
- Настройка и проверка алертинга
- Много заданий со ⭐ (необязательных)

Подготовка окружения

1. Открывать порты в файрволле для новых сервисов нужно **самостоятельно** по мере их добавления.
2. Создадим Docker хост в GCE и настроим локальное окружение на работу с ним (ссылка на gist):

Подготовка окружения

```
$ export GOOGLE_PROJECT=_ваш-проект_

$ docker-machine create --driver google \
    --google-machine-image
https://www.googleapis.com/compute/v1/projects/ubuntu-os-
cloud/global/images/family/ubuntu-1604-lts \
    --google-machine-type n1-standard-1 \
    --google-zone europe-west1-b \
    docker-host
...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this
virtual machine, run: docker-machine env docker-host

$ eval $(docker-machine env docker-host)

# узнаем IP адрес
$ docker-machine ip docker-host
```

Мониторинг Docker контейнеров

Разделим файлы Docker Compose:

В данный момент и мониторинг и приложения у нас описаны в одном большом `docker-compose.yml`. С одной стороны это просто, а с другой - мы смешиваем различные сущности, и сам файл быстро растет.

Оставим описание приложений в `docker-compose.yml`, а мониторинг выделим в отдельный файл `docker-compose-monitoring.yml`.

Для запуска приложений будем как и ранее использовать `docker-compose up -d`, а для мониторинга - `docker-compose -f docker-compose-monitoring.yml up -d`

cAdvisor

Мы будем использовать cAdvisor для наблюдения за состоянием наших Docker контейнеров.

cAdvisor

cAdvisor собирает информацию о ресурсах потребляемых контейнерами и характеристиках их работы.

Примерами метрик являются:

- процент использования контейнером CPU и памяти, выделенные для его запуска,
- объем сетевого трафика
- и др.

Файл docker-compose-monitoring.yml

cAdvisor также будем запускать в контейнере. Для этого добавим новый сервис в наш компоуз файл мониторинга `docker-compose-monitoring.yml` (ссылка на Gist).

Поместите данный сервис в одну сеть с Prometheus, чтобы тот мог собирать с него метрики.

```
services:
...
  cadvisor:
    image: google/cadvisor:v0.29.0
    volumes:
      - '/:/rootfs:ro'
      - '/var/run:/var/run:rw'
      - '/sys:/sys:ro'
      - '/var/lib/docker/:/var/lib/docker:ro'
    ports:
      - '8080:8080'
```

Файл prometheus.yml

Добавим информацию о новом сервисе в конфигурацию Prometheus, чтобы он начал собирать метрики:

```
scrape_configs:
...
  - job_name: 'cadvisor'
    static_configs:
      - targets:
        - 'cadvisor:8080'
```

Пересоберем образ Prometheus с обновленной конфигурацией:

```
$ export USER_NAME=username # где username - ваш логин на Docker Hub
$ docker build -t $USER_NAME/prometheus .
```

cAdvisor UI

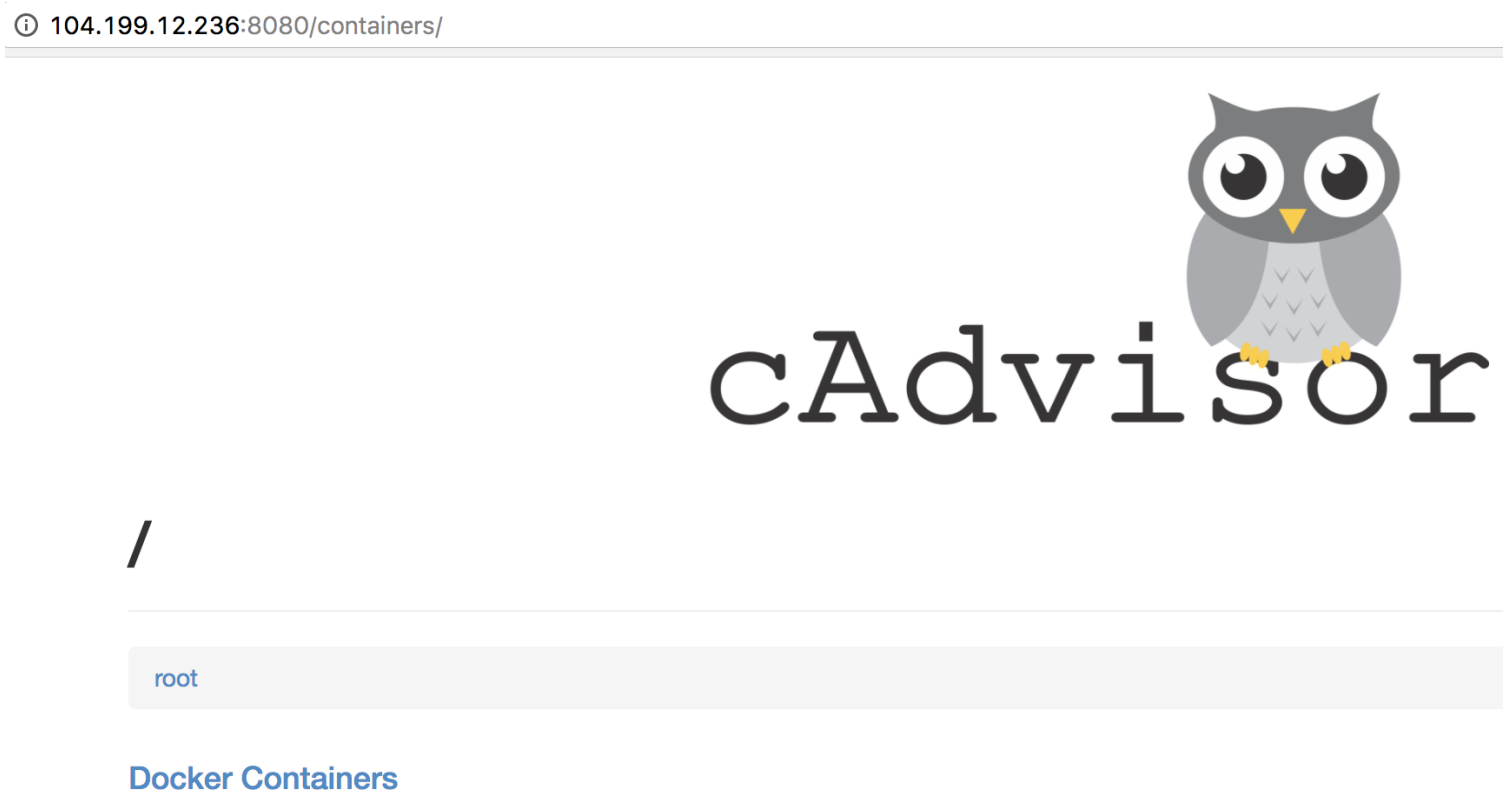
Запустим сервисы:

```
$ docker-compose up -d
$ docker-compose -f docker-compose-monitoring.yml up -d
```

cAdvisor имеет UI, в котором отображается собираемая о контейнерах информация.

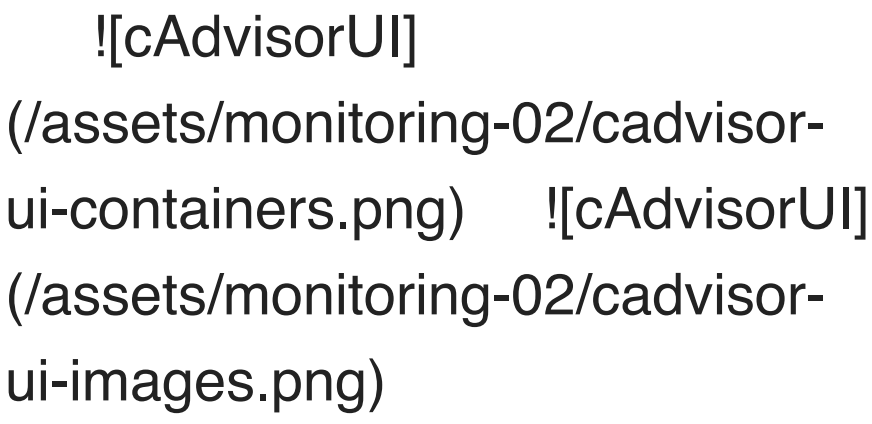
Откроем страницу Web UI по адресу `http://<docker-machine-host-ip>:8080`

cAdvisor UI



Нажмите ссылку **Docker Containers** (внизу слева) для просмотра информации по контейнерам

cAdvisor UI



В UI мы можем увидеть:

- список контейнеров, запущенных на хосте
- информацию о хосте (секция *Driver Status*)
- информацию об образах контейнеров (секция *Images*)

cAdvisor UI

Нажмем на название одного из контейнеров, чтобы посмотреть информацию о его работе:

Subcontainers

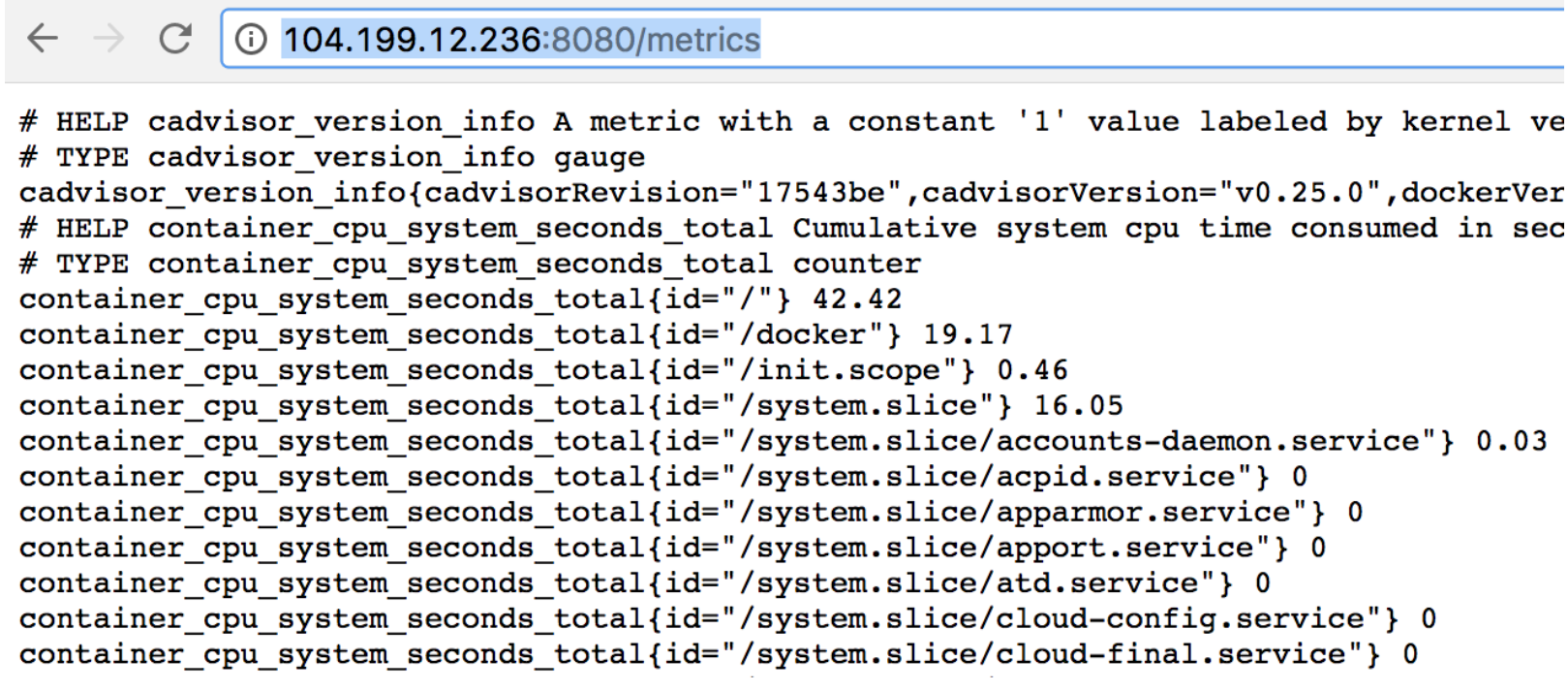
- reddit_cadvisor_1 (/docker/b6cb859eab1261ee3849082751ecb04ec19393369868263cc85e169c18a1a059)
- reddit_prometheus_1 (/docker/fd8322bf5904c32386dad54812098cbd6e63f21be33c42453ff063d94cabd36c)
- reddit_post_db_1 (/docker/b36bfd998eb6c3c9625e1e0a45c55ba2ef988f8158861a8c2e9a9d54c2dea126)
- reddit_comment_db_1 (/docker/2c675170c46f0b389eb92becac45fe9320781e70492ef12d50fd4be61df3b91a)
- reddit_node-exporter_1 (/docker/5334d725737cb8e46ed9fe2ed50ebe8b0609caa465c6446b0702261db4c7cbd4)

cAdvisor UI

Здесь отображается информация по процессам, использованию CPU, памяти, сети и файловой системы:

cAdvisor UI

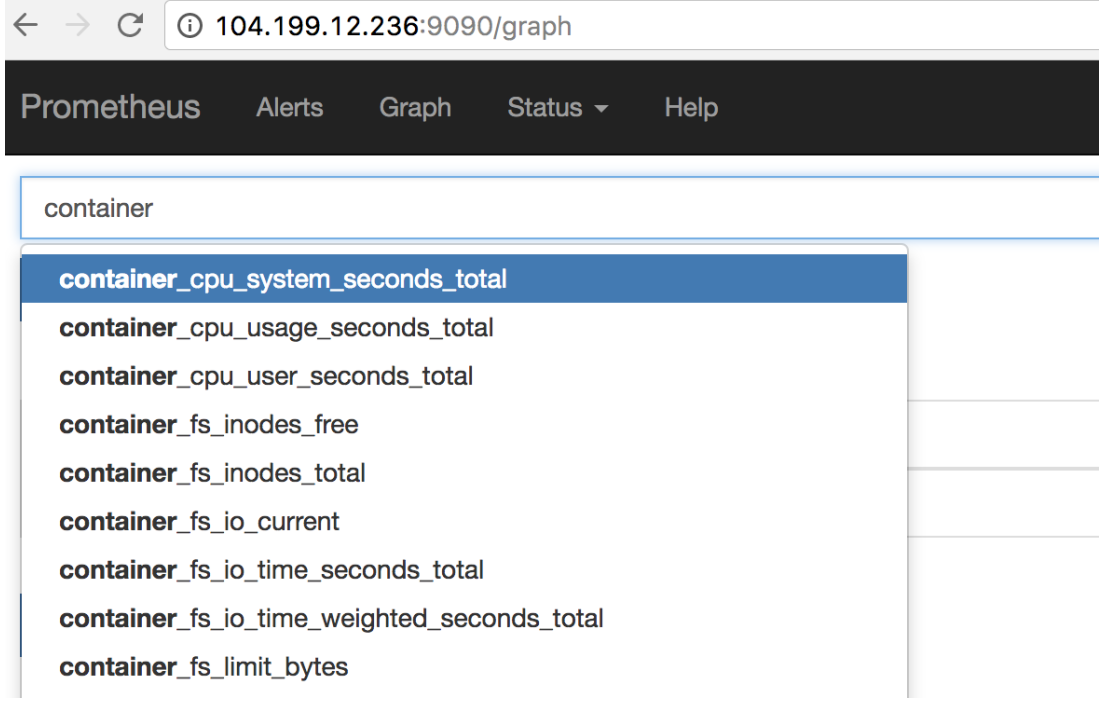
По пути `/metrics` все собираемые метрики публикуются для сбора Prometheus:



Видим, что имена метрик контейнеров начинаются со слова `container`

cAdvisor UI

Проверим, что метрики контейнеров собираются Prometheus. Введем, слово `container` и посмотрим, что он предложит дополнить:



Визуализация метрик: Grafana

Используем инструмент Grafana для визуализации данных из Prometheus.

Добавим новый сервис в `docker-compose-monitoring.yml`, используя пример со следующего слайда или этот gist

> ℹ️ Не забудьте добавить сервис **grafana** в одну сеть с Prometheus:

Визуализация метрик: Grafana

Пример конфигурации для Docker Compose:

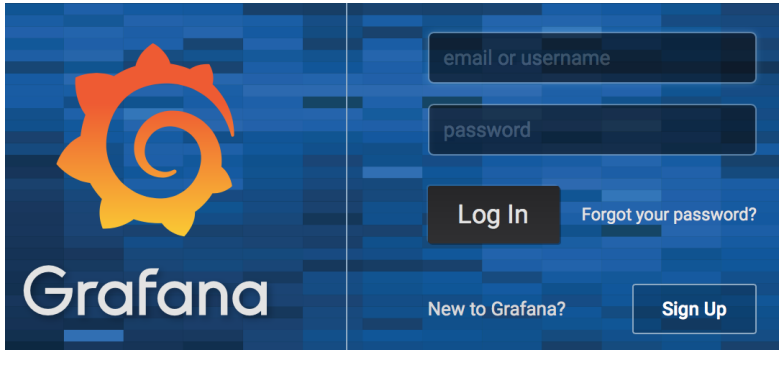
```
services:
...
  grafana:
    image: grafana/grafana:5.0.0
    volumes:
      - grafana_data:/var/lib/grafana
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=secret
    depends_on:
      - prometheus
    ports:
      - 3000:3000
...
volumes:
  grafana_data:
```

Grafana: Web UI

Запустим новый сервис:

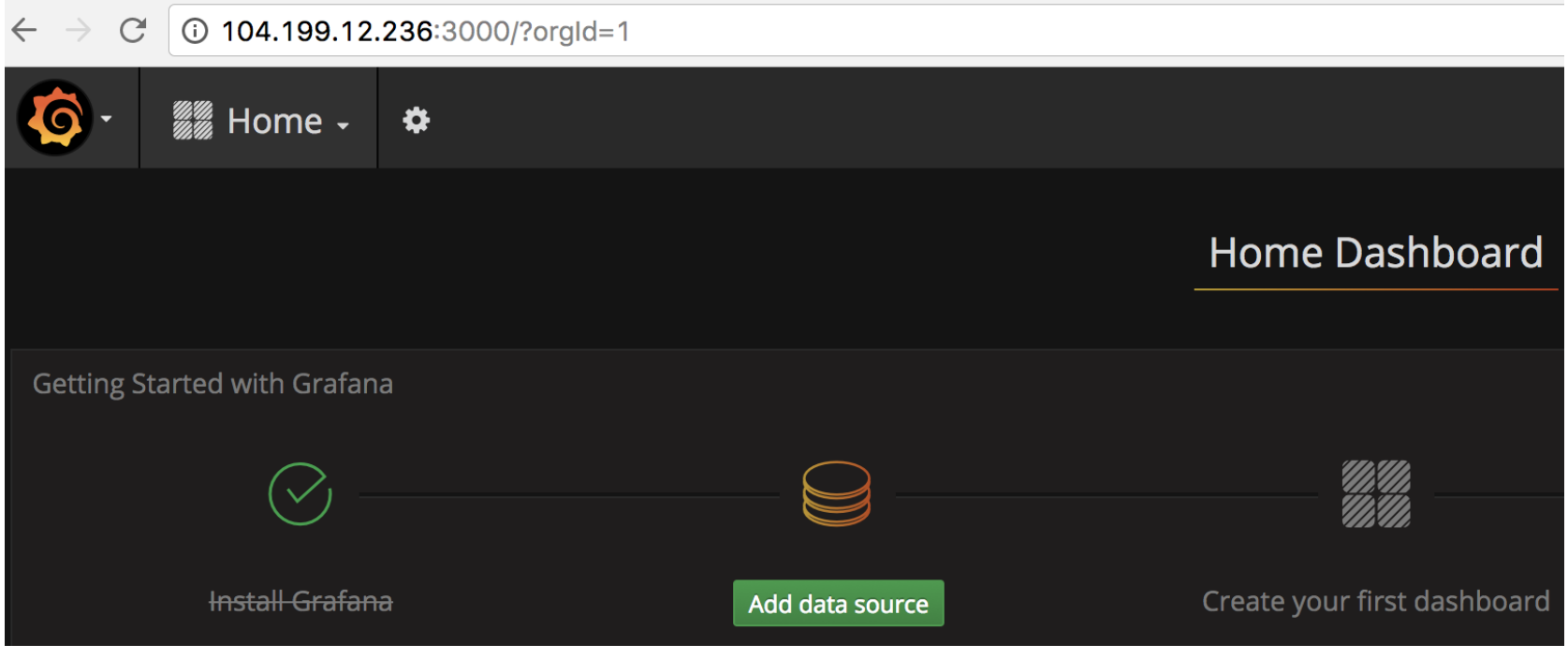
```
$ docker-compose -f docker-compose-monitoring.yml up -d grafana
```

Откроем страницу Web UI Grafana по адресу `http://<docker-machine-host-ip>:3000` и используем для входа логин и пароль администратора, которые мы передали через переменные окружения:



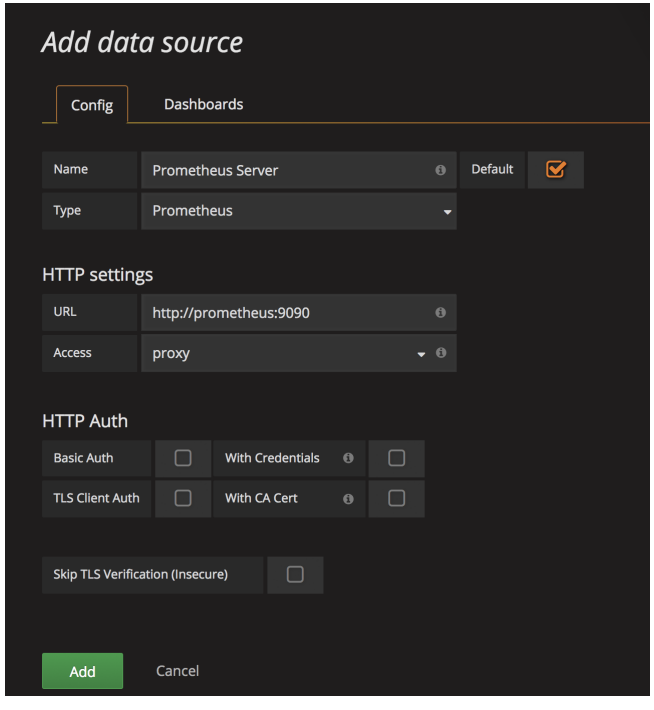
Grafana: Добавление источника данных

Нажмем **Add data source** (Добавить источник данных):



Grafana: Добавление источника данных

Зададим нужный тип и параметры подключения:

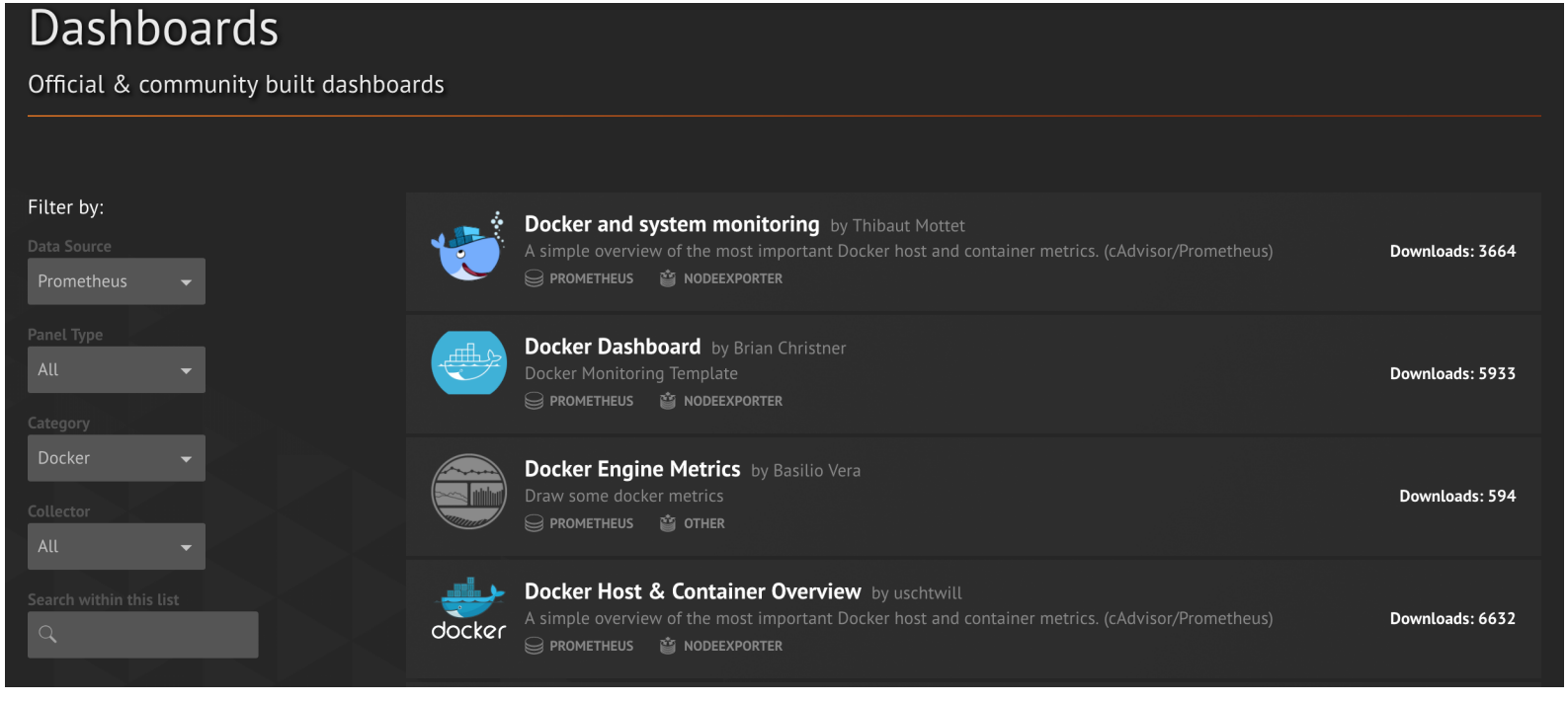


- **Name**: Prometheus Server
- **Type**: Prometheus
- **URL**: http://prometheus:9090
- **Access**: Proxy

И затем нажмем **Add**

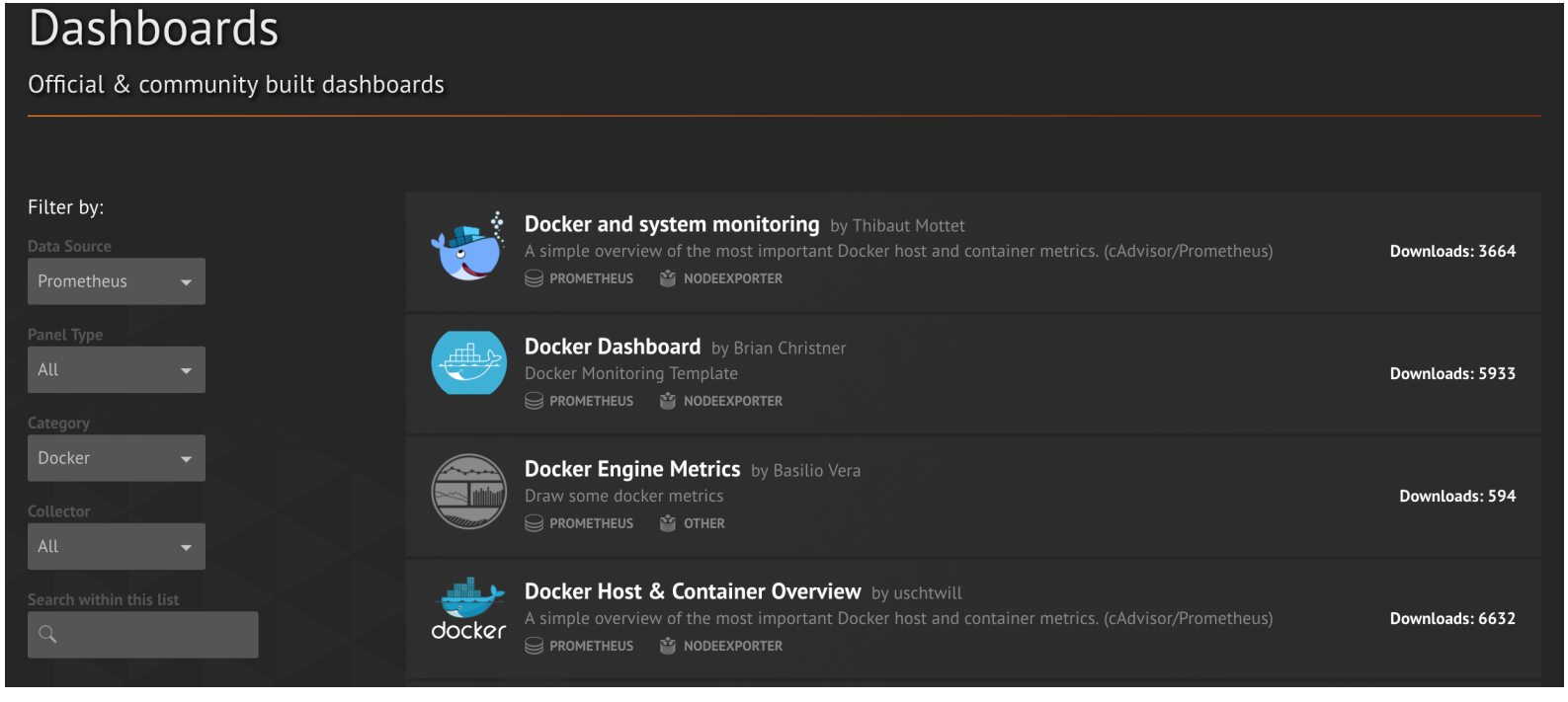
Дашборды

Перейдем на сайт Grafana, где можно найти и скачать большое количество уже созданных официальных и комьюнити дашбордов для визуализации различного типа метрик для разных систем мониторинга и баз данных

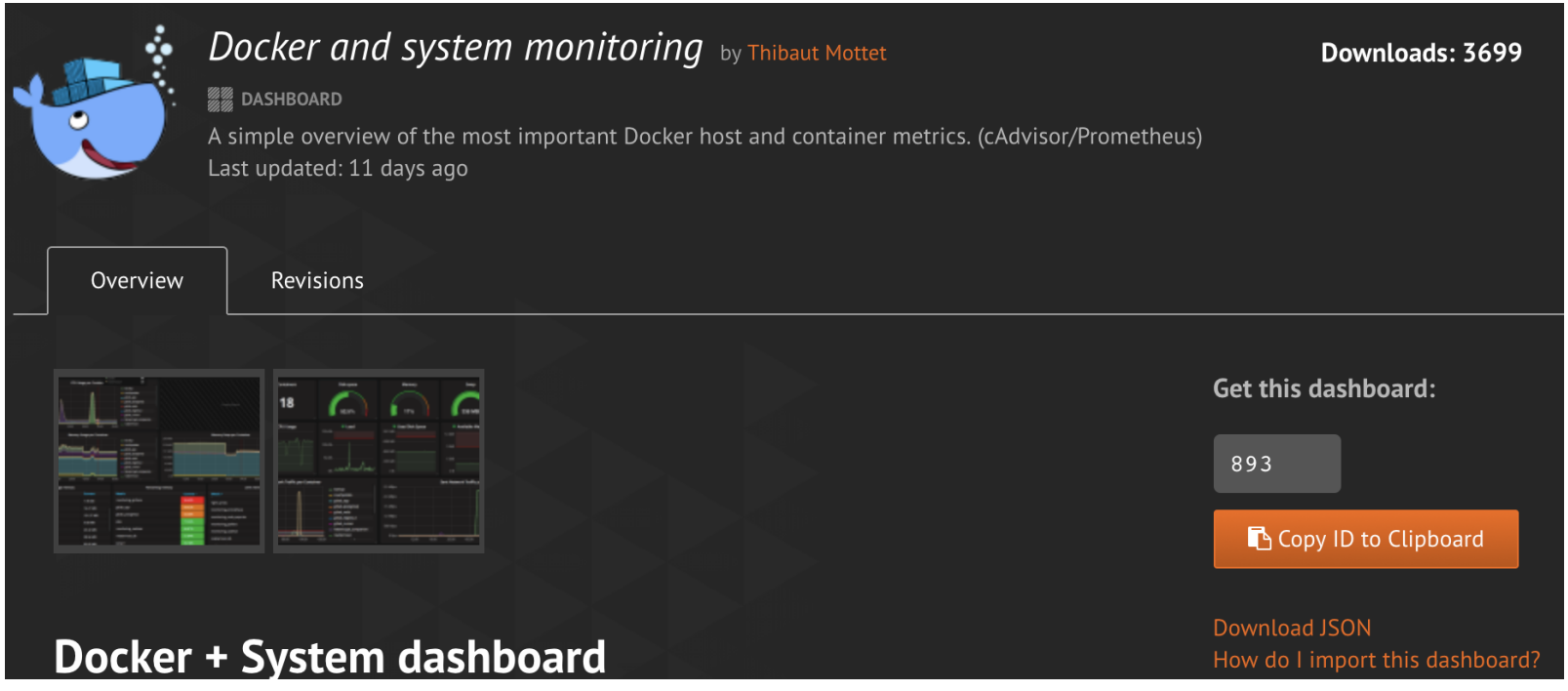


Дашборды

Выберем в качестве источника данных нашу систему мониторинга (Prometheus) и выполним поиск по категории Docker. Затем выберем популярный дашборд:

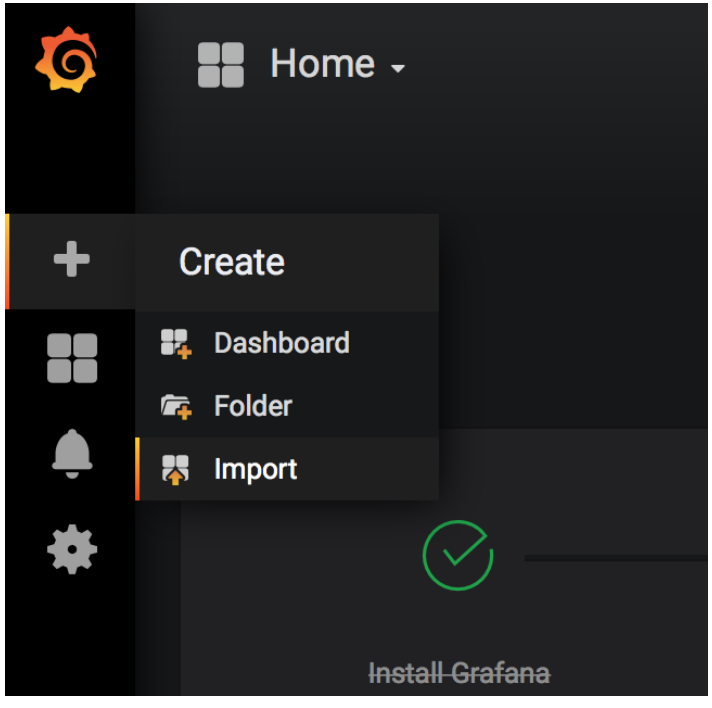


Дашборды



Нажмем **Загрузить JSON**. В директории `monitoring` создайте директории `grafana/dashboards`, куда поместите скачанный дашборд. Поменяйте название файла дашборда на `DockerMonitoring.json`.

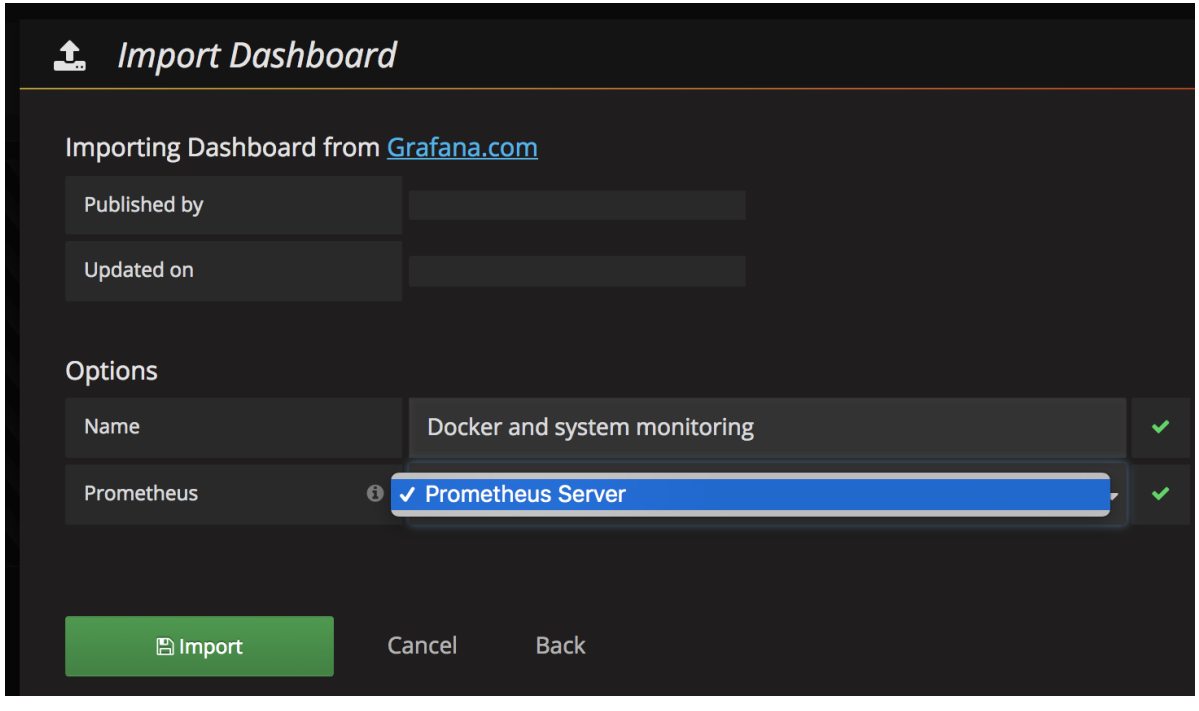
Импорт дашборда



Снова откроем веб-интерфейс Grafana и выберем импорт шаблона (**Import**)

Импорт дашборда

Загрузите скачанный дашборд. При загрузке укажите источник данных для визуализации (*Prometheus Server*):



Импорт дашборда

Должен появиться набор графиков с информацией о состоянии хостовой системы и работе контейнеров:

Мониторинг работы приложения

В качестве примера метрик приложения в сервис UI мы добавили:

- счетчик `ui_request_count`, который считает каждый приходящий HTTP-запрос (добавляя через лейблы такую информацию как HTTP метод, путь, код возврата, мы уточняем данную метрику)
- гистограмму `ui_request_latency_seconds`, которая позволяет отслеживать информацию о времени обработки каждого запроса

Мониторинг работы приложения

В качестве примера метрик приложения в сервис Post мы добавили:

- Гистограмму `post_read_db_seconds`, которая позволяет отследить информацию о времени требуемом для поиска поста в БД

Зачем?

Созданные метрики придадут видимости работы нашего приложения и понимания, в каком состоянии оно сейчас находится.

Например, время обработки HTTP запроса не должно быть большим, поскольку это означает, что пользователю приходится долго ждать между запросами, и это ухудшает его общее впечатление от работы с приложением. Поэтому большое время обработки запроса будет для нас сигналом проблемы.

Отслеживая приходящие HTTP-запросы, мы можем, например, посмотреть, какое количество ответов возвращается с кодом ошибки. Большое количество таких ответов также будет служить для нас сигналом проблемы в работе приложения.

prometheus.yml

Добавим информацию о post-сервисе в конфигурацию Prometheus, чтобы он начал собирать метрики и с него:

```
scrape_configs:
...
  - job_name: 'post'
    static_configs:
      - targets:
        - 'post:5000'
```

Пересоберем образ Prometheus с обновленной конфигурацией:

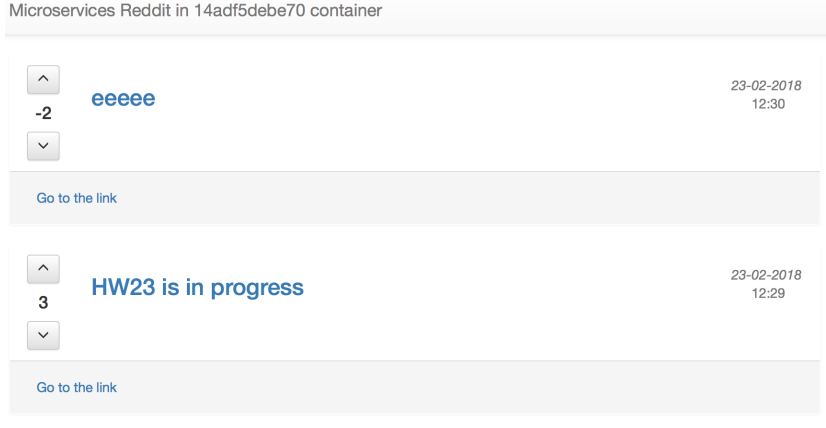
```
$ export USER_NAME=username  # где, usename - ваш логин от DockerHub
$ docker build -t $USER_NAME/prometheus .
```

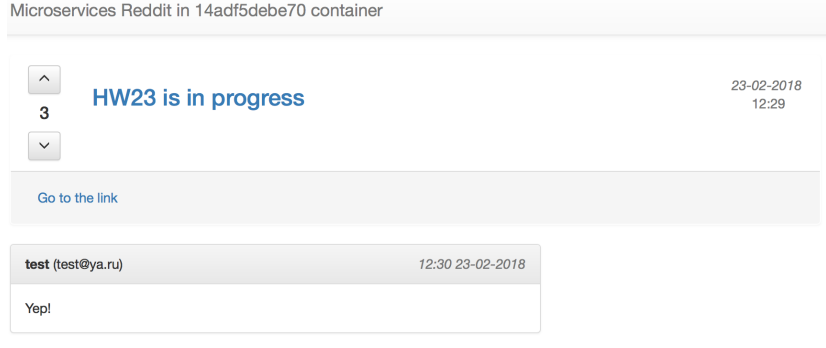
prometheus.yml

Пересоздадим нашу Docker инфраструктуру мониторинга:

```
$ docker-compose -f docker-compose-monitoring.yml down
$ docker-compose -f docker-compose-monitoring.yml up -d
```

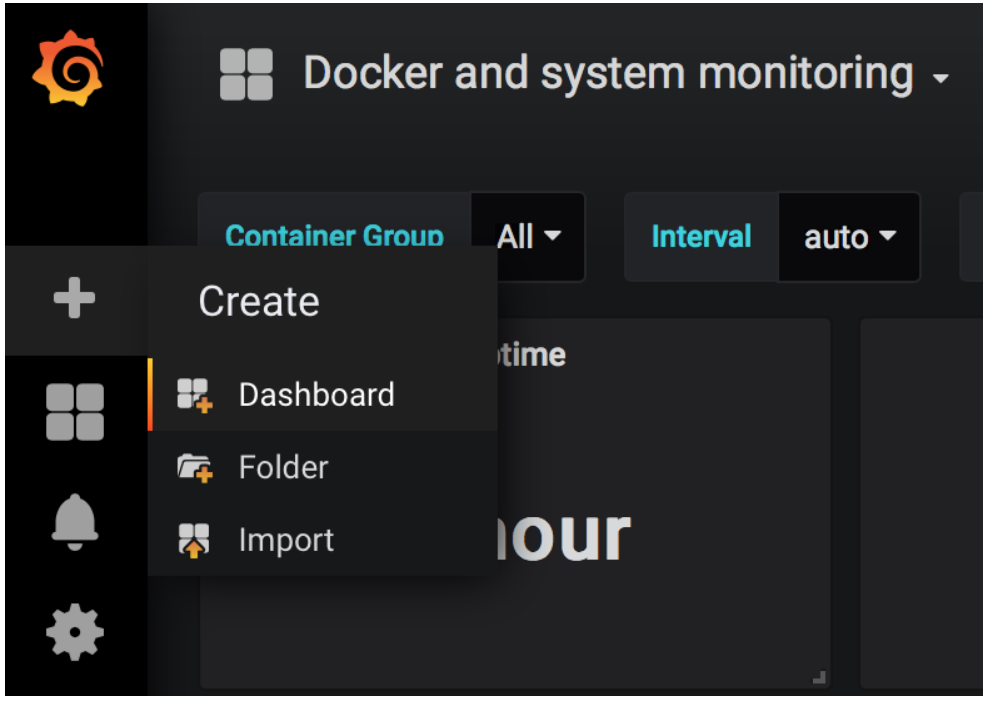
И добавим несколько постов в приложении и несколько комментов, чтобы собрать значения метрик приложения:





Создание дашборда в Grafana

Построим графики собираемых метрик приложения. Выберем создать новый дашборд:



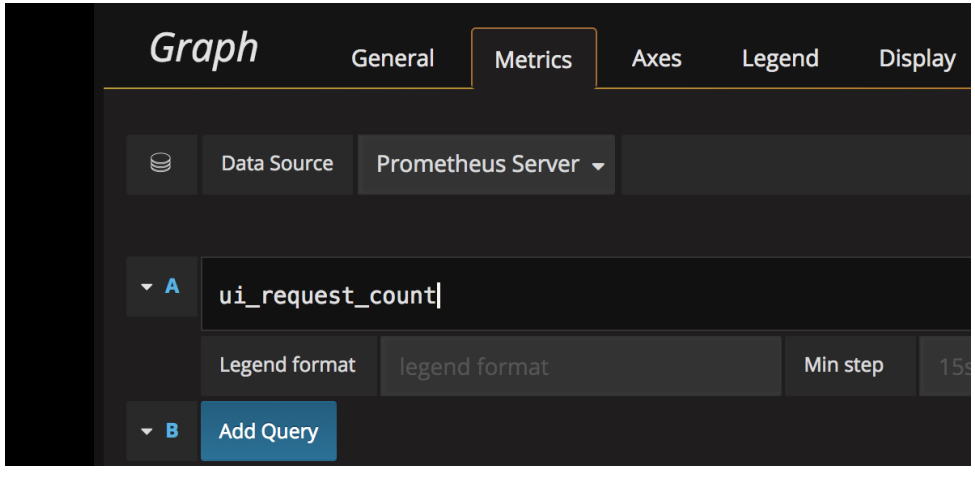
Снова откроем веб-интерфейс Grafana и выберем создание шаблона (**Dashboard**)

Создание дашборда в Grafana

1. Выбираем "Построить график" (**New Panel ➡️ Graph**)
2. Жмем один раз на имя графика (**Panel Title**), затем выбираем **Edit**:

Создание дашборда в Grafana

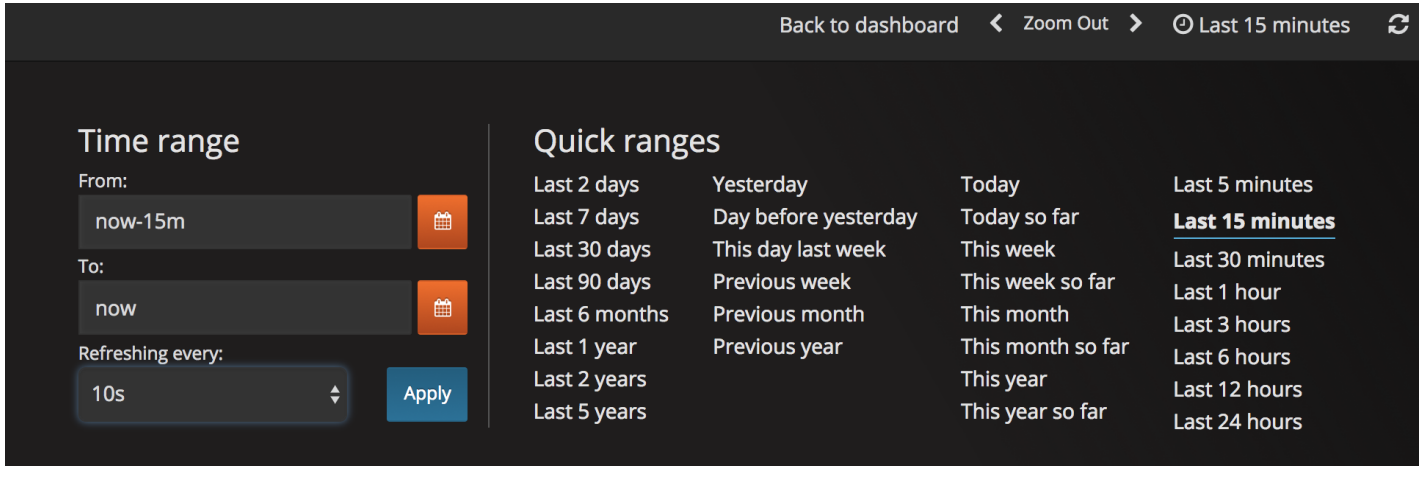
Построим для начала простой график изменения счетчика HTTP-запросов по времени. Выберем источник данных и в поле запроса введем название метрики:



Далее достаточно нажать мышкой на любое место UI, чтобы убрать курсор из поля запроса, и Grafana выполнит запрос и построит график

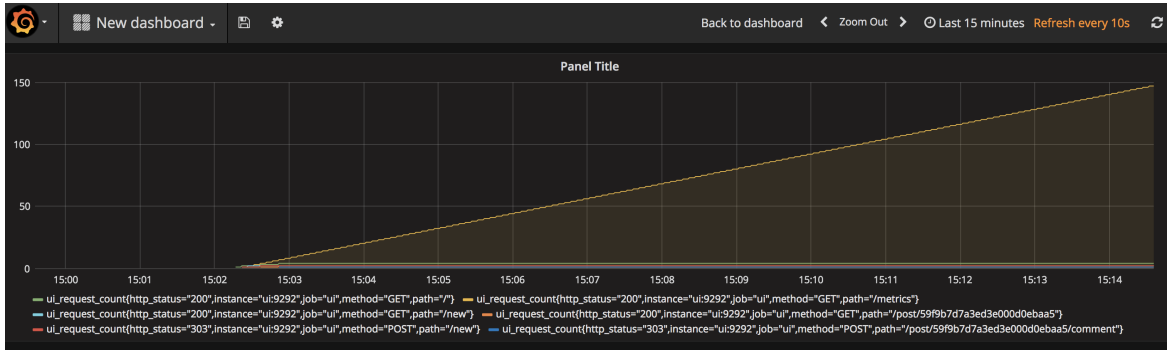
Создание дашборда в Grafana

В правом верхнем углу мы можем уменьшить временной интервал, на котором строим график, и настроить автообновление данных:

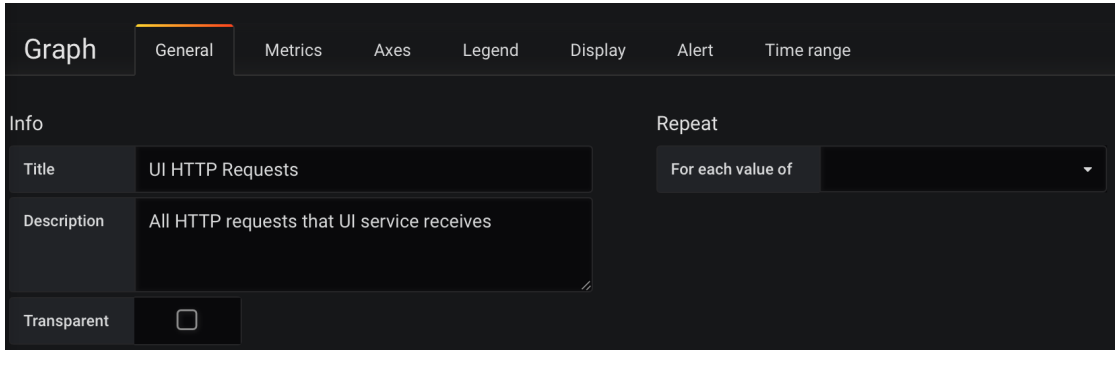


Создание дашборда в Grafana

Сейчас мы с вами получили график различных HTTP запросов, поступающих UI сервису:

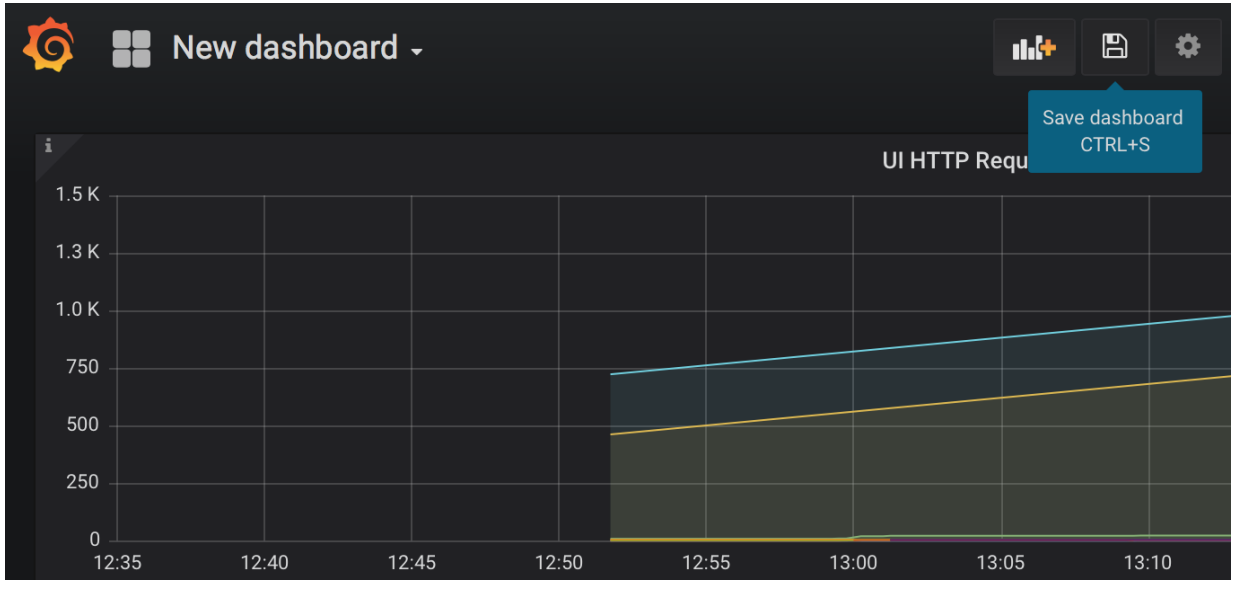


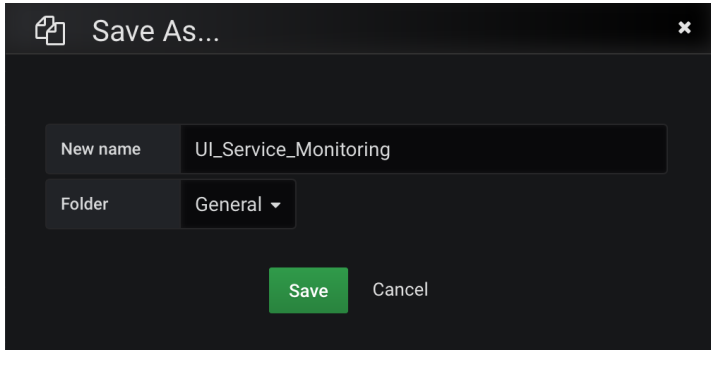
Изменим заголовок графика и описание:



Создание дашборда в Grafana

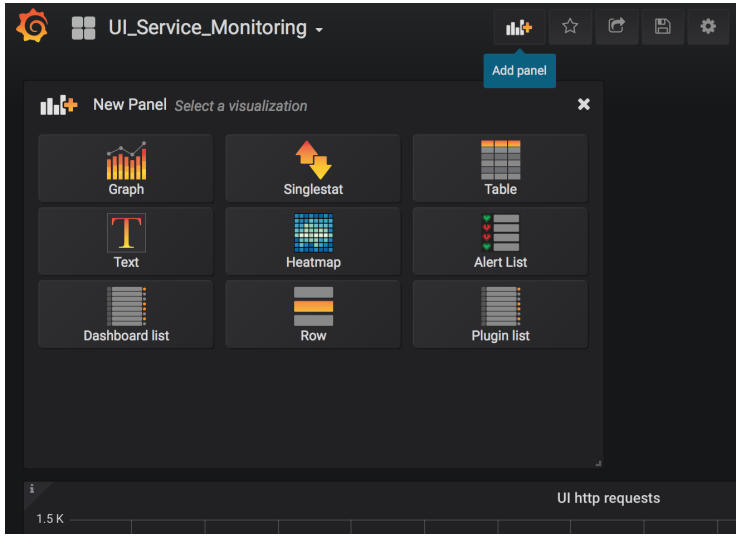
Сохраним созданный дашборд:



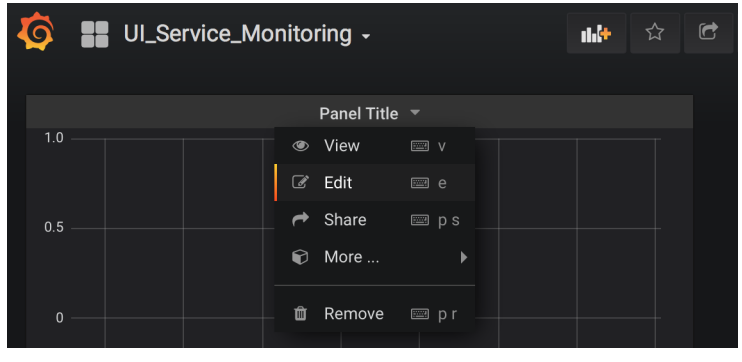


Создание дашборда в Grafana

Построим график запросов, которые возвращают код ошибки на этом же дашборде. Добавим еще один график на наш дашборд:

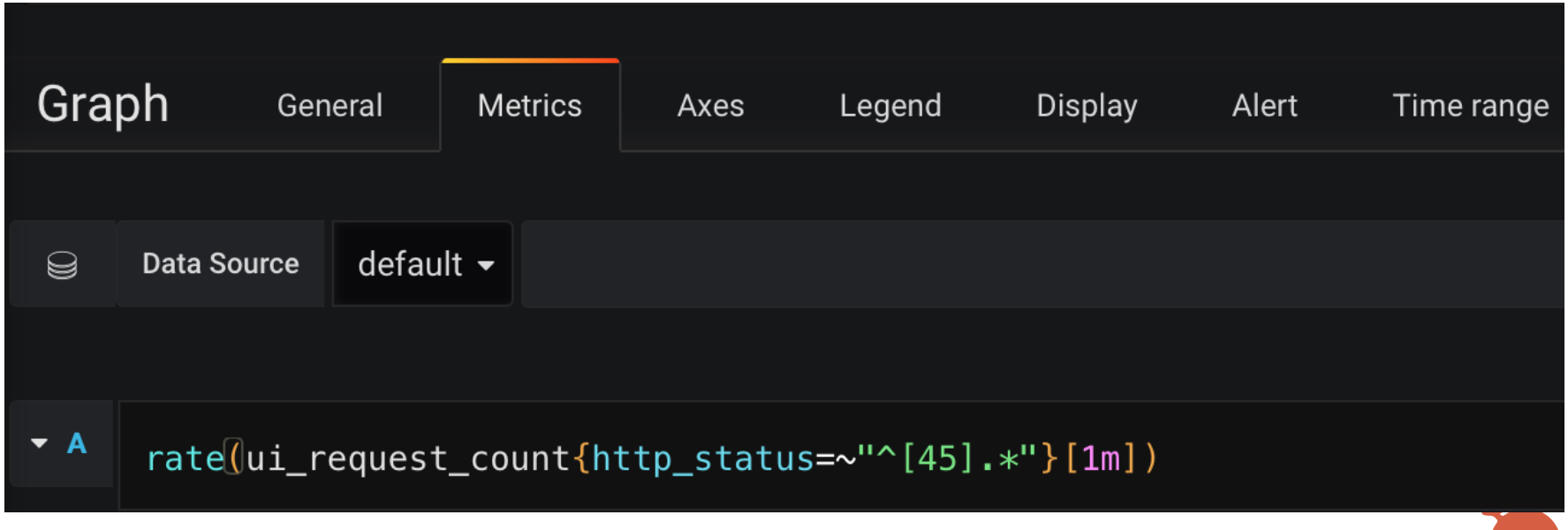


Переходим в режим правки графика:



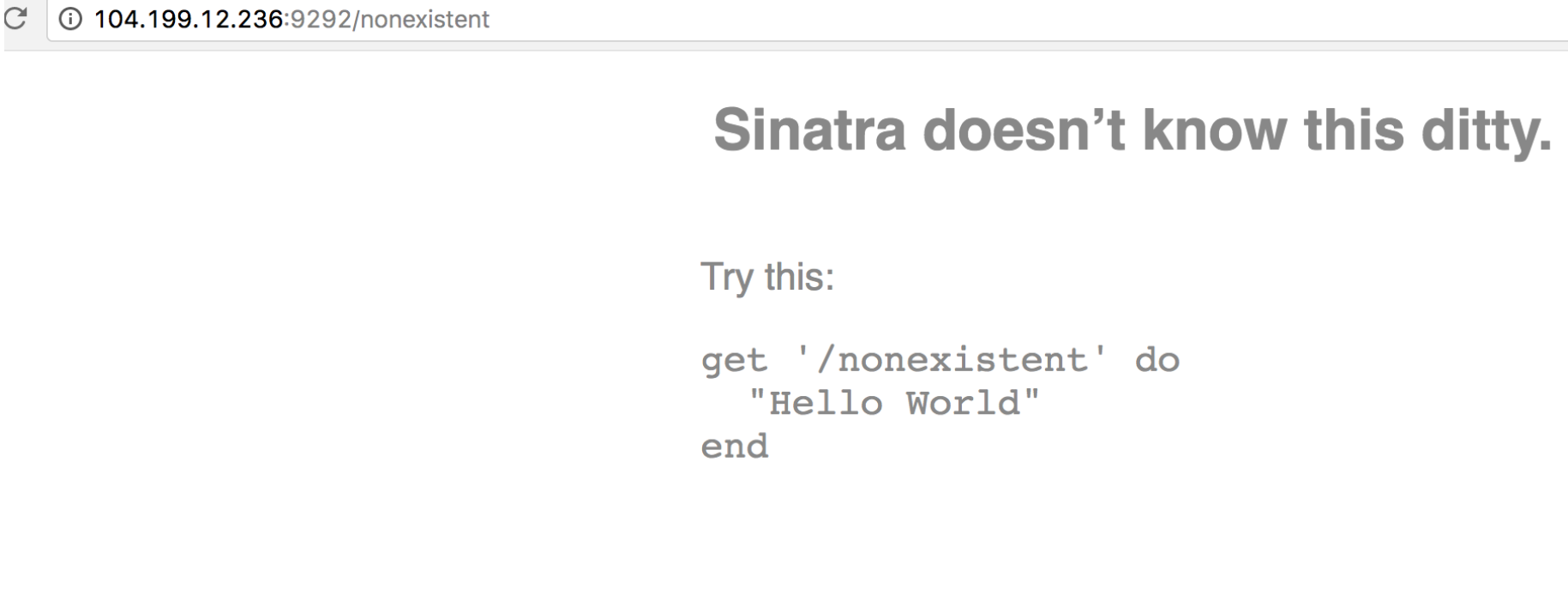
Создание дашборда в Grafana

В поле запросов запишем выражение для поиска всех http запросов, у которых код возврата начинается либо с 4 либо с 5 (используем регулярное выражения для поиска по лейблу). Будем использовать функцию rate(), чтобы посмотреть не просто значение счетчика за весь период наблюдения, но и скорость увеличения данной величины за промежуток времени (возьмем, к примеру 1-минутный интервал, чтобы график был хорошо видим)



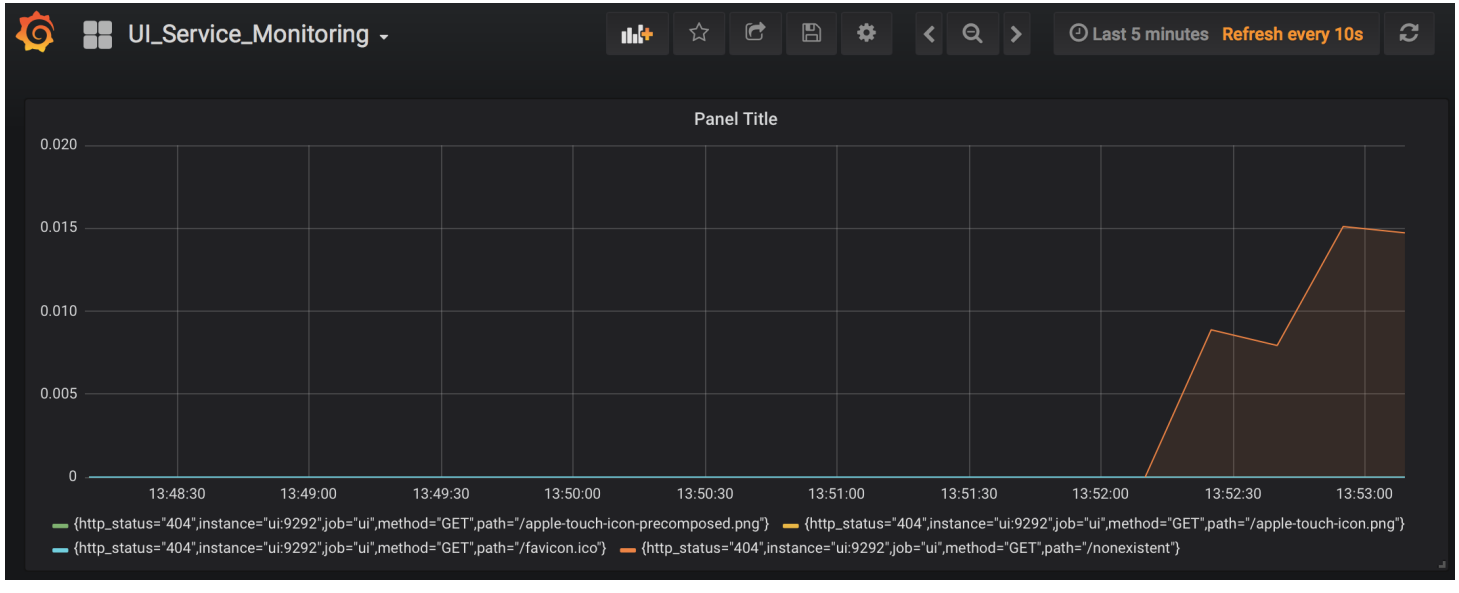
Создание дашборда в Grafana

График ничего не покажет, если не было запросов с ошибочным кодом возврата. Для проверки правильности нашего запроса обратимся по несуществующему HTTP пути, например, http://:9292/nonexistent, чтобы получить код ошибки 404 в ответ на наш запрос.



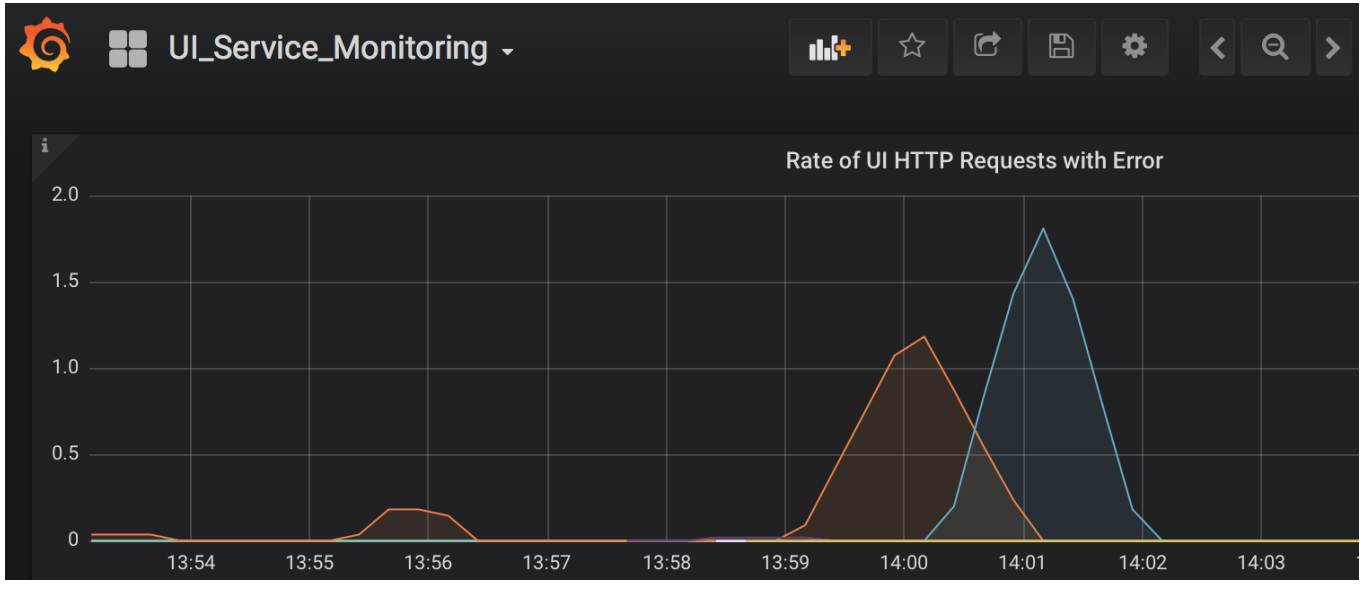
Создание дашборда в Grafana

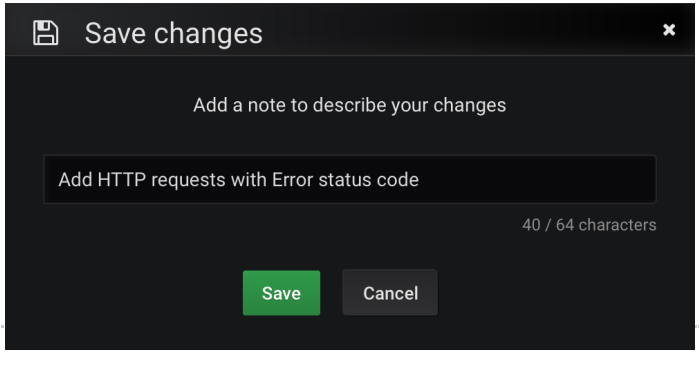
Проверим график (временной промежуток можно уменьшить для лучшей видимости графика)



Создание дашборда в Grafana

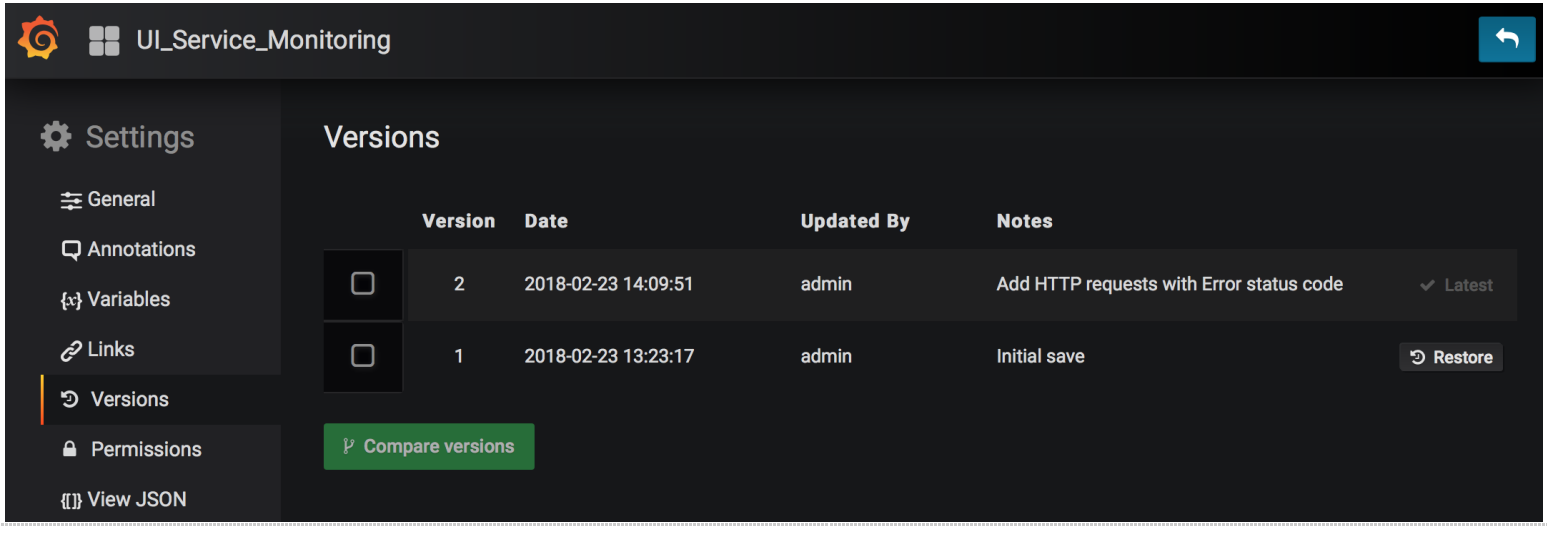
Добавьте заголовок и описание графика и нажмите сохранить изменения дашборда





Создание дашборда в Grafana

Grafana поддерживает версионирование дашбордов, именно поэтому при сохранении нам предлагалось ввести сообщение, поясняющее изменения дашборда. Вы можете посмотреть историю изменений своего



Самостоятельно

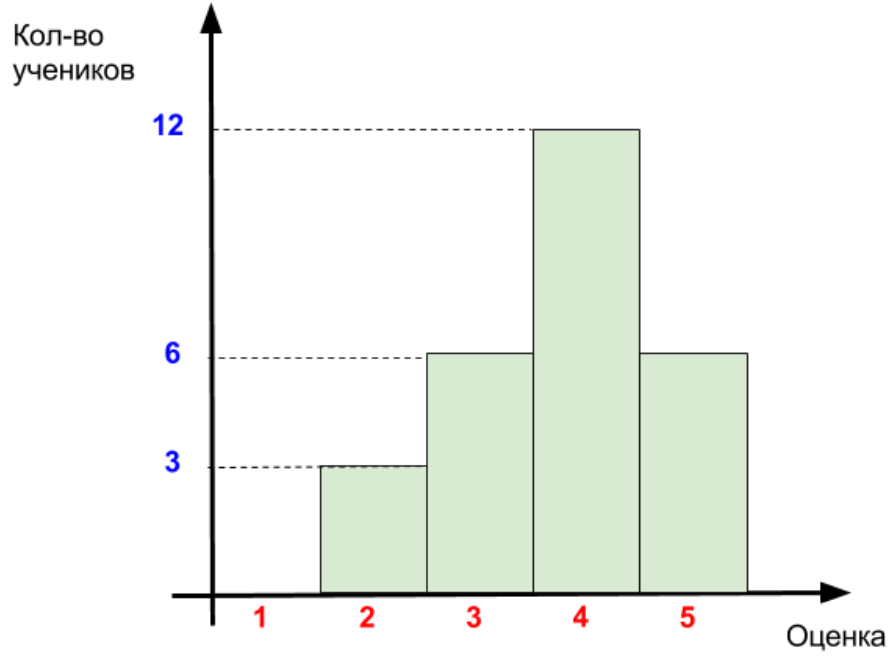
Как вы можете заметить, первый график, который мы сделали просто по ui_request_count не отображает никакой полезной информации, т.к. тип метрики count, и она просто растет. Задание: Используйте для первого графика (UI http requests) функцию rate аналогично второму графику (Rate of UI HTTP Requests with Error)

Гистограмма

Гистограмма представляет собой графический способ представления распределения вероятностей некоторой случайной величины на заданном промежутке значений. Для построения гистограммы берется интервал значений, который может принимать измеряемая величина и разбивается на промежутки (обычно одинаковой величины), данные промежутки помечаются на горизонтальной оси X. Затем над каждым интервалом рисуется прямоугольник, высота которого соответствует числу измерений величины, попадающих в данный интервал.

Гистограмма

Простым примером гистограммы может быть распределение оценок за контрольную в классе, где учится 21 ученик. Берем промежуток возможных значений (от 1 до 5) и разбиваем на равные интервалы. Затем на каждом интервале рисуем столбец, высота которого соответсвует частоте появлению данной оценки.



Histogram метрика

В Prometheus есть тип метрик histogram. Данный тип метрик в качестве своего значение отдает ряд распределения измеряемой величины в заданном интервале значений. Мы используем данный тип метрики для измерения времени обработки HTTP запроса нашим приложением.

Histogram метрика

Рассмотрим пример гистограммы в Prometheus. Посмотрим информацию по времени обработки запроса приходящих на главную страницу приложения.

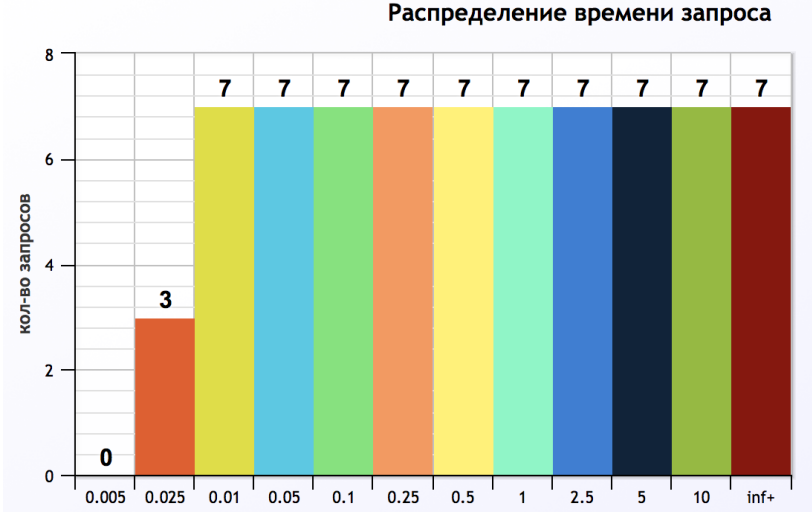
ui_request_latency_seconds_bucket{path="/"}

Graph | Console

Element	Value
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="0.005",path="/"}	0
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="5",path="/"}	7
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="0.5",path="/"}	7
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="1",path="/"}	7
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="10",path="/"}	7
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="0.25",path="/"}	7
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="0.05",path="/"}	7
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="2.5",path="/"}	7
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="0.025",path="/"}	3
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="0.01",path="/"}	0
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="0.1",path="/"}	7
ui_request_latency_seconds_bucket{instance="ui:9292",job="ui",le="+Inf",path="/"}	7

Histogram метрика

Эти значения означают, что запросов с временем обработки <= 0.025s было 3 штуки, а запросов 0.01 <= 0.01s было 7 штук (в этот столбец входят 3 запроса из предыдущего столбца и 4 запроса из промежутка [0.025s; 0.01s], такую гистограмму еще называют кумулятивной). Запросов, которые бы заняли > 0.01s на обработку не было, поэтому величина всех последующих столбцов равна 7.

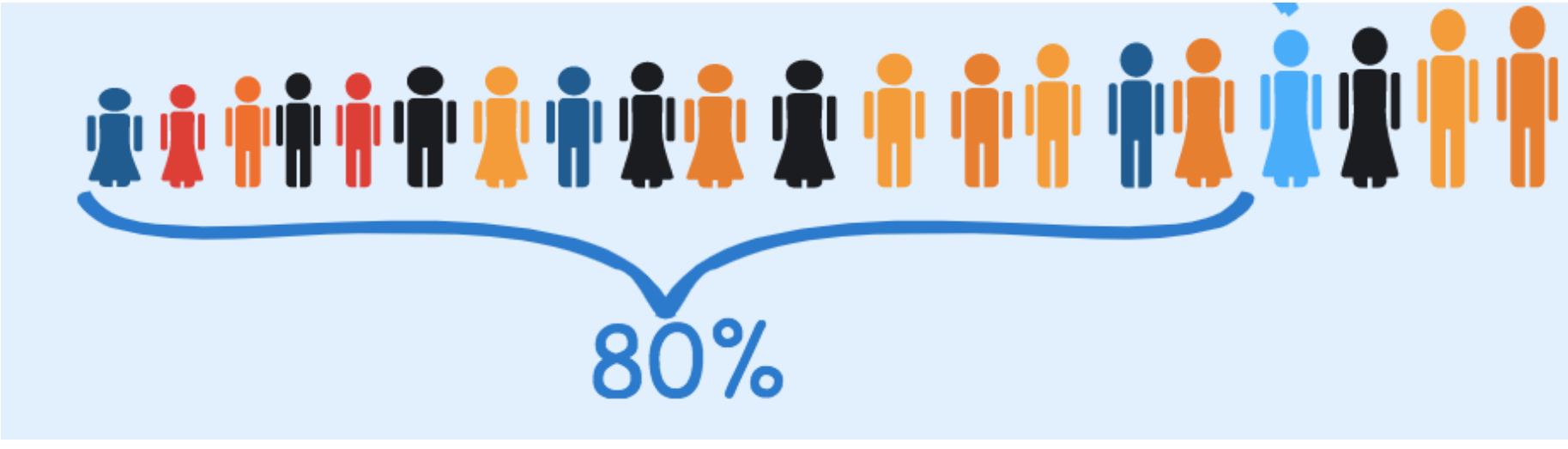


Процентиль

- Числовое значение в наборе значений
- Все числа в наборе меньше процентиля, попадают в границы заданного процента значений от всего числа значений в наборе

Пример процентиля

В классе 20 учеников. Ваня занимает 4-е место по росту в классе. Тогда рост Вани (180 см) является 80-м процентилем. Это означает, что 80 % учеников имеют рост менее 180 см.



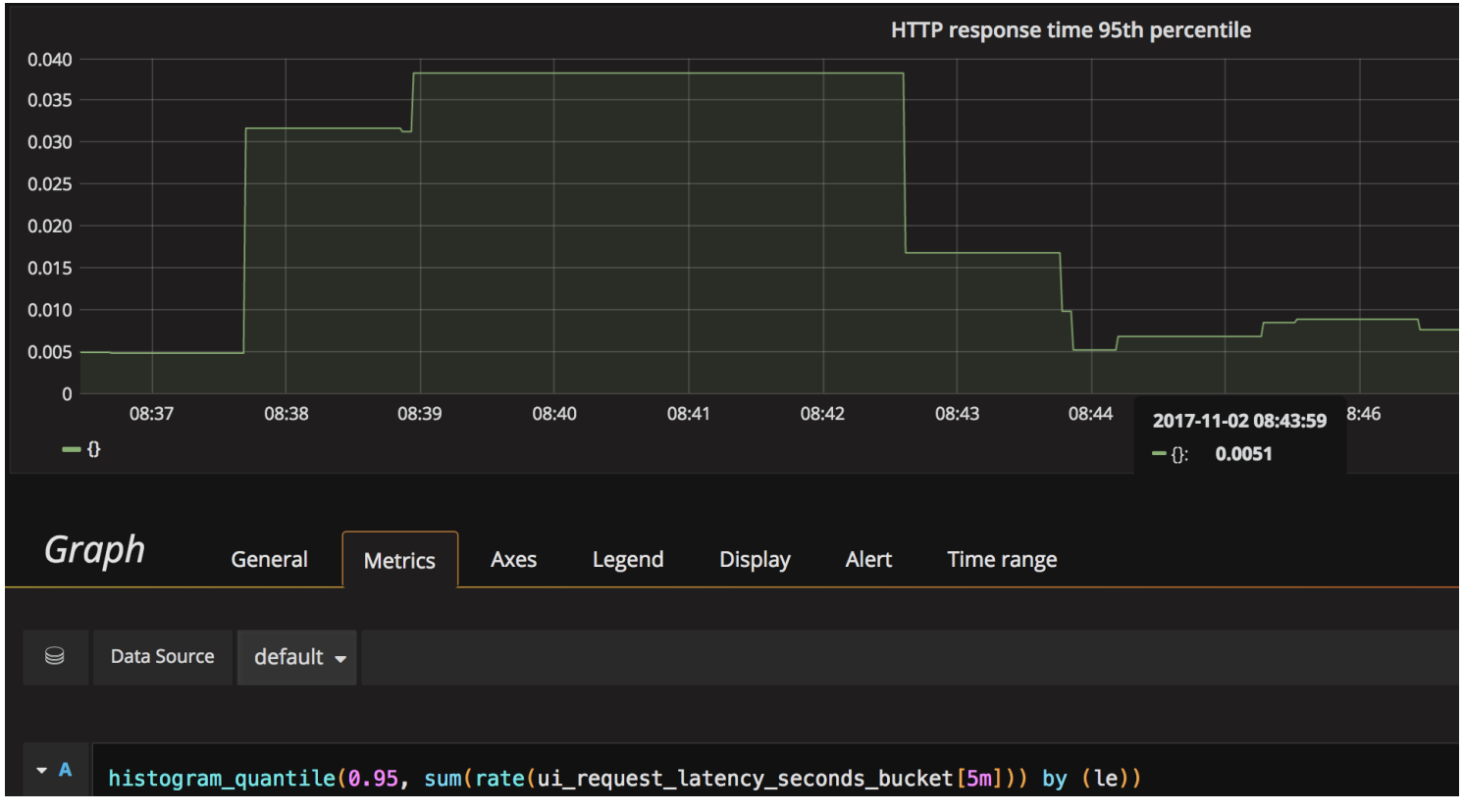
95-й процентиль

Часто для анализа данных мониторинга применяются значения 90, 95 или 99-й процентиля.

Мы вычислим 95-й процентиль для выборки времени обработки запросов, чтобы посмотреть какое значение является максимальной границей для большинства (95%) запросов. Для этого воспользуемся встроенной функцией histogram_quantile():

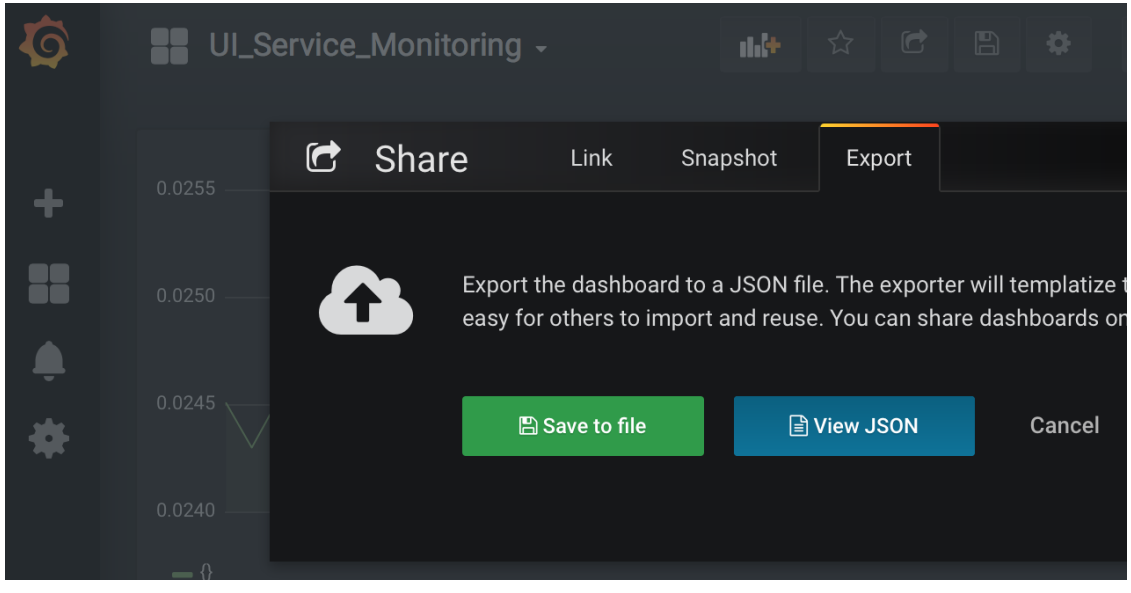
95-й процентиль

Добавьте третий по счету график на ваш дашборд. В поле запроса введите следующее выражение для вычисления 95 процентиля времени ответа на запрос (gist):



95-й процентиль

Сохраним изменения дашборда и эспортируем его в JSON файл, который загрузим на нашу локальную машину



Положите загруженный файл в созданную ранее директорию monitoring/grafana/dashboards под названием UI_Service_Monitoring.json

Сбор метрик бизнес-логики

Мониторинг бизнес-логики

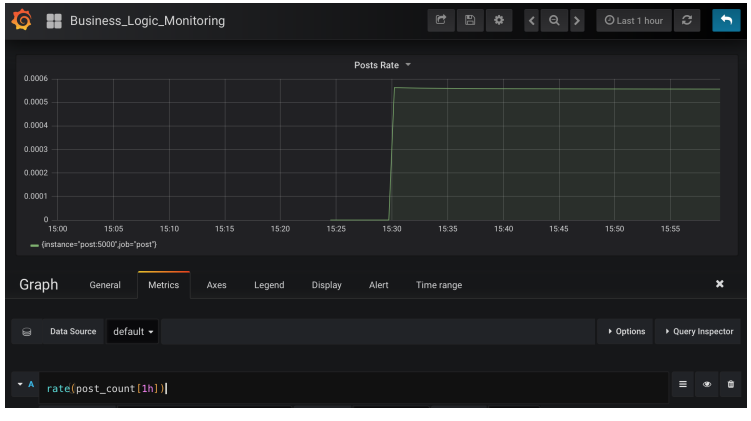
В качестве примера метрик бизнес логики мы в наше приложение мы добавили счетчики **количества постов** и **комментариев**

- post_count
- comment_count

Мы построим график скорости роста значения счетчика за последний час, используя функцию rate(). Это позволит нам получать информацию об активности пользователей приложения.

Мониторинг бизнес-логики

1. Создайте новый дашборд, назовите его Business_Logic_Monitoring и постройте график функции rate(post_count[1h])



2. Постройте еще один график для счетчика comment, экспортируйте дашборд и сохраните в директории monitoring/grafana/dashboards под названием Business_Logic_Monitoring.json.

Алертинг

Правила алертинга

Мы определим несколько правил, в которых зададим условия состояний наблюдаемых систем, при которых мы должны получать оповещения, т.к. заданные условия могут привести к недоступности или неправильной работе нашего приложения.

P.S. Стоит заметить, что в самой Grafana тоже есть alerting. Но по функционалу он уступает Alertmanager в Prometheus.

Alertmanager

Alertmanager - дополнительный компонент для системы мониторинга Prometheus, который отвечает за первичную обработку алертов и дальнейшую отправку оповещений по заданному назначению.

Создайте новую директорию monitoring/alertmanager. В этой директории создайте Dockerfile со следующим содержимым:

```
FROM prom/alertmanager:v0.14.0

ADD config.yml /etc/alertmanager/
```

Alertmanager

Настройки Alertmanager-а как и Prometheus задаются через YAML файл или опции командой строки. В директории monitoring/alertmanager создайте файл config.yml, в котором определите отправку нотификаций в ВАШ тестовый слак канал. Для отправки нотификаций в слак канал потребуется создать СВОЙ Incoming Webhook **monitoring/alertmanager/config.yml**
ссылка на gist

```
global:
  slack_api_url:
'https://hooks.slack.com/services/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'
#Заменяем на свои значения

route:
  receiver: 'slack-notifications'

receivers:
- name: 'slack-notifications'
  slack_configs:
  - channel: '#userchannel' #Заменяем на свои значения
```

Alertmanager

1. Соберем образ alertmanager:

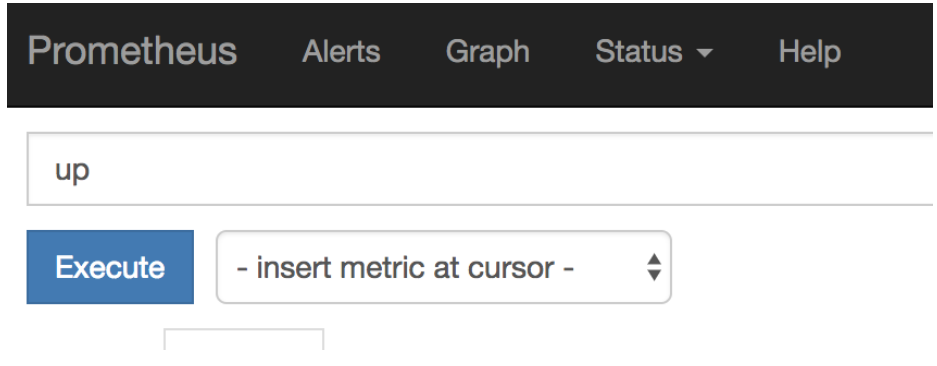
```
monitoring/alertmanager $ docker build -t $USER_NAME/alertmanager .
```

2. Добавим новый сервис в компоуз файл мониторинга. Не забудьте добавить его в одну сеть с сервисом Prometheus:

```
services:
...
 alertmanager:
    image: ${USER_NAME}/alertmanager
    command:
      - '--config.file=/etc/alertmanager/config.yml'
    ports:
      - 9093:9093
```

Alert rules

Создадим файл alerts.yml в директории prometheus, в котором определим условия при которых должен срабатывать алерт и посылаться Alertmanager-у. Мы создадим простой алерт, который будет срабатывать в ситуации, когда одна из наблюдаемых систем (endpoint) недоступна для сбора метрик (в этом случае метрика up с лейблом instance равным имени данного эндпоинта будет равна нулю). Выполните запрос по имени метрики up в веб интерфейсе Prometheus, чтобы убедиться, что сейчас все эндпоинты доступны для сбора метрик:



Alert rules

monitoring/prometheus/alerts.yml ссылка на gist

```
groups:
  - name: alert.rules
    rules:
    - alert: InstanceDown
      expr: up == 0          # любое PromQL выражение
        for: 1m              # В течении какого времени, по умолчанию 0
        labels:              # Дополнительные метки
          severity: page
      annotations:
        description: '{{ $labels.instance }} of job {{ $labels.job }} has been
down for more than 1 minute'
        summary: 'Instance {{ $labels.instance }} down'
```

Alert rules

Добавим операцию копирования данного файла в Dockerfile: monitoring/prometheus/Dockerfile

```
FROM prom/prometheus:v2.1.0

ADD prometheus.yml /etc/prometheus/

ADD alerts.yml /etc/prometheus/
```

prometheus.yml

Добавим информацию о правилах, в конфиг Prometheus
ссылка на gist

```
global:
  scrape_interval: '5s'
...

rule_files:
  - "alerts.yml"

alerting:
  alertmanagers:
  - scheme: http
    static_configs:
    - targets:
      - "alertmanager:9093"
```

Пересоберем образ Prometheus:

```
$ docker build -t $USER_NAME/prometheus .
```

Проверка алерта

Пересоздадим нашу Docker инфраструктуру мониторинга:

```
$ docker-compose -f docker-compose-monitoring.yml down
$ docker-compose -f docker-compose-monitoring.yml up -d
```

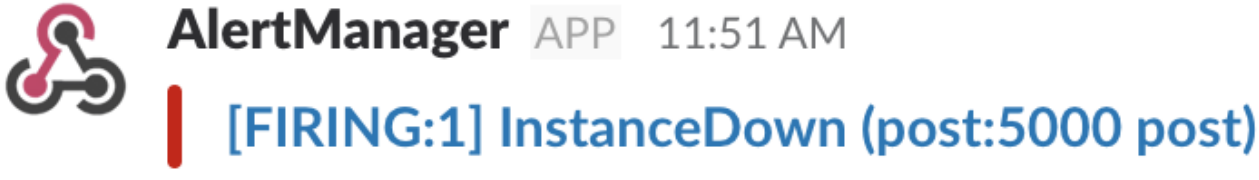
Алерты можно посмотреть в веб интерфейсе Prometheus:

Проверка алерта

Остановим один из сервисов и подождем одну минуту

```
$ docker-compose stop post
```

В канал должно придти сообщение:



У Alertmanager также есть свой веб интерфейс, доступный на порту 9093, который мы прописали в компоуз файле.

P.S. Проверить работу вебхуков слака можно обычным curl.

Завершение работы

- Запушьте собранные вами образы на DockerHub:

```
$ docker login
Login Succeeded

$ docker push $USER_NAME/ui
$ docker push $USER_NAME/comment
$ docker push $USER_NAME/post
$ docker push $USER_NAME/prometheus
$ docker push $USER_NAME/alertmanager
```

- Удалите виртуалку:

```
$ docker-machine rm docker-host
```

Добавьте ссылку на докер хаб с вашими образами в README.md и описание PR

Задания со *

- Если в прошлом ДЗ вы реализовали Makefile, добавьте в него билд и публикацию добавленных в этом ДЗ сервисов;
- В Docker в экспериментальном режиме реализована отдача метрик в формате Prometheus. Добавьте сбор этих метрик в Prometheus. Сравните количество метрик с Cadvisor. Выберите готовый дашборд или создайте свой для этого источника данных. Выгрузите его в monitoring/grafana/dashboards;
- Для сбора метрик с Docker демона также можно использовать Telegraf от InfluxDB. Добавьте сбор этих метрик в Prometheus. Сравните количество метрик с Cadvisor. Выберите готовый дашборд или создайте свой для этого источника данных. Выгрузите его в monitoring/grafana/dashboards;
- Придумайте и реализуйте другие алерты, например на 95 процентиль времени ответа UI, который рассмотрен выше; Настройте интеграцию Alertmanager с e-mail помимо слака;

Задания с **

- В Grafana 5.0 была добавлена возможность описать в конфигурационных файлах источники данных и дашборды. Реализуйте автоматическое добавление источника данных и созданных в данном ДЗ дашбордов в графану;
- Реализуйте сбор метрик со Stackdriver, в PR опишите, какие метрики удалось собрать;
- Придумайте свои метрики приложения/бизнес метрики и реализуйте их в коде приложения. Опишите в PR что было добавлено;

Задания со ***

- Реализуйте схему с проксированием запросов от Grafana к Prometheus через Trickster, кеширующий прокси от Comcast;
- Используя связку Autoheal + AWX, реализуйте автоматическое исправление проблем (например рестарт одного из микросервисов при падении);

  - Autoheal - проект команды OpenShift для автоматического исправления проблем по результатам алертов;
  - AWX - open source версия Ansible Tower, установить его можно либо вручную, либо используя одну из готовых ролей, например;

Дополнительные папки создавайте в директории monitoring

Проверка ДЗ

- Результаты вашей работы находятся в ветке monitoring-2 вашего microservices репозитория.
- В README внесите описание того, что сделано
- Создайте Pull Request к ветке master (описание PR нужно заполнять);
- В ревьюверы можно никого не добавлять;
- Добавьте "Labels" monitoring и monitoring-2 к вашему Pull Request;
- После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить и закрыть PR.