GeekBrains

Мониторинг распределённых приложений и сервисов

Логи и как с ними работать на примере ELK и Sentry

На этом уроке

1. Разберём, что такое логи и как с ними работать.
2. Познакомимся с двумя подходами к обработке ошибок.

Оглавление

Логирование

Elastic Stack

- Архитектура
- EFK
- Kibana
- Elasticsearch

Sentry

- Обзор и запуск
- Первый проект

Дополнительные материалы

Используемые источники

Логирование

Логи — важная часть данных, которая содержит информацию о работе системы, сетевого оборудования и, конечно же, наших приложений и сервисов. Изучая логи, можно разобраться в причинах сбоев, отследить какие-то аномалии в работе, действия пользователей и т. д.

В нашей системе всё от событий ядра до действий пользователя и состояния приложений отслеживается и обычно записывается в специальную директорию /var/log. Здесь содержится множество файлов с логами, в том числе:

- /var/log/syslog или /var/log/messages — глобальный системный журнал;
- /var/log/auth.log или /var/log/secure — информация об авторизации пользователей;
- /var/log/dmesg — драйверы устройств.

Обычно собираемые логи имеют определённый формат:

- timestamp;
- имя приложения, генерирующего событие;
- место, откуда было отправлено сообщение;
- приоритет (комбинация двух параметров: северити + категория);
- само сообщение.

Северити (severity, критичность/серьёзность) делятся на следующие группы:

- `emerg` — система не работает (PANIC);
- `alert` — серьёзная проблема;
- `crit` — критическая ошибка;
- `err` — ошибка (ERROR);
- `warn` — предупреждение (WARN);
- `notice` — всё нормально, но сообщение важное;
- `info` — информационные сообщения;
- `debug` — отладочные сообщения.

Некоторые возможные категории логов (facility):

0	`kern`	Сообщения, отправляемые ядром
1	`user`	Пользовательские программы
2	`mail`	Почта
3	`daemon`	Сервисы (демоны)
4	`auth`	Безопасность / вход в систему / аутентификация

5	`syslog`	Сообщения от syslog
8	`uucp`	Unix-to-Unix CoPy (копирование файлов между компьютерами)
9	`cron`	Планировщик заданий
10	`authpriv`	Безопасность / вход в систему / аутентификация — защищённый режим

В наших системах за сбор и распределение логов отвечает демон `rsyslog`, который конфигурируется файлом /etc/rsyslog.conf. В этом файле можно очень гибко задать фильтрацию логов по приоритету и подсистемам и затем отправить их на удалённый сервер или записать в файл. Например:

```
auth.info @@10.10.10.10:1514  #отправит все логи, связанные с категорией auth, на сервер 10.10.10.10
kern.warn /var/log/kern.log #будет записывать логи ядра только с уровнем warning и выше
```

Генерирующиеся логи надо где-то хранить и обрабатывать. Дальше мы рассмотрим два подхода:

- система централизованного хранения логов (на примере EFK);
- система обработки ошибок (на примере Sentry).

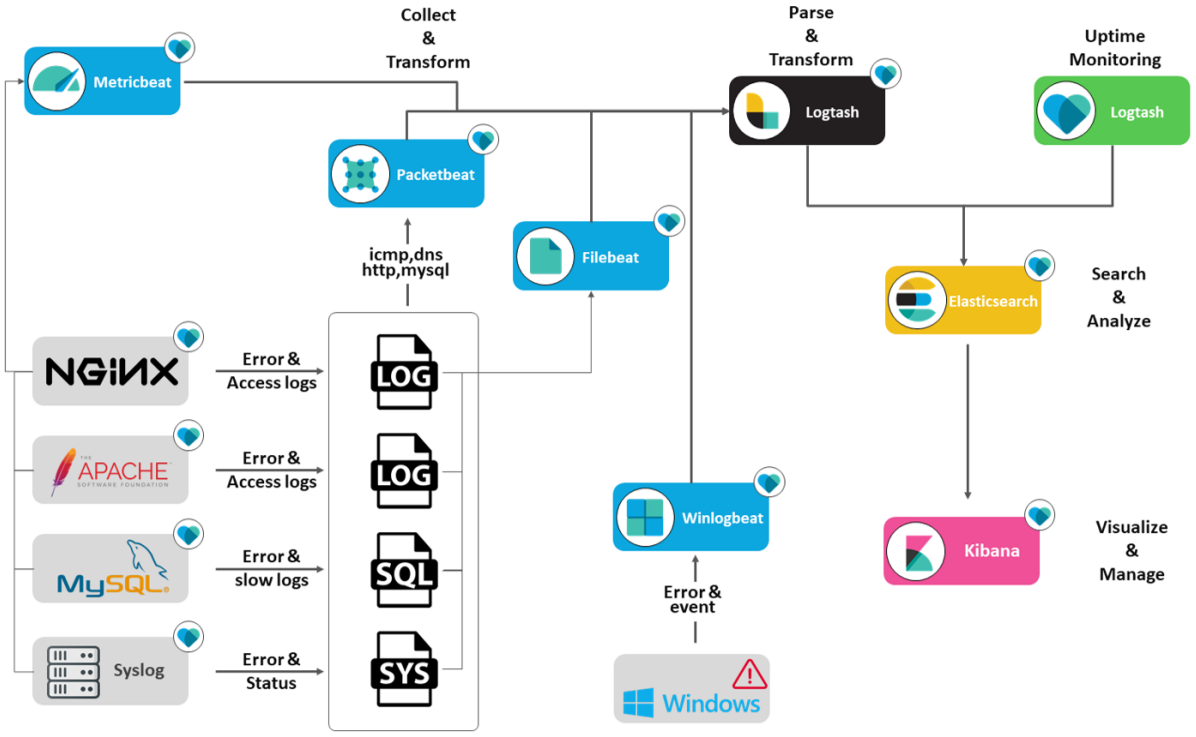
Elastic Stack

Архитектура

Elastic Stack — набор программного обеспечения для сбора, поиска, анализа и визуализации логов, которые были сгенерированы каким-либо источником. Централизованное решение для хранения логов не отменяет необходимости хранить логи локально хотя бы небольшой промежуток времени.

Классическая архитектура Elastic Stack имеет следующий вид:

1. Источник логов.
2. Beats — сборщики логов, могут отправить данные в Logstash или Elasticsearch.
3. Logstash обрабатывает логи и складывает в Elasticsearch.
4. Elasticsearch — распределённая поисковая система, которая хранит данные.
5. Kibana — веб-интерфейс для визуализации данных.



Как уже было сказано, Beats — это сборщики логов, устанавливаемые на клиентских устройствах. Существует большой выбор коллекторов, среди которых:

- **Filebeat** — собирает информацию из журналов системы и отправляет её на сервер;
- **Winlogbeat** — аналог Filebeat, только для Windows-систем;
- **Packetbeat** — сборщик для сетевого взаимодействия;
- **Metricbeat** — сборщик метрик с системы (по действиям аналогичен мониторинг-агентам);
- **Auditbeat** — сборщик аудит-данных Linux-систем;
- **Heartbeat** — проверяет доступность и время отклика сервисов.

С популяризацией платформы Kubernetes всё большую распространённость приобретает EFK:

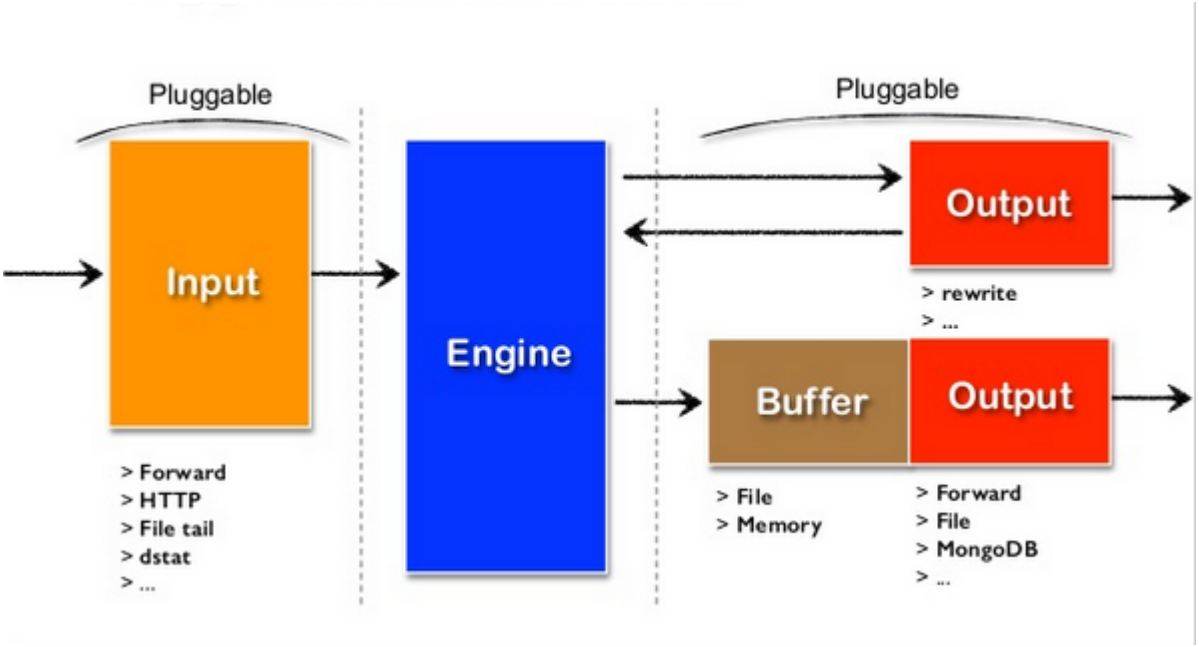
- Elasticsearch;
- Fluentd;
- Kibana.

Fluentd — Cloud-Native-коллектор логов с открытым исходным кодом.

Особенности Fluentd:

1. Низкое потребление ресурсов.
2. Унифицированные данные (все полученные логи Fluentd переводит в формат JSON).

3. Архитектура Fluentd позволяет расширять набор функций с помощью многочисленных плагинов.



EFK

Запустим наш Elastic Stack (развернём EFK). Для этого:

1. В вашей рабочей директории создадим docker-compose.yaml со следующим содержимым:

```
version: '3.7'
services:
  nginx:
    image: nginx
    container_name: nginx
    ports:
      - 80:80
    logging:
      driver: "fluentd"
      options:
        fluentd-async-connect: "true"
        fluentd-address: localhost:24224
        tag: fluentd
    networks:
      - elasticsearch

  fluentd:
    build: ./fluentd
    container_name: fluentd
    volumes:
      - ./fluentd/conf:/fluentd/etc
    ports:
      - "24224:24224"
      - "24224:24224/udp"
```

```
    networks:
      - elasticsearch

  apm-server:
    image: docker.elastic.co/apm/apm-server:7.9.2
    container_name: apm-server
    volumes:
      -
./apm_elastic/apm-server.docker.yml:/usr/share/apm-server/apm-server.yml:r
o
    environment:
      - output.elasticsearch.hosts=["elasticsearch:9200"]
    ports:
      - "8200:8200"
    networks:
      - elasticsearch

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.9.2
    container_name: elasticsearch
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
    volumes:
      - ./elasticsearch:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
    networks:
      - elasticsearch

  kibana:
    image: kibana:7.9.2
    container_name: kibana
    environment:
      ELASTICSEARCH_URL: http://elasticsearch:9200
      ELASTICSEARCH_HOSTS: http://elasticsearch:9200
    volumes:
      - ./kibana.yml:/usr/share/kibana/config/kibana.yml
    ports:
      - "5601:5601"
    networks:
      - elasticsearch

networks:
    elasticsearch:
```

2. Затем подготовим fluentd/Dockerfile для Fluentd с дополнительным плагином для Elasticsearch:

```
# fluentd/Dockerfile
FROM fluent/fluentd:v1.11-debian-1
USER root
```

```
RUN ["gem", "install", "fluent-plugin-elasticsearch", "--no-document",
"--version", "4.3.2"]
# USER fluent
```

3. И также подготовим конфигурацию для нашего Fluentd:

```
# fluentd/conf/fluent.conf
<source>
  @type forward
  port 24224
  bind 0.0.0.0
</source>
<filter **>
  @type stdout
</filter>
<match *.**>
  @type copy
  <store>
    @type elasticsearch
    host elasticsearch
    port 9200
    logstash_format true
    logstash_prefix fluentd
    logstash_dateformat %Y%m%d
    include_tag_key true
    type_name access_log
    tag_key @log_name
    flush_interval 1s
  </store>
  <store>
    @type stdout
  </store>
</match>
```

4. Создадим базовый конфигурационный файл для Kibana:

```
# kibana.yml
server.name: kibana
server.host: "0"
elasticsearch.hosts: [ "http://elasticsearch:9200" ]
xpack.reporting.capture.browser.chromium.disableSandbox: false
```

Запустим нашу систему и проверим, что всё работает:

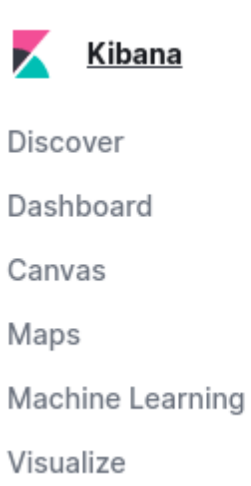
```
docker-compose up -d
docker ps
```

Внимание! *Возможно, для корректной работы потребуется установить плагин для Docker:* ***docker plugin install --alias fluentd-async akerouanton/fluentd-async-logger:v0.3.***

Kibana

Когда все контейнеры стартанули, мы можем перейти непосредственно к работе с логами. Откроем Kibana в браузере (по умолчанию порт 5601) и первым делом зададим индекс, который нас интересует, настроив шаблон. Укажем fluentd* в качестве шаблона и выберем @timestamp для Time field. С этого момента мы можем изучать и анализировать наши данные. Также доступны подготовленные примеры для ознакомления с возможностями Kibana.

Пройдёмся по некоторым пунктам меню. В левом верхнем углу кнопка ☰ открывает панель навигации по Kibana. Здесь есть несколько основных групп, среди которых в этом уроке нас будут интересовать две: **Kibana** и **Management**.



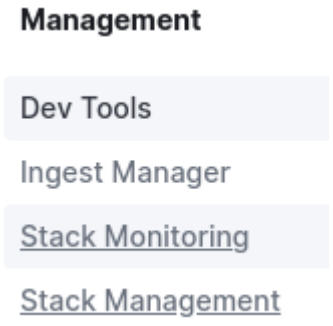
Рассмотрим пару вкладок в группе Kibana и начнём с **Discover**. Здесь происходит поиск данных. Указав Kibana, где найти данные, мы можем выполнить поиск, применяя различные фильтры:

- выберите шаблон для индекса;
- установите временной интервал поиска;
- с помощью Kibana Query Language (KQL) отфильтруйте данные;
- выберите только те поля из доступных, которые вам нужны;
- создайте на основе своей выборки новую визуализацию.

Во вкладке **Visualize** собраны все типы визуализации. Для создания новой визуализации необходимо сделать следующее:

- выбрать один из типов;
- указать источник;
- указать поисковый запрос;
- в конструкторе можно указать метрику для агрегации и саму агрегацию.

Теперь рассмотрим группу Management. Здесь очень интересны две вкладки: **Dev Tools** и **Stack Management**.



Dev Tools **Console** позволяет взаимодействовать с API Elasticsearch напрямую. Можно управлять индексами, получать и отправлять данные.

Dev Tools **Search Profiler** помогает в разработке и анализе ваших запросов.

Dev Tools **Grok Debugger** позволяет отлаживать Grok-шаблоны перед использованием в конвейерах.

Stack Management — здесь вы можете управлять всем, что связано с Elastic Stack, в том числе индексами, лицензиями, настройками интерфейса, отчётами и т. д. Здесь можно настроить отдельные спейсы (рабочие пространства) для команд и подключить удалённые кластеры Elasticsearch.

Elasticsearch

Elasticsearch — распределённая поисковая и аналитическая система, основа Elastic Stack. Она обеспечивает поиск и анализ всех типов данных в реальном времени, прекрасно масштабируется и, соответственно, может работать с огромным количеством информации.

Данные в Elasticsearch делятся на индексы (indices), состоящие из одного или нескольких сегментов (shards).

Как мы можем видеть, в Kibana наши indices имеют состояние yellow. Это связано с тем, что мы поднимали однонодовый кластер, а по умолчанию количество реплик для каждого индекса = 1, но Elasticsearch не будет записывать два одинаковых shard на одну ноду. Есть несколько вариантов развития событий:

- ничего не делать и смириться со статусом, но в этом случае есть большая вероятность пропустить какую-нибудь проблему;
- поменять для каждого индекса количество реплик на 0, используя API Elasticsearch;
- настроить шаблон для новых индексов с количеством реплик по умолчанию = 0;
- также можно прокинуть в Docker предварительно подготовленную конфигурацию Elasticsearch.

Немного примеров команд для работы с Elasticsearch через консоль:

```
# получить информацию о состоянии кластера
curl -XGET localhost:9200/_cluster/health
 # получить все индексы
curl -XGET localhost:9200/_cat/indices

 # получить свойства индекса
curl -XGET localhost:9200/apm-7.9.2-onboarding-2021.01.26/_settings

# изменить количество реплик в определённом индексе
curl -XPUT -H "Content-Type: application/json" -d '{"index" :
{"number_of_replicas" : 0 }}'
localhost:9200/apm-7.9.2-onboarding-2021.01.26/_settings
```

И в заключение скажем пару слов о сопоставлении (mapping). Mapping — это процесс схемы, описывающий свойства полей в данных. При индексации для каждого поля Elasticsearch динамически

определяет его тип, но также можно задать тип данных вручную. Есть возможность задавать несколько типов для одного поля.

Sentry

Обзор и запуск

Sentry — сервис для мониторинга событий, позволяющий отслеживать и устранять сбои в режиме реального времени. Основное отличие от систем логирования заключается в том, что Sentry обрабатывает исключения, то есть сбои в работе приложения. Есть два пути для начала использования Sentry:

- оформить подписку на официальном сайте и использовать их ресурсы (есть даже бесплатный вариант);
- запустить Sentry на своём «железе».

Архитектуру можно посмотреть в документации на официальном сайте. Если в двух словах, в каждое наше приложение встраивается кусок кода, отвечающий за взаимодействие с Sentry-сервером, который отлавливает ошибки как на стороне клиента, так и на стороне сервера.

Для Standalone-запуска можно использовать Docker Compose, который содержит все зависимости:

```
# скачиваем репозиторий с Sentry
git clone https://github.com/getsentry/onpremise.git
# переходим в скачанный репозиторий
cd onpremise
# запускаем предварительную настройку
./install.sh

## в процессе установки будет запрос на создание аккаунта для вашего сервера

# запускаем Sentry
docker-compose up -d
```

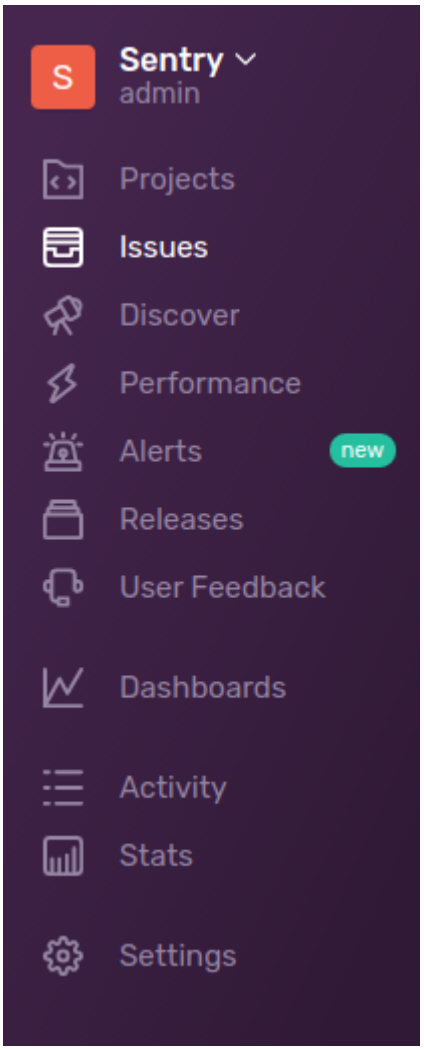
Внимание! *Для запуска Sentry со всеми зависимостями рекомендуется выделить не менее 2.4 Гб оперативной памяти.*

После запуска всех контейнеров мы можем ознакомиться с интерфейсом по адресу **localhost:9000**.

Первый проект

Залогинившись, используя учётные данные при конфигурации, мы попадаем на основной дашборд. С левой стороны мы видим навигационную панель. Пройдёмся по порядку:

1. **Projects** — здесь мы создаём наши проекты, выбираем платформу и можем задать базовые настройки. При нажатии кнопки Create появляется подсказка об интеграции SDK с выбранной платформой.
2. **Issues** — в этой вкладке мы можем посмотреть все отловленные баги с каждого проекта по отдельности или со всех одновременно.
3. **Discover** — здесь мы можем создать выборки из наших проектов и посмотреть статистику.
4. **Performance** — мониторинг производительности нашего приложения, который надо настроить.
5. **Alerts** — создание алертов на события.
6. **Releases** — информация о релизах.
7. **User Feedback** — возможность получать обратную связь от пользователей.
8. **Dashboards** — наглядная информация о месте и количестве багов, сколько из них обработано, сколько пользователей пострадали.
9. **Activity** — информация об обработанных ошибках.
10. **Stats** — обобщённая статистика со всех проектов.
11. **Settings** — настройки нашего Sentry.



Создадим наш первый проект:

1. Перейдём на вкладку Projects и нажмём кнопку Create project.
2. Выберем проект на Python или Flask.
3. Чтобы посмотреть, как всё работает, запустим пример, представленный в репозитории, предварительно установив все зависимости:

```
pip install flask
pip install raven
pip install --upgrade 'sentry-sdk[flask]'
vim my_test_flask_app.py
```

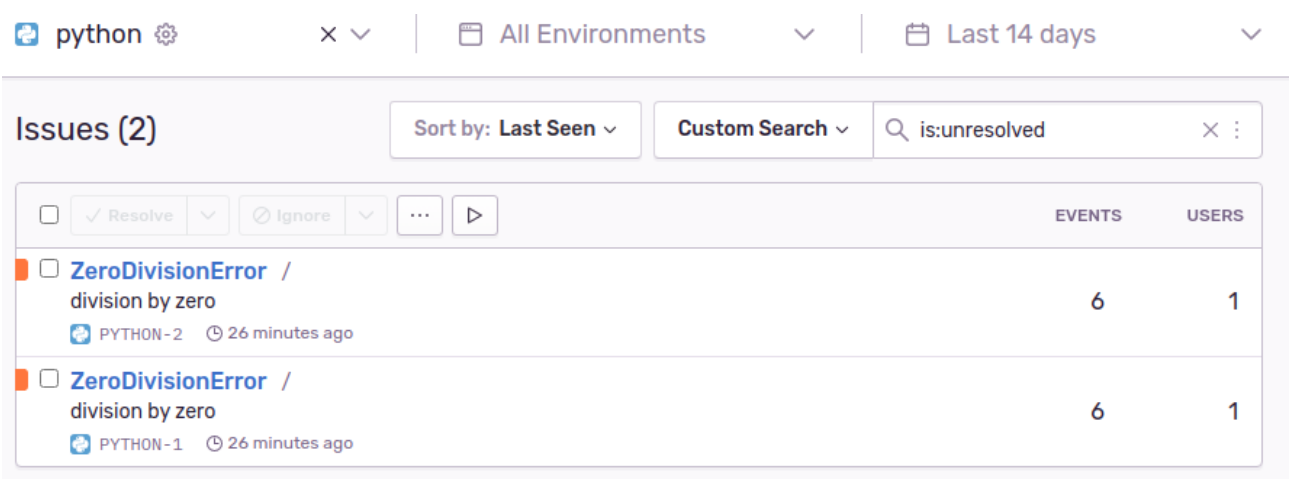
```
## my_test_flask_app.py

import sentry_sdk
```

```
from flask import Flask
from flask import Flask, render_template
from raven.contrib.flask import Sentry
from sentry_sdk.integrations.flask import FlaskIntegration

# дополнительная секция для мониторинга производительности
sentry_sdk.init(
   dsn='http://0d3f29a5798c4b1cb0f495fc1911e1bf@localhost:9000/5',
   integrations=[FlaskIntegration()],


   traces_sample_rate=1.0
)
app = Flask(__name__)

# секция для отправки ошибок в Sentry
sentry = Sentry(app,
dsn='http://0d3f29a5798c4b1cb0f495fc1911e1bf@localhost:9000/5')

@app.route('/')
def hello_error():

   1 / 0 #ZeroDivisionError to be sent to sentry
   return render_template("hello_world.html")

if __name__ == '__main__':
   app.run(debug=True, host='0.0.0.0', port='4567')
```

```
python3 my_test_flask_app.py
```

4. После запуска приложения на вкладке Issues нашего проекта мы можем обнаружить две новые проблемы.

   Внимание! *Если проблема будет повторяться (например, можно обновить страницу localhost:4567 несколько раз), то дублирование проблемы не произойдёт. Вместо этого будет увеличиваться количество событий для этой проблемы.*
5. Также во вкладке Performance есть информация о мониторинге производительности нашего приложения.
6. В качестве примера ещё можно создать Alerts. Их можно задавать как на основе метрик, так и на основе событий. Alerts → Create Alert Rule → <your project> → Metric Alert OR Issue Alert → задаём имя, условия → Create Alert Rule.



Практическое задание

1. Elastic Stack:
    - запустить Elastic Stack и несколько сервисов для генерации логов;
    - каждый сервис должен писать логи в отдельный индекс;
    - настроить ротацию логов в 1 день;
    - *настроить с помощью Prometheus сбор метрик с EFK

      или

      *настроить отправку метрик с Prometheus в Elastic Stack.
2. Sentry:
    - запустить Sentry и приложение, умеющее с ним работать;
    - *настроить интеграцию с GitHub и Slack.

Дополнительные материалы

1. Распределённое логирование и трассировка для микросервисов.
2. Observability: как наблюдать за системой.
3. Ultimate Guide to Logging.
4. Лог-файлы Linux по порядку.
5. Elastic APM в приложении.
6. Sentry — трекинг Java Exception в Java.
7. Мониторинг ошибок с помощью Sentry во фронтенд-приложениях, написанных на JavaScript. Часть 1.
8. Running the Elastic Stack on Docker.
9. Сбор и анализ логов с Fluentd.
10. Collect Logs with Fluentd in K8s (Part 2).
11. Elastic Beats.
12. 5 Logstash Alternatives.

13. EFK (Elasticsearch + Fluentd + Kibana) или как мы логи собираем.
14. Установка Elasticsearch, Logstash и Kibana (комплекс Elastic) в Ubuntu 18.04.
15. Improve Your Workflow with Sentry.
16. Сборка sentry и его зависимостей в rpm. Установка Sentry из rpm, базовая настройка. Подключение к LDAP.

Используемые источники

1. 12 Critical Linux Log Files You Must be Monitoring.
2. Elastic Stack and Product Documentation.
3. Elastic APM: Application Performance Monitoring with the ELK Stack & Logz.io.
4. Sentry Documentation.
5. Flask Example for getsentry.
6. Docker Compose.
7. Что такое Sentry и почему без него тяжело при разработке веб-проекта.
8. Sentry vs Logging.