Логирование и распределенная трассировка

Проект microservices и проверка ДЗ

Создайте новую ветку в вашем microservices репозитории для выполнения данного ДЗ. Это первое задание про мониторинг, ветку назовите **logging-1**.

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

- Сбор неструктурированных логов
- Визуализация логов
- Сбор структурированных логов
- Распределенная трасировка

Подготовка

Код микросервисов обновился для добавления функционала логирования. Новая версия кода доступа по ссылке.

- Обновите код в директории **/src** вашего репозитория из кода по ссылке выше.
- Если вы используется python-alpine, добавьте в **/src/post-py/Dockerfile** установку пакетов gcc и musl-dev
- Выполните сборку образов при помощи скриптов docker_build.sh в директории каждого сервиса:

```
/src/ui      $ bash docker_build.sh && docker push $USER_NAME/ui
/src/post-py $ bash docker_build.sh && docker push $USER_NAME/post
/src/comment $ bash docker_build.sh && docker push $USER_NAME/comment
```

Или сразу все из корня репозитория:

```
for i in ui post-py comment; do cd src/$i; bash docker_build.sh; cd -; done
```

Внимание! В данном ДЗ мы используем отдельные теги для контейнеров приложений :logging

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):

```
$ 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-open-port 5601/tcp \
    --google-open-port 9292/tcp \
    --google-open-port 9411/tcp \
    logging
...
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 logging

$ eval $(docker-machine env logging)

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

Логирование Docker контейнеров

Elastic Stack

Как упоминалось на лекции хранить все логи стоит **централизованно**: на одном (нескольких) серверах. В этом ДЗ мы рассмотрим пример системы централизованного логирования на примере Elastic стека (ранее известного как ELK): который включает в себя 3 осовных компонента:

- ElasticSearch (TSDB и поисковый движок для хранения данных)
- Logstash (для агрегации и трансформации данных)
- Kibana (для визуализации)

Однако для агрегации логов вместо Logstash мы будем использовать Fluentd, таким образом получая еще одно популярное сочетание этих инструментов, получившее название EFK

docker-compose-logging.yml

Создадим отдельный compose-файл для нашей системы логирования в папке **docker/** (gist)

docker/docker-compose-logging.yml

```
version: '3'
services:
  fluentd:
    build: ./fluentd
    ports:
      - "24224:24224"
      - "24224:24224/udp"

  elasticsearch:
    image: elasticsearch
    expose:
      - 9200
    ports:
      - "9200:9200"

  kibana:
    image: kibana
    ports:
      - "5601:5601"
```

Fluentd

Fluentd инструмент, который может использоваться для отправки, агрегации и преобразования лог-сообщений. Мы будем использовать Fluentd для агрегации (сбора в одной месте) и парсинга логов сервисов нашего приложения.

Создадим образ Fluentd с нужной нам конфигурацией.

Создайте в вашем проекте microservices директорию **logging/fluentd**

В созданной директорий, создайте простой **Dockerfile** со следущим содержимым:

```
FROM fluent/fluentd:v0.12
RUN gem install fluent-plugin-elasticsearch --no-rdoc --no-ri --version 1.9.5
RUN gem install fluent-plugin-grok-parser --no-rdoc --no-ri --version 1.0.0
ADD fluent.conf /fluentd/etc
```

fluent.conf

В директории logging/fluentd создайте файл конфигурации:

logging/fluentd/fluent.conf (ссылка на gist)

```
  <source>
  @type forward  # Используем in_forward плагин для приема логов
                 # https://docs.fluentd.org/v0.12/articles/in_forward
  port 24224
  bind 0.0.0.0
</source>

<match *.**>
  @type copy     # Используем copy плагин, чтобы переправить все входящие логи в ElasticSearch, а также вывести в output
                 # https://docs.fluentd.org/v0.12/articles/out_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>
```

Fluentd

Соберите docker image для fluentd

Из директории logging/fluentd

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

Структурированные логи

Логи должны иметь заданную (единую) структуру и содержать необходимую для нормальной эксплуатации данного сервиса информацию о его работе

Лог-сообщения также должны иметь понятный для выбранной системы логирования формат, чтобы избежать ненужной траты ресурсов на преобразование данных в нужный вид. Структурированные логи мы рассмотрим на примере сервиса post

Структурированные логи

Правим .env файл и меняем теги нашего приложения на logging

Запустите сервисы приложения

```
docker/ $ docker-compose up -d
```

И выполните команду для просмотра логов post сервиса:

```
docker/ $ docker-compose logs -f post
Attaching to reddit_post_1
```

⚠️ Внимание! Среди логов можно наблюдать проблемы с доступностью Zipkin, у нас он пока что и правда не установлен. Ошибки можно игнорировать. (Github issue)

Откройте приложение в браузере и создайте несколько постов, пронаблюдайте, как пишутся логи post серсиса в терминале

```
docker/ $ docker-compose logs -f post
Attaching to reddit_post_1

post_1        | {"event": "find_all_posts", "level": "info", "message": "Successfully retrieved all posts from the
database", "params": {}, "request_id": "17501ae3-3d4f-4fe6-ac99-ca7cb58492a9", "service": "post", "timestamp": "2017-12-
10 23:36:59"}

post_1        | {"addr": "172.21.0.7", "event": "request", "level": "info", "method": "GET", "path": "/posts?",
"request_id": "17501ae3-3d4f-4fe6-ac99-ca7cb58492a9", "response_status": 200, "service": "post", "timestamp": "2017-12-
10 23:36:59"}

post_1        | {"event": "post_create", "level": "info", "message": "Successfully created a new post", "params":
{"link": "https://github.com/hynek/structlog", "title": "Structlog is awesome! "}, "request_id": "2aaf1ad3-42cf-4105-
b585-d990eb22d85b", "service": "post", "timestamp": "2017-12-10 23:37:18"}
```

Каждое событие, связанное с работой нашего приложения логируется в JSON формате и имеет нужную нам структуру: тип события (event), сообщение (message), переданные функции параметры (params), имя сервиса (service) и др.

Отправка логов во Fluentd

Как отмечалось на лекции, по умолчанию Docker контейнерами используется json-file драйвер для логирования информации, которая пишется сервисом внутри контейнера в stdout (и stderr). Для отправки логов во Fluentd используем docker драйвер fluentd

Отправка логов во Fluentd

Определим драйвер для логирования для сервиса post внутри compose-файла (ссылка на gist):

docker/docker-compose.yml

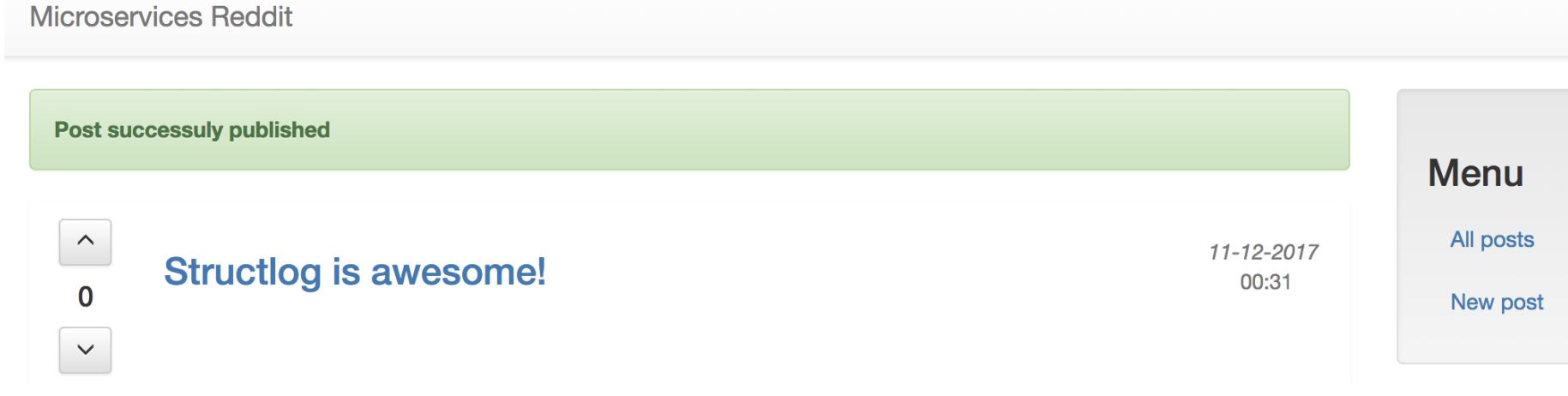
```
…
 post:
…
   logging:
      driver: "fluentd"
      options:
        fluentd-address: localhost:24224
tag: service.post
```

Сбор логов Post сервиса

Поднимем инфраструктуру централизованной системы логирования и перезапустим сервисы приложения Из каталога docker

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

Создадим несколько постов в приложении:



Kibana

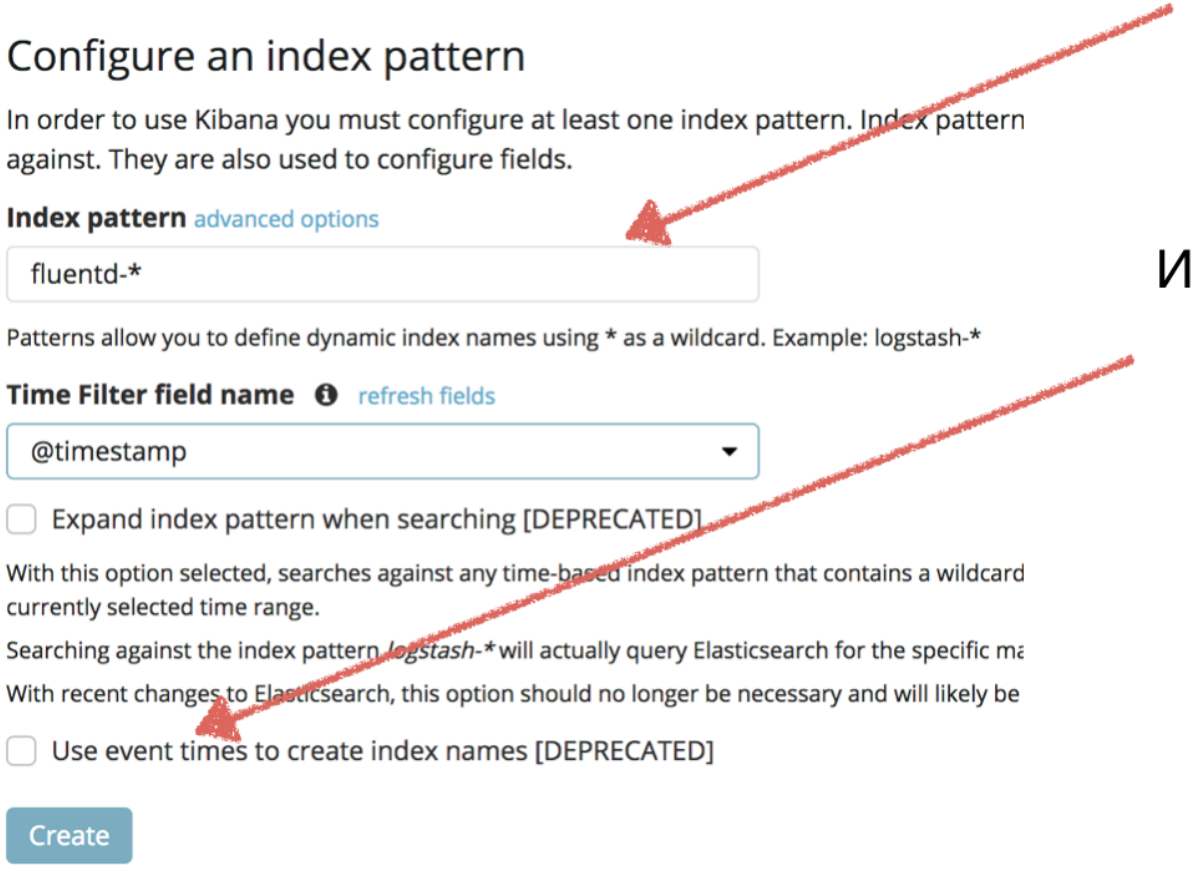
Kibana - инструмент для визуализации и анализа логов от компании Elastic.

Откроем WEB-интерфейс Kibana для просмотра собранных в ElasticSearch логов Post-сервиса (kibana слушает на порту 5601)

Kibana

Введем в поле патерна индекса fluentd-*

И создадим индекс маппинг



Kibana

Нажмем “Discover”, чтобы посмотреть информацию о полученных лог сообщениях

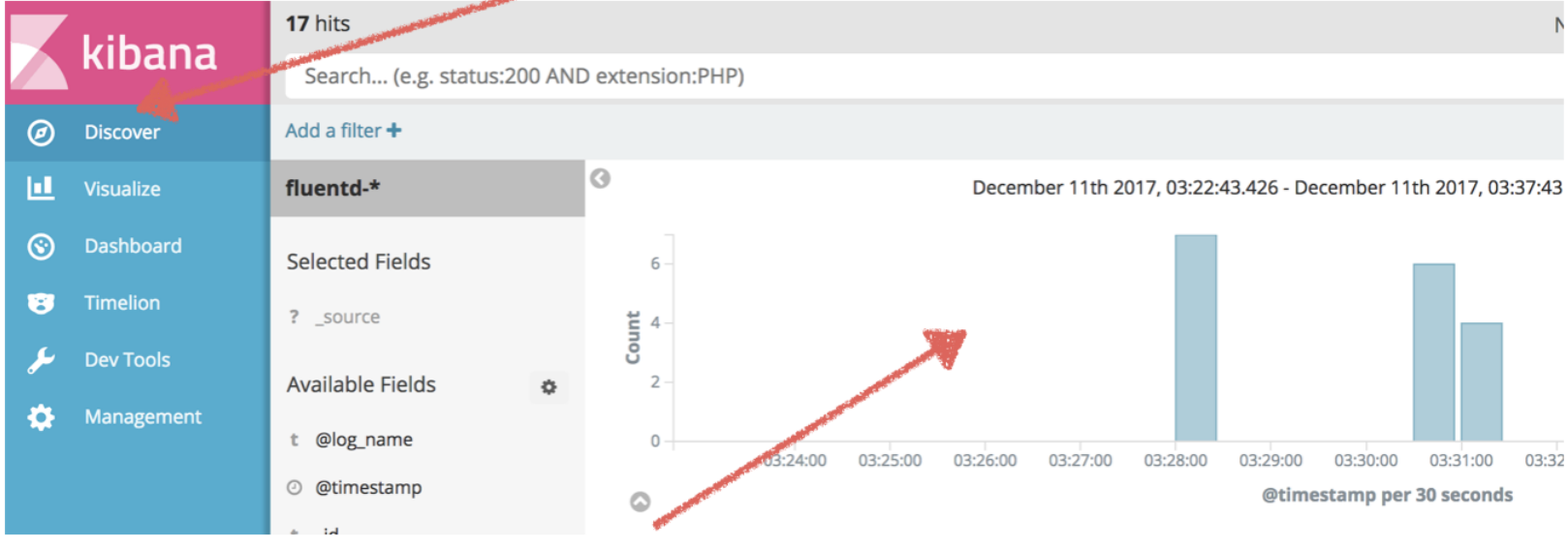
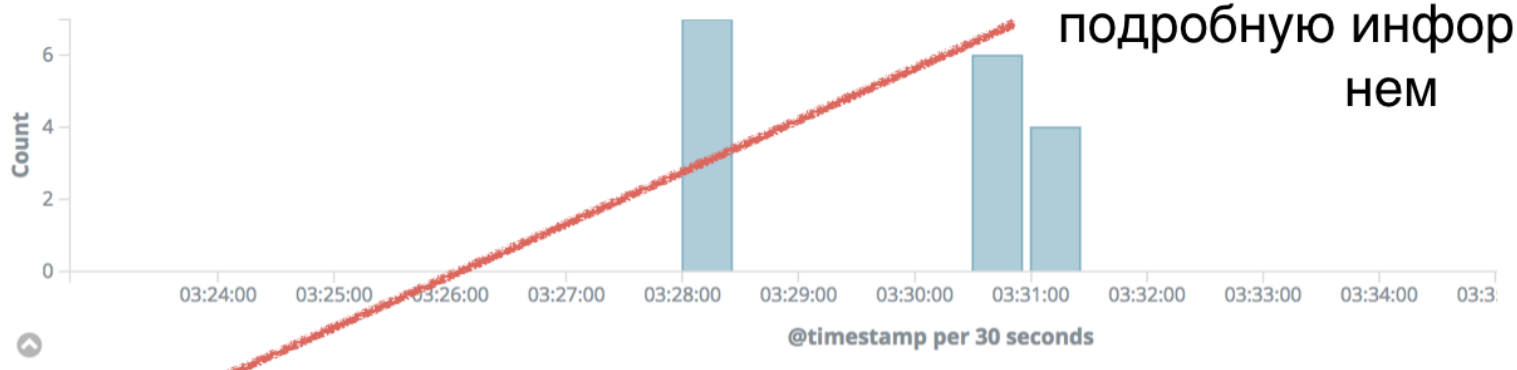


График покажет в какой момент времени поступало то или иное количество лог сообщений

Kibana

Нажмем на знак “развернуть” напротив одного из лог сообщений, чтобы посмотреть подробную информацию о нем



Time	_source
December 11th 2017, 03:31:13.000	log: {"event": "find_all_posts", "level": "info", "message": "Successfully the database", "params": {}, "request_id": "46619009-c7c8-4b6e-8ce5-95daa03 "timestamp": "2017-12-11 00:31:13"} container_id: af9683692d3a15078ea1b351 2bea53ea0fb48e container_name: /reddit_post_1 source: stdout @timestamp: 1:13.000 @log_name: af9683692d3a _id: AWBC-5rhCl56LW-dD5WK _type: access

Kibana

Видим лог-сообщение, которые мы недавно наблюдали в терминале. Теперь эти лог-сообщения хранятся централизованно в ElasticSearch. Также видим доп. информацию о том, откуда поступил данный лог.

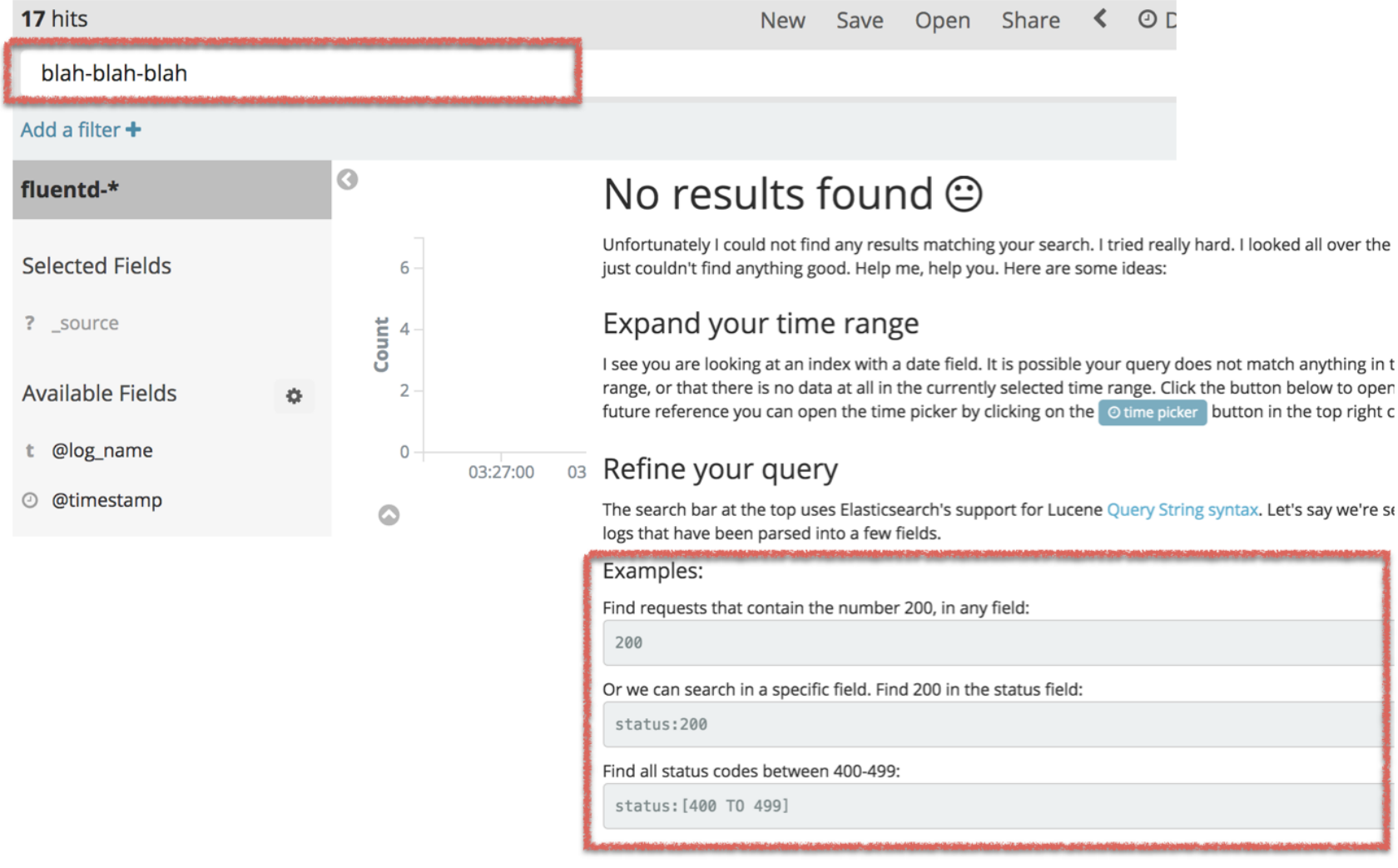
Kibana

Обратим внимание на то, что наименования в левом столбце, называются полями. По полям можно производить поиск для быстрого нахождения нужной информации

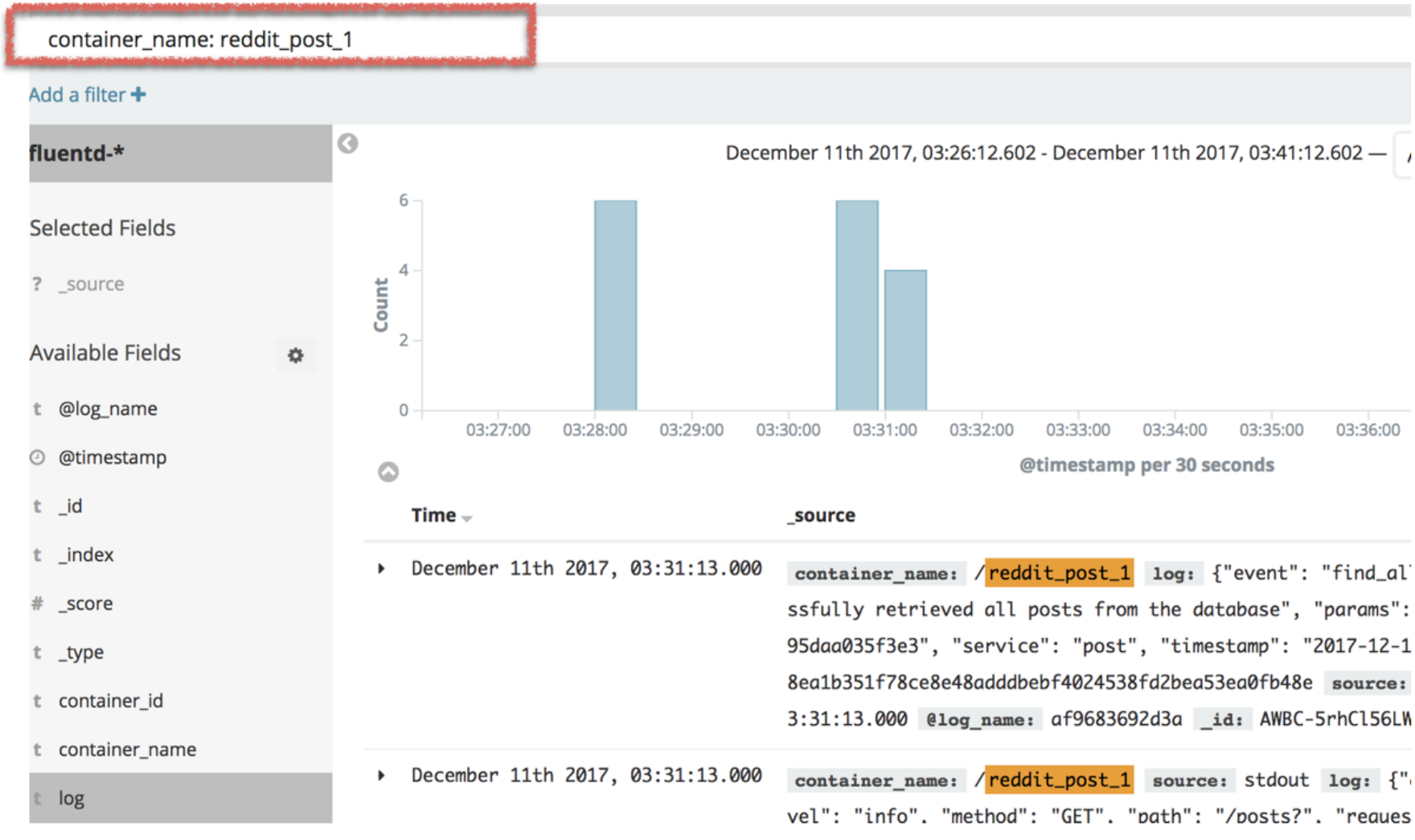
Table | JSON — View surrounding documents | View single document

t	@log_name	af9683692d3a
⏲	@timestamp	December 11th 2017, 03:31:13.000
t	_id	AWBC-5rhCl56LW-dD5WK
t	_index	fluentd-20171211
#	_score	-
t	_type	access_log
t	container_id	af9683692d3a15078ea1b351f78ce8e48adddbebf4024538fd2bea53ea0fb48e
t	container_name	/reddit_post_1
t	log	{"event": "find_all_posts", "level": "info", "message": "Successfully retrieved all posts from the database", "params": {}, "request_id": "46619009-c7c8-4b6e-8ce5-95daa035f3e3", "service": "post", "timestamp": "2017-12-11 00:31:13"}
t	source	stdout

Для того чтобы посмотреть некоторые примеры поиска, можно ввести в поле поиска произвольное выражение

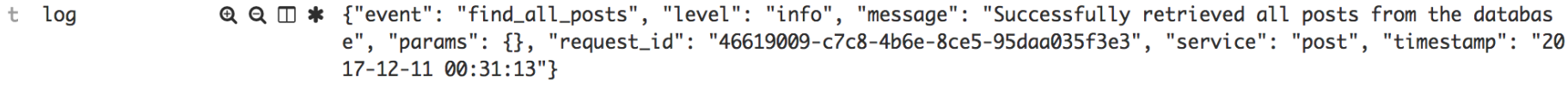


К примеру, посмотрев список доступных полей, мы можем выполнить поиск всех логов, поступивших с контейнера **reddit_post_1**



Фильтры

Заметим, что поле log содержит в себе JSON объект, который содержит много интересной нам информации.



Нам хотелось бы выделить эту информацию в поля, чтобы иметь возможность производить по ним поиск. Например, для того чтобы найти все логи, связанные с определенным событием (event) или конкретным сервисов (service).

Мы можем достичь этого за счет использования **фильтров** для выделения нужной информации.

Фильтры

Добавим фильтр для парсинга json логов, приходящих от post сервиса, в конфиг fluentd

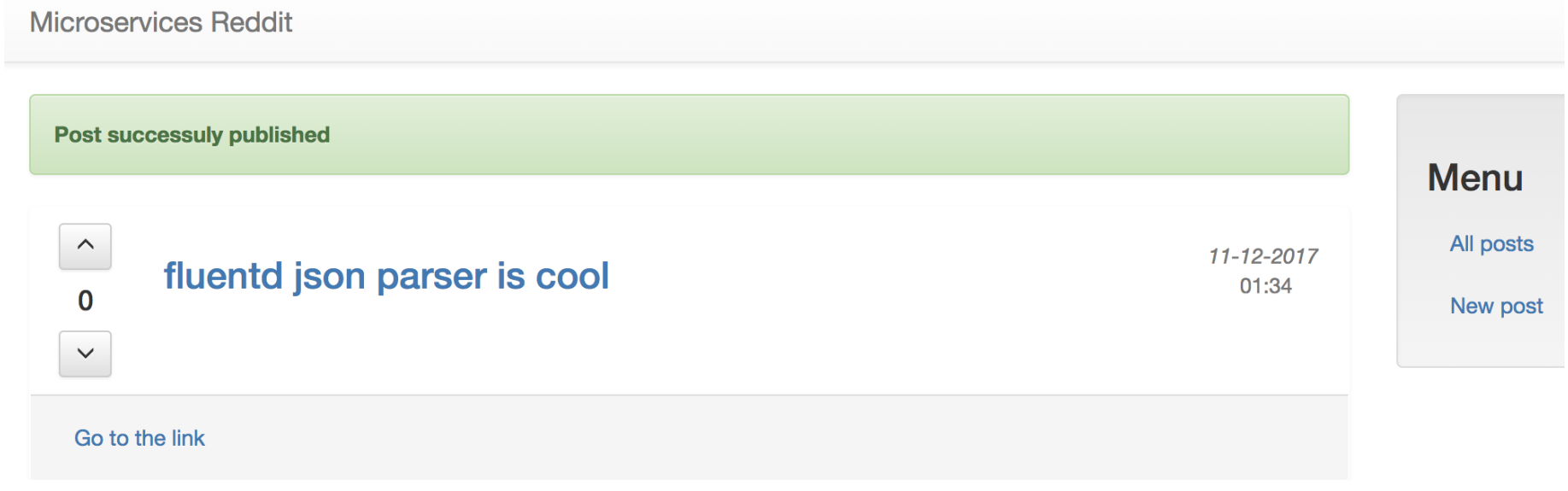
logging/fluentd/fluent.conf

```
  <source>
   @type forward
   port 24224
   bind 0.0.0.0
</source>

<filter service.post>
   @type parser
   format json
   key_name log
</filter>

<match *.**>
   @type copy
...
```

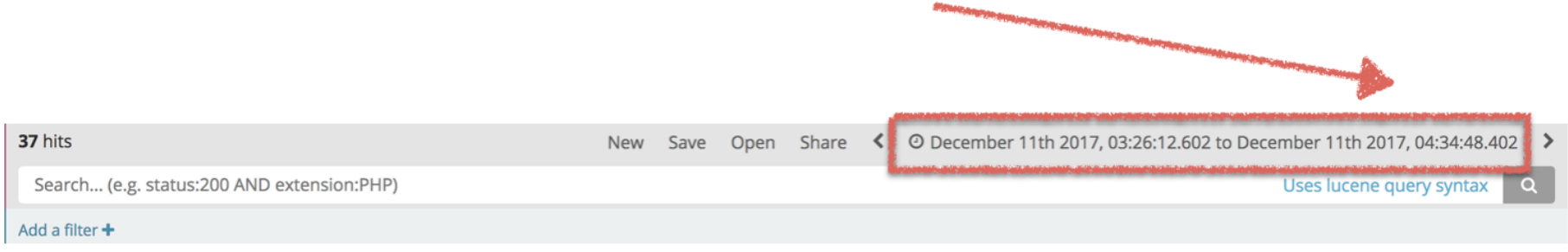
После этого персоберите образ и перезапустите сервис fluentd

```
logging/fluentd $ docker build -t $USER_NAME/fluentd
docker/ $ docker-compose -f docker-compose-logging.yml up -d fluentd
```

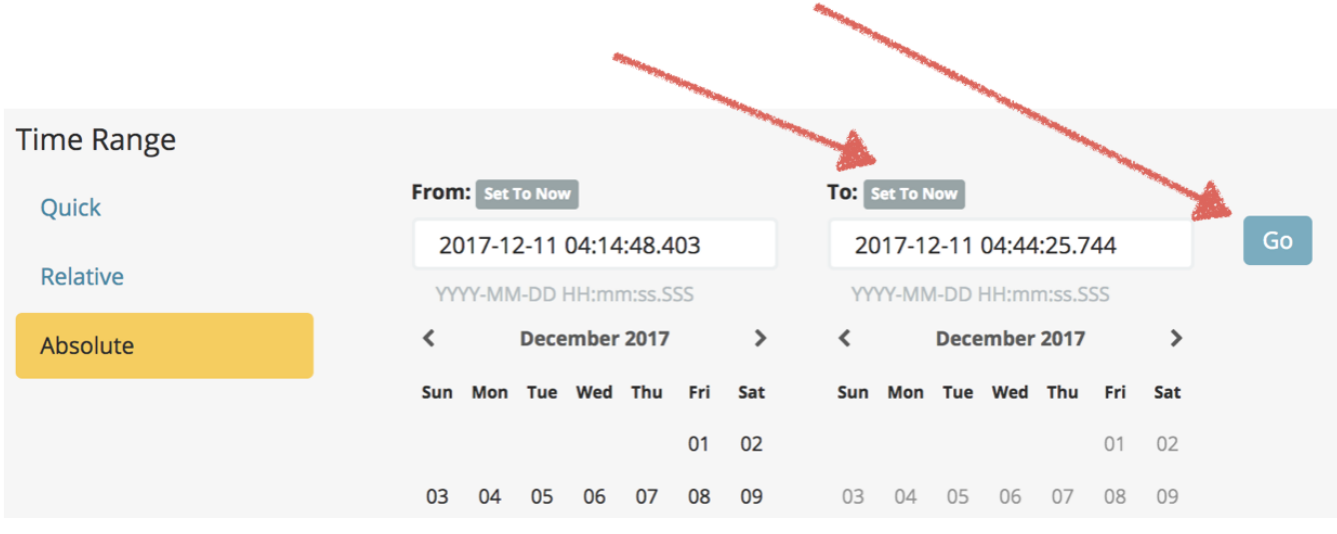
Создадим пару новых постов, чтобы проверить парсинг логов



Вновь обратимся к Kibana. Прежде чем смотреть логи убедимся, что временной интервал выбран верно. Нажмите один раз на дату со временем



Выберите set to now и Go



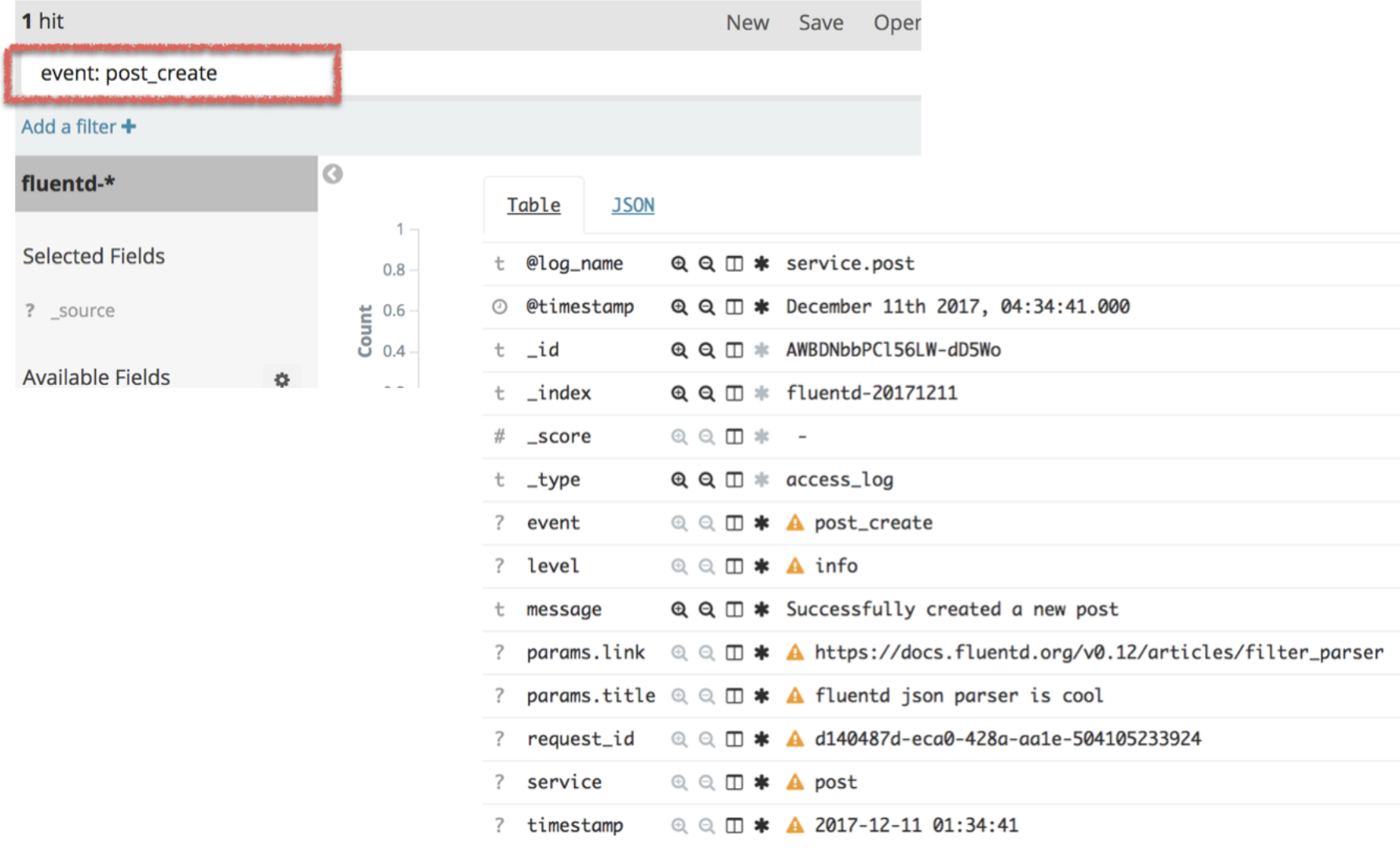
Взглянем на одно из сообщений и увидим, что вместо одного поля log появилось множество полей с нужной нам информацией

December 11th 2017, 04:34:41.000 **addr:** 172.21.0.8 **event:** request **level:** info **method:** GET **path:** /posts? **request_id:** b3c0d78a-59e-46b7-a02e-8bbf2ba28ac6 **response_status:** 200 **service:** post **timestamp:** 2017-12-11 01:34:41 **@timestamp:** December 11th 2017, 04:34:41.000 **@log_name:** service.post **_id:** AWBDNbbPCl56LW-dD5Wr **_type:** access_log **_index:** fluentd-20171211 **_score:** -

Table JSON View surrounding documents View single docume

t	@log_name	service.post
⏲	@timestamp	December 11th 2017, 04:34:41.000
t	_id	AWBDNbbPCl56LW-dD5Wr
t	_index	fluentd-20171211
#	_score	-
t	_type	access_log
?	addr	172.21.0.8
?	event	request
?	level	info
?	method	GET
?	path	/posts?
?	request_id	b3c0d78a-59ee-46b7-a02e-8bbf2ba28ac6
?	response_status	200
?	service	post
?	timestamp	2017-12-11 01:34:41

Выполним для пример поиск по событию создания нового поста



Неструктурированные логи

Неструктурированные логи

Неструктурированные логи отличаются отсутствием четкой структуры данных. Также часто бывает, что формат лог-сообщений не подстроен под систему централизованного логирования, что существенно увеличивает затраты вычислительных и временных ресурсов на обработку данных и выделение нужной информации.

На примере сервиса ui мы рассмотрим пример логов с неудобным форматом сообщений.

Логирование UI сервиса

По аналогии с post сервисом определим для ui сервиса драйвер для логирования fluentd в compose-файле

docker/docker-compose.yml

```
ui:
    image: artemmkin/ui
    environment:
      - POST_SERVICE_HOST=post
      - POST_SERVICE_PORT=5000
      - COMMENT_SERVICE_HOST=comment
      - COMMENT_SERVICE_PORT=9292
    ports:
      - "9292:9292"
    depends_on:
      - post
    logging:
      driver: "fluentd"
      options:
        fluentd-address: localhost:24224
        tag: service.ui
```

Перезапустим ui сервис Из каталога docker

```
$ docker-compose stop ui
$ docker-compose rm ui
$ docker-compose up -d
```

Посмотрим на формат собираемых сообщений

Парсинг

Когда приложение или сервис не пишет структурированные логи, приходится использовать старые добрые **регулярные выражения** для их парсинга в **/docker/fluentd/fluent.conf**

Следующее **регулярное выражение** нужно, чтобы успешно выделить интересующую нас информацию из лога UI-сервиса в поля (ссылка на gist)

```
  …
<filter service.post>
  @type parser
  format json
  key_name log
</filter>

<filter service.ui>
  @type parser
  format /\[(?<time>[^\]]*)\]  (?<level>\S+) (?<user>\S+)[\W]*service=(?<service>\S+)[\W]*event=(?
<event>\S+)[\W]*(?:path=(?<path>\S+)[\W]*)?request_id=(?<request_id>\S+)[\W]*(?:remote_addr=(?
<remote_addr>\S+)[\W]*)?(?:method= (?<method>\S+)[\W]*)?(?:response_status=(?<response_status>\S+)
[\W]*)?(?:message='(?<message>[^\']*)[\W]*)?/
  key_name log
</filter>
…
```

Перезапускаем кибану

Обновите образ

Из каталога docker

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

Парсинг

Результат должен выглядить следующим образом

Table JSON

t	@log_name	service.ui
⏲	@timestamp	December 12th 2017, 15:54:32.000
t	_id	AWBKyoN3n7PT1mjlldn1
t	_index	fluentd-20171212
#	_score	-
t	_type	access_log
t	event	show_all_posts
t	level	INFO
t	message	Successfully showed the home page with posts
t	request_id	97f21446-0591-472d-a60b-bab22d3e02ac
t	service	ui
?	user	--

t	@log_name	service.ui
⏲	@timestamp	December 12th 2017, 15:54:32.000
t	_id	AWBKyoN3n7PT1mjlldn2
t	_index	fluentd-20171212
#	_score	-
t	_type	access_log
t	event	request
t	level	INFO
t	method	GET
t	path	/
?	remote_addr	82.179.176.251
t	request_id	97f21446-0591-472d-a60b-bab22d3e02ac
#	response_status	200
t	service	ui
?	user	--

Парсинг

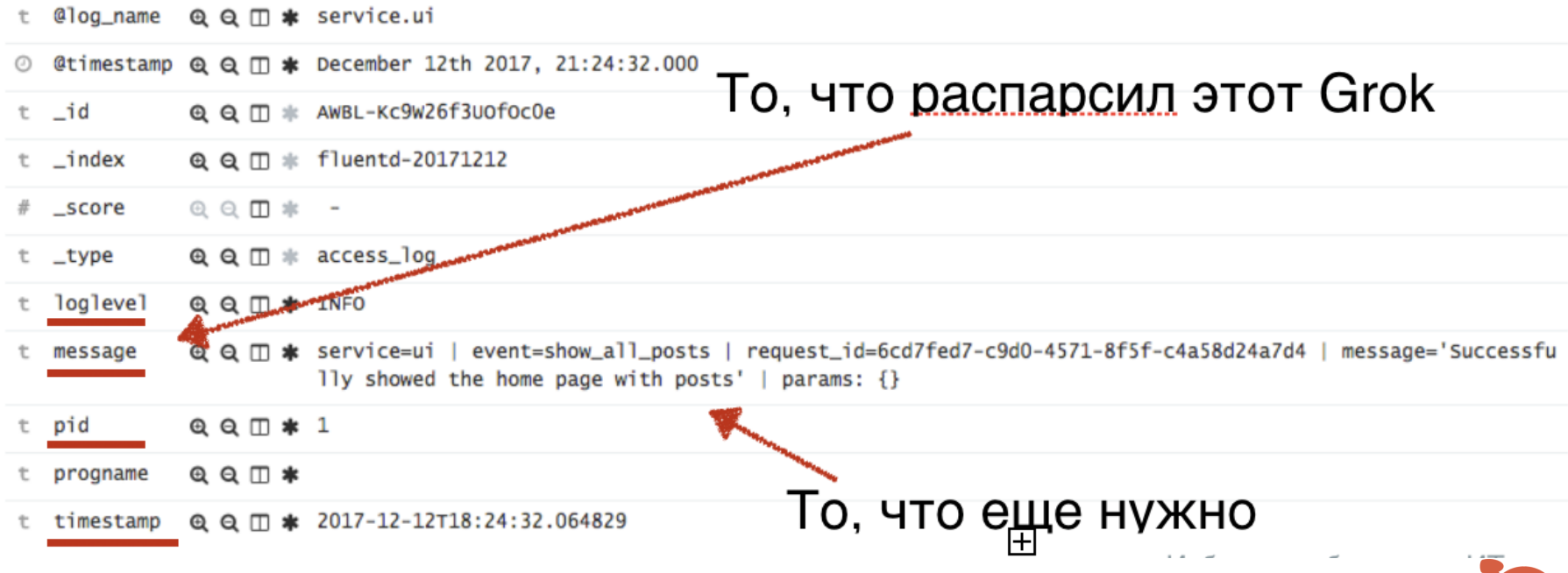
Созданные регулярки могут иметь ошибки, их сложно менять и невозможно читать. Для облегчения задачи парсинга вместо стандартных регулярок можно использовать **grok**-шаблоны. По-сути **grok**'и - это именованные шаблоны регулярных выражений (очень похоже на функции). Можно использовать готовый regexp, просто сославшись на него как на функцию **docker/fluentd/fluent.conf**

```
  …
<filter service.ui>
  @type parser
  format grok
  grok_pattern %{RUBY_LOGGER}
key_name log
</filter>
…
```

Парсинг

Это grok-шаблон, зашитый в плагин для fluentd. В развернутом виде он выглядит вот так:

```
%{RUBY_LOGGER} [(?<timestamp>(?>\d\d){1,2}-(?:0?[1-9]|1[0-2])-(?:(?:0[1-9])|(?:
[12][0-9])|(?:3[01])|[1-9])[T ](?:2[0123]|[01]?[0-9]):?(?:[0-5][0-9])(?::?(?:(?:[0-
5]?[0-9]|60)(?:[:.,][0-9]+)?))?(?:Z|[+-](?:2[0123]|[01]?[0-9])(?::?(?:[0-5][0-
9])))?) #(?<pid>\b(?:[1-9][0-9]*)\b)\] *(?<loglevel>
(?:DEBUG|FATAL|ERROR|WARN|INFO)) -- +(?<progname>.*?): (?<message>.*)
```



Парсинг

Как было видно на предыдущем слайде - часть логов нужно еще распарсить. Для этого используем несколько Grok-ов по-очереди

docker/fluentd/fluent.conf (ссылка на gist)

```
  …
<filter service.ui>
  @type parser
  key_name log
  format grok
  grok_pattern %{RUBY_LOGGER}
</filter>

<filter service.ui>
  @type parser
  format grok
  grok_pattern service=%{WORD:service} \| event=%{WORD:event} \| request_id=%{GREEDYDATA:request_id} \|
message='%{GREEDYDATA:message}'
  key_name message
  reserve_data true
</filter>
…
```

Парсинг

В итоге получим в Kibana (если совершаем действия в ui-сервисе):

t	@log_name	service.ui
🕒	@timestamp	December 13th 2017, 11:46:23.000
t	_index	AWBPDbQCI15IV2WOAWpM
t	_index	fluentd-20171213
#	_score	-
t	_type	access_log
t	event	show_post
t	loglevel	INFO
t	message	Successfully showed the post
t	pid	1
t	progname	
t	request_id	ac97346b-9c21-4358-a2e8-8101cfa18cd3
t	service	ui
t	timestamp	2017-12-13T08:46:23.826566

Задания со *

1. UI-сервис шлет логи в нескольких форматах.

Такой лог остался неразобранным. Составьте конфигурацию fluentd так, чтобы разбирались оба формата логов UI-сервиса (тот, что сделали до этого и текущий) одновременно.

2. Разобраться с темой распределенного трейсинга и решить проблему в конце данного файла

Распределенный трейсинг

Zipkin

Добавьте в compose-файл для сервисов логирования сервис распределенного трейсинга Zipkin (ссылка на gist)

docker/docker-compose-logging.yml

```
version: '3'

services:
  zipkin:
    image: openzipkin/zipkin
    ports:
      - "9411:9411"
…
```

docker-compose.yml

Правим наш docker/docker-compose-logging.yml

Добавьте для каждого сервиса поддержку ENV переменных и задайте параметризованный параметр ZIPKIN_ENABLED

```
environment:
 - ZIPKIN_ENABLED=${ZIPKIN_ENABLED}
```

В .env файле укажите

```
ZIPKIN_ENABLED=true
```

Перевыкатите приложение с обновлением

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

Networks

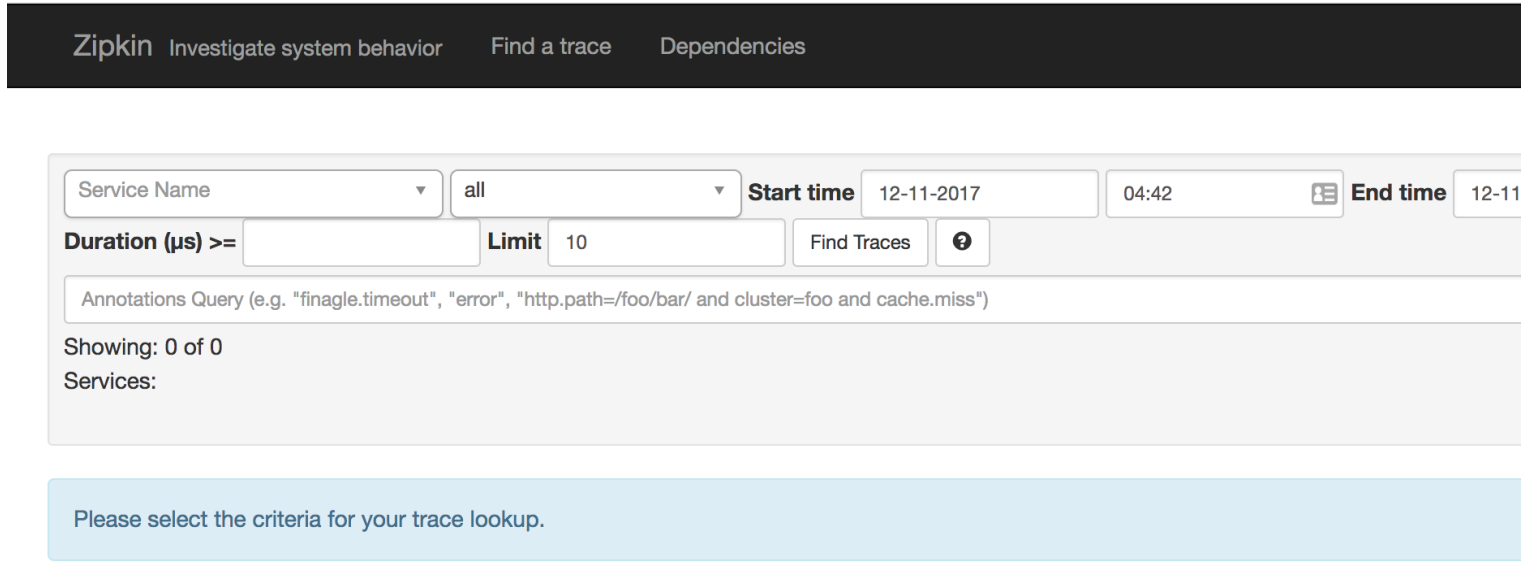
Zipkin должен быть в одной сети с приложениями, поэтому, если вы выполняли задание с сетями, вам нужно объявить эти сети в **docker-compose-logging.yml** и добавить в них zikpkin похожим образом:

```
services:
  zipkin:
    image: openzipkin/zipkin
    ports:
      - "9411:9411"
    networks:
      - front_net
      - back_net

networks:
  back_net:
  front_net:
```

Пересоздадим наши сервисы

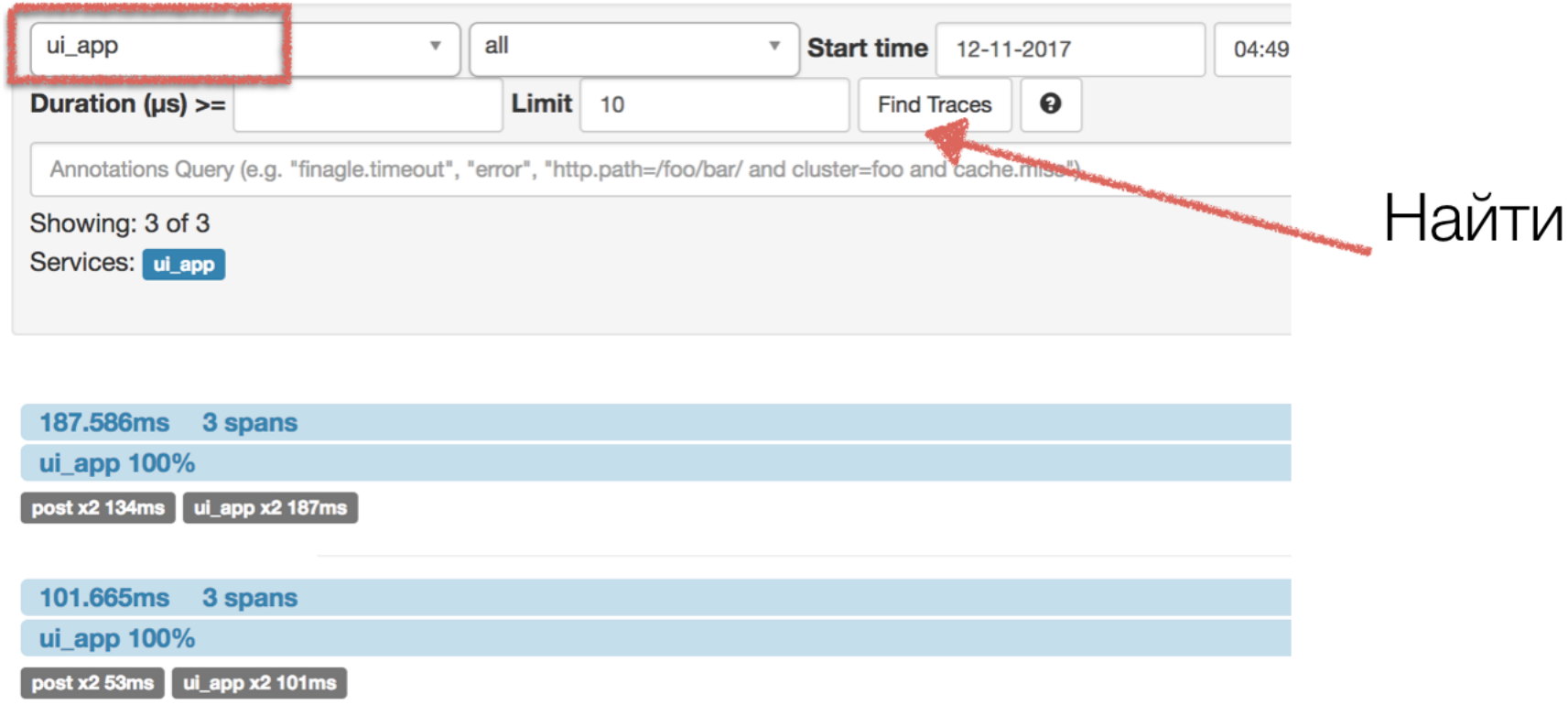
```
$ docker-compose -f docker-compose-logging.yml -f docker-compose.yml down
$ docker-compose -f docker-compose-logging.yml -f docker-compose.yml up -d
```

Откроем Zipkin WEB UI на порту 9411, пока никаких трейсов поиск не должен дать, т.к. никаких запросов нашему приложению еще не поступало

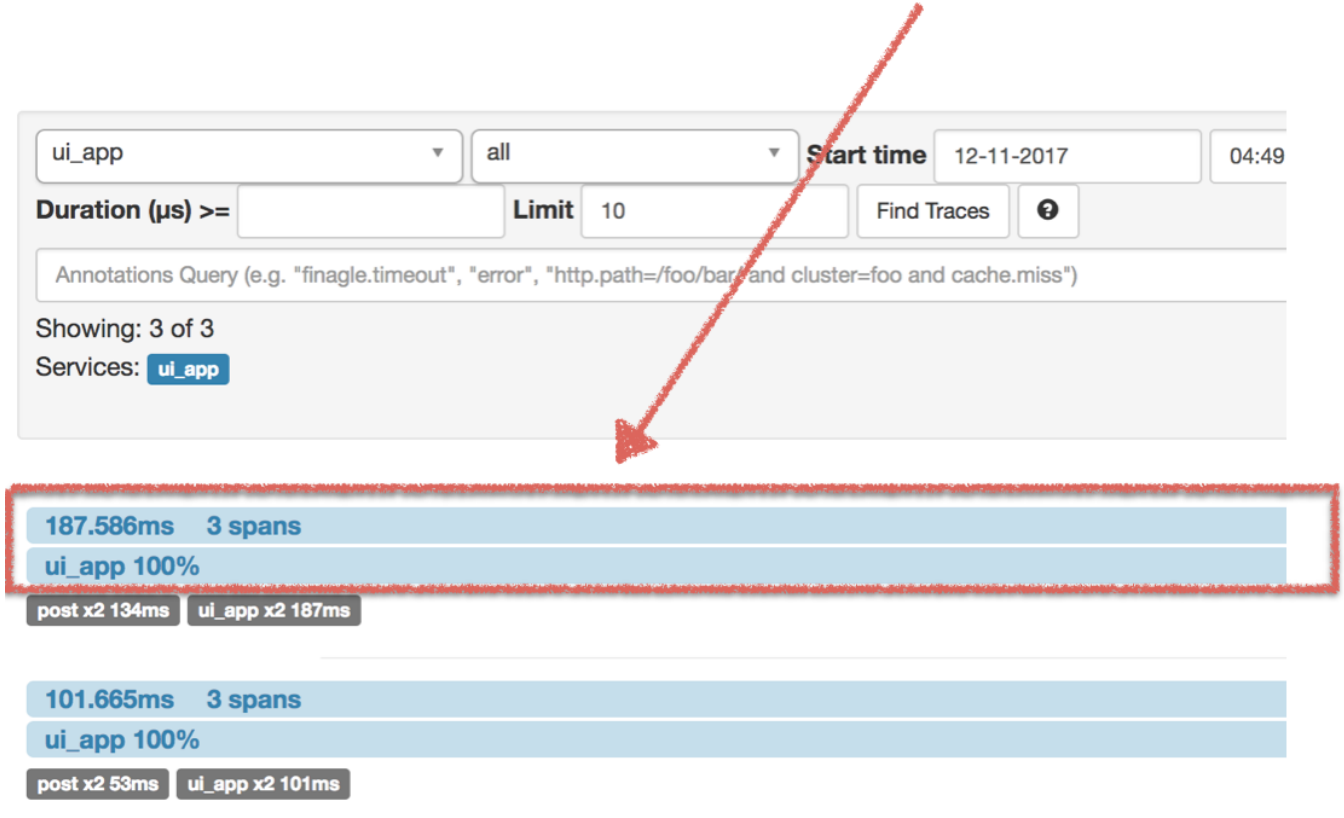


Откроем главную страницу приложения и обновим ее несколько раз.

Заглянув затем в UI Zipkin (страницу потребуется обновить), мы должны найти несколько трейсов (следов, которые оставили запросы проходя через систему наших сервисов).

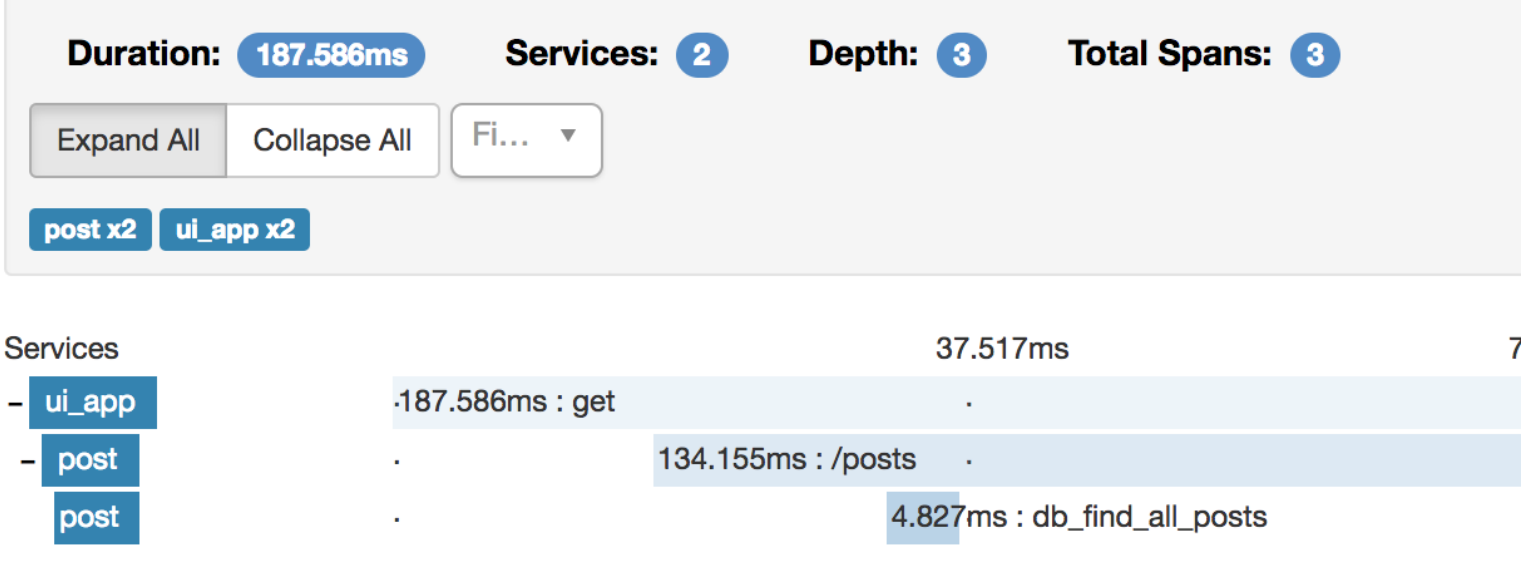


Нажмем на один из трейсов, чтобы посмотреть, как запрос шел через нашу систему микросервисов и каково общее время обработки запроса у нашего приложения при запросе главной страницы

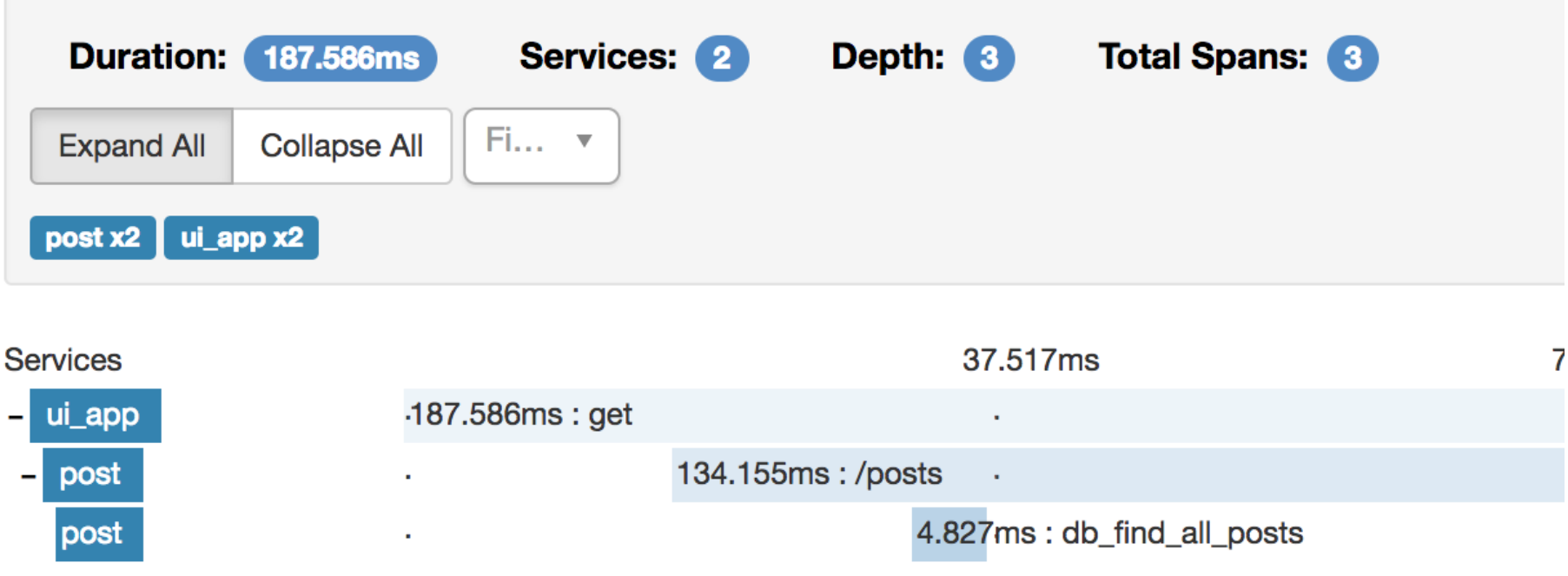


Видим, что первым делом наш запрос попал к ui сервису, который смог обработать наш запрос за суммарное время равное 187.566 ms.

Из этих 187 ms ушло 134.155ms на то чтобы ui мог направить запрос post сервису по пути /posts и получить от него ответ в виде списка постов. Post сервис в свою очередь использовал функцию обращения к БД за списком постов, на что ушло 4.827 ms.



Повторим немного терминологию: синие полоски со временем называются **span** и представляют собой одну операцию, которая произошла при обработке запроса. Набор **span**-ов называется трейсом. Суммарное время обработки нашего запроса равно верхнему **span**-у, который включает в себя время всех **span**-ов, расположенных под ним.



Самостоятельное задание со звездочкой

С нашим приложением происходит что-то странное. Пользователи жалуются, что при нажатии на пост они вынуждены долго ждать, пока у них загрузится страница с постом. Жалоб на загрузку других страниц не поступало. Нужно выяснить, в чем проблема, используя Zipkin.

Ссылка на репозиторий со сломанным кодом приложения.

Описание проблемы можно опубликовать в PR, в разделе выполнения задания со звездочкой

Проверка ДЗ

- Результаты вашей работы находятся в ветке **logging-1** вашего microservices репозитория
- В README внесите описание того, что сделано
- Создайте Pull Request к ветке master (**описание PR нужно заполнять**)
- Добавьте "Labels" **logging-1** и **Logging** к вашему Pull Request
- После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить и закрыть PR.