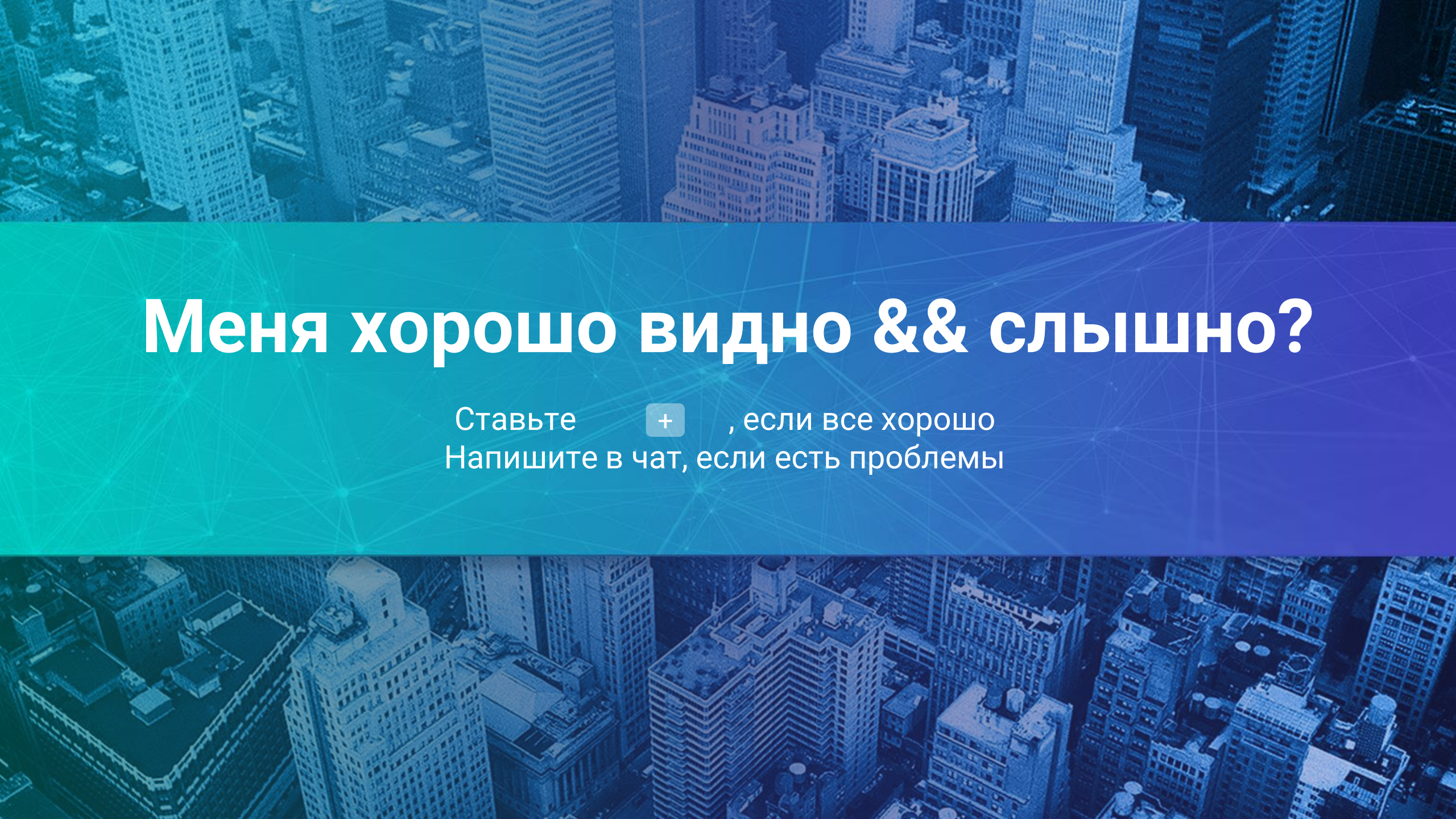

OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование

Проверить, идет ли запись!



Меня хорошо видно && слышно?

Ставьте + , если все хорошо

Напишите в чат, если есть проблемы

ELK stack

Елагин Алексей

Безработный

@sh1kel

Правила вебинара

Активно участвуем

Задаем вопрос в чат или голосом

Off-topic обсуждаем в Slack #канал группы или #general

Вопросы вижу в чате, могу ответить не сразу

Карта курса

Мы тут

1 Классическое логирование в Linux

2 Системы логирования (ELK, EFK, Graylog2)

3 ELK стэк

4 Logstash

5 Graylog2

Маршрут вебинара

Какую проблему решает ELK

Особенности эксплуатации

Практика

Рефлексия

Цели вебинара | После занятия вы сможете

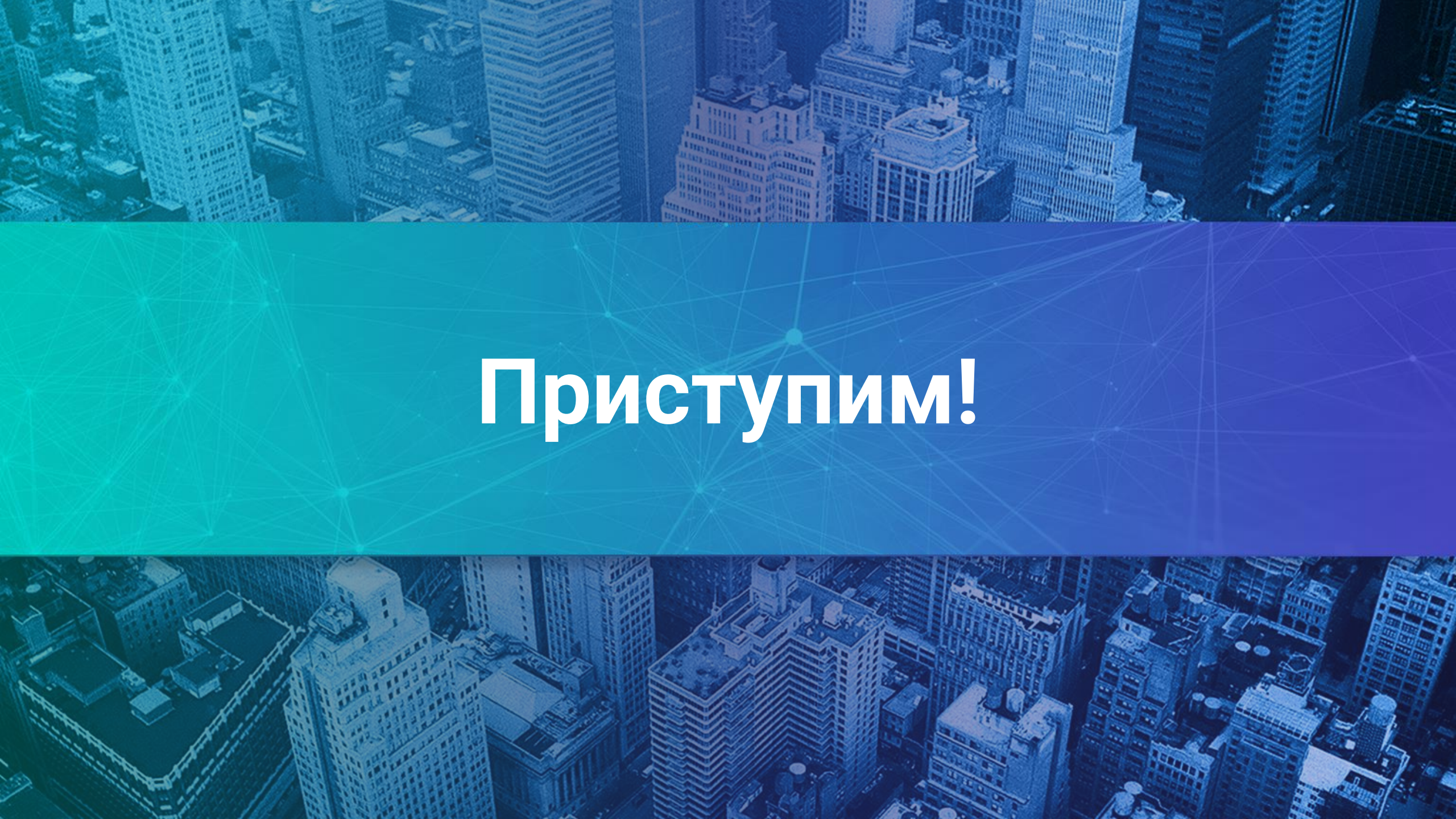
1. Правильно разворачивать ELK
2. Тюнить Elasticsearch для prod окружений
3. Делать запросы в Kibana

Смысл | Зачем вам это уметь

1. Elasitcsearch – гибкий, но сложный инструмент, который требует настройки

2. Без понимания внутреннего устройства стэка, большая вероятность наступить на все возможные грабли

3. Правильный подход к разворачиванию и настройке обеспечит простой сбор и анализ логов больших масштабов


Приступим!

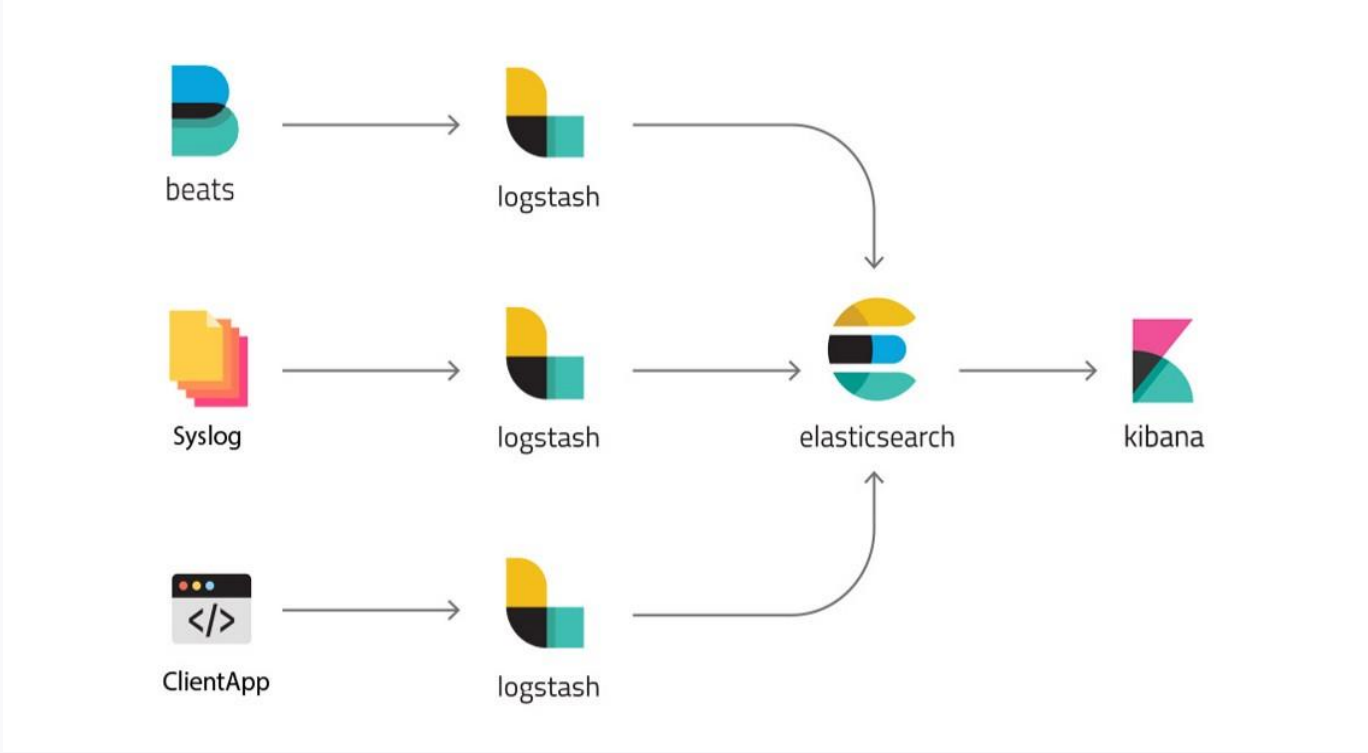
Что вообще за ELK

Три компонента:

1. **Elasticsearch** – хранение и анализ данных
2. **Logstash** – парсинг и трансформация (и немного транспорт)
3. **Kibana** – web интерфейс для выполнения запросов и визуализации

Как они взаимодействуют?

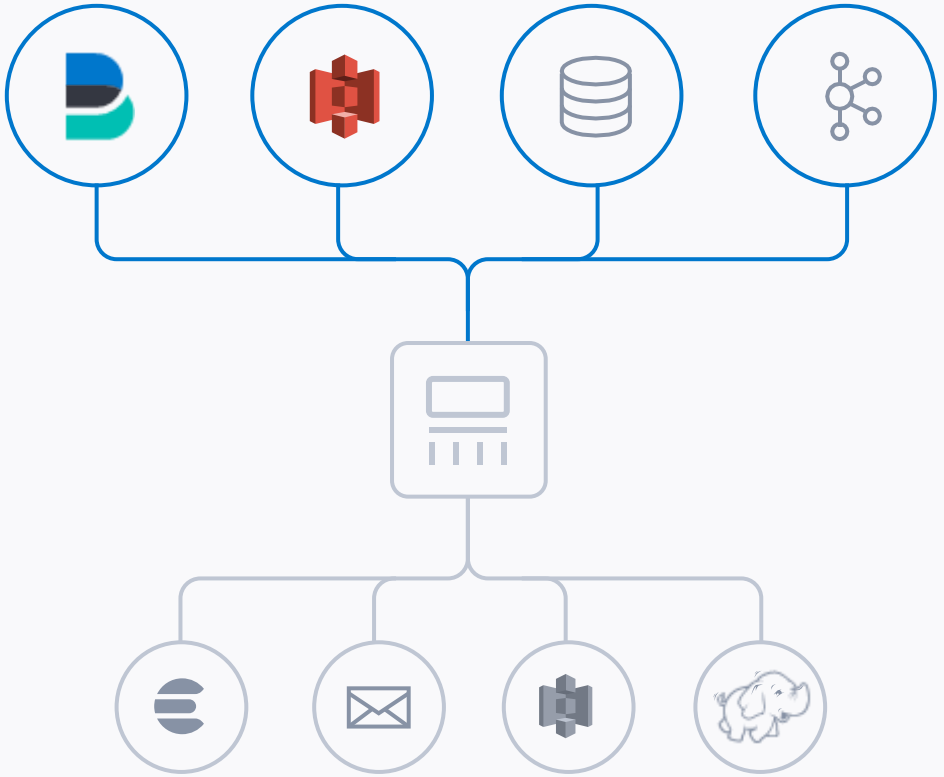
Logstash принимает логи от наших приложений (и вероятно обрабатывает) и отправляет дальше по цепочке.
Elasticsearch принимает данные, индексирует, выполняет запросы.
Kibana предоставляет интерфейс для конструирования запросов, настройки индексов и визуализации данных.



Logstash

Что умеет **Logstash?**

- Трансформация логов независимо от их формата и сложности с помощью **grok**.
- Получение geo координат из IP адреса.
- Анонимизация или исключение чувствительных данных из логов.



Elasticsearch

Что умеет **Elasticsearch**?
- Комбинирование различных вариантов данных – структурированных или нет, geo-данных, метрик, чего угодно.
- Выполнение поисковых запросов по данным и агрегация результатов.

Частые кейсы использования:
- Логи
- Метрики
- Данные мониторинга
- Поисковый движок для сайта
- Корпоративное хранилище документов
- Хранилище данных для картографии
- Хранение и анализ данных для определения угроз безопасности

Elasticsearch внутри

- Написана на Java
- Движок индексирования Lucene
- Синхронизация на Raft

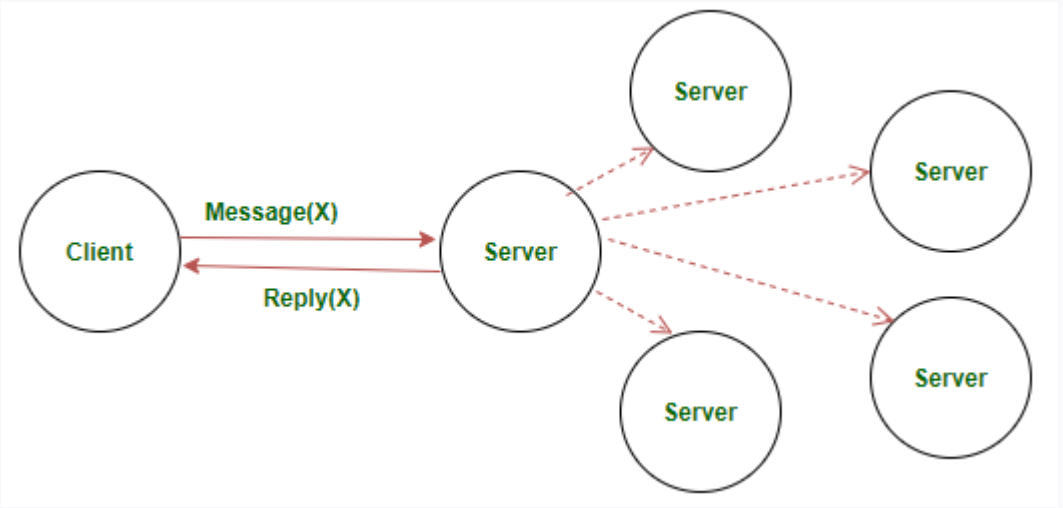
Плюсы и минусы подхода

+ Платформонезависимость
+ Масштабируемость
+ Fault tolerance

- Любит много памяти
- Проблемы с выделением памяти
- Геораспределенные базы работают плохо

Как с этим жить?

- Оставлять много памяти под page cache
- Делать много шардов

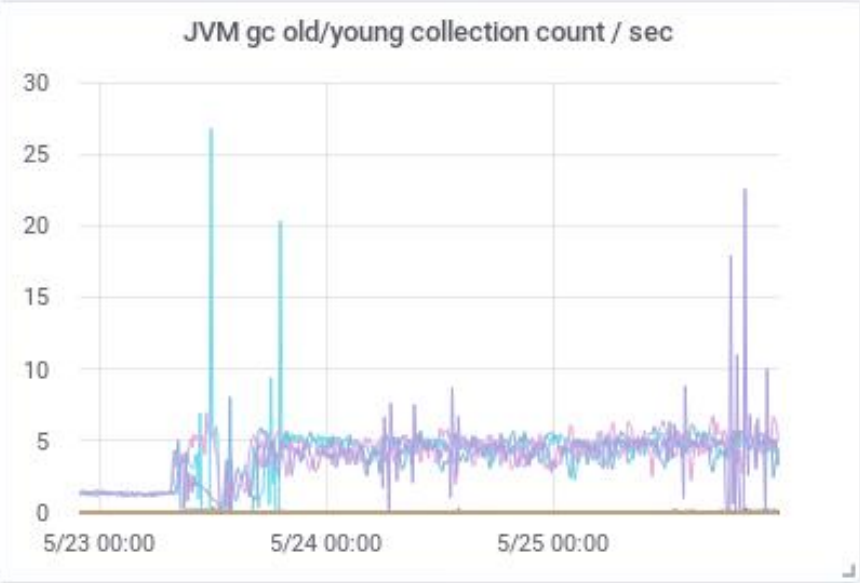


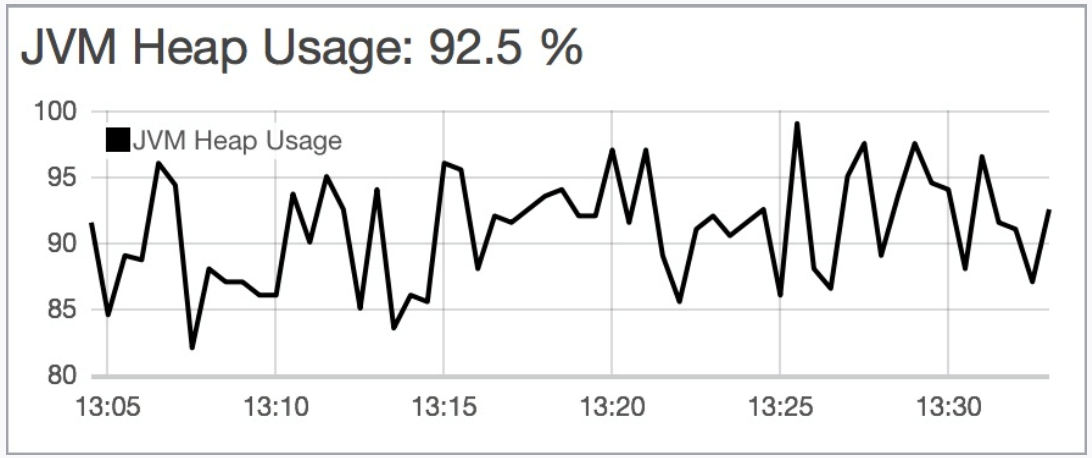
Elasticsearch еще про память

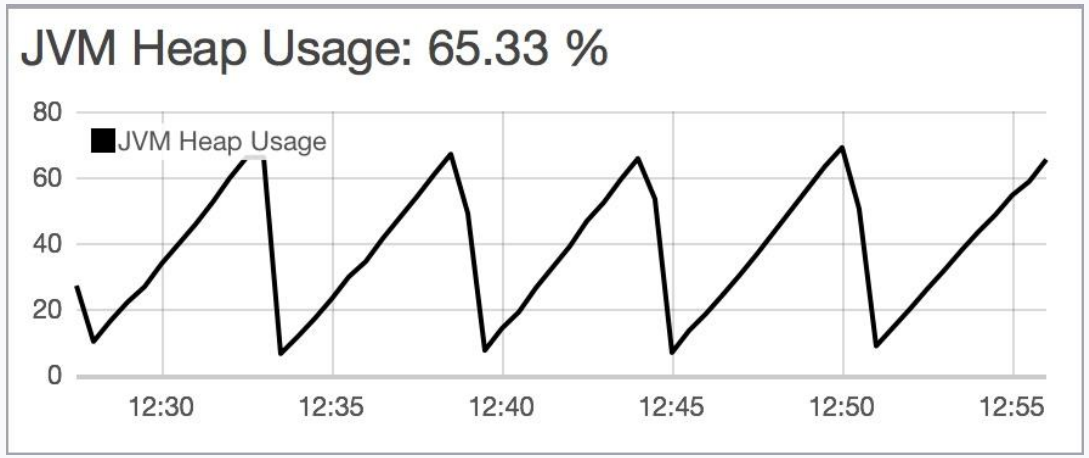
JVM и память.
Чем больше тем лучше? Нет!

Oop = Ordinary object pointer
Xmx & Xms <= 31Gb

Garbage collector time



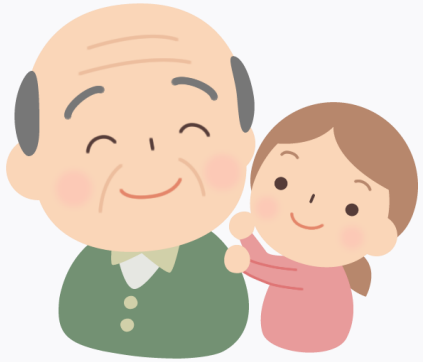




Elasticsearch и память: как улучшить?

Что там с **Garbage Collector?**
Java 8 и *ConcMarkSweepGC – не очень.*
Java 11 и *G1GC – очень.*

Young generation vs Old generation
По-умолчанию 1/2
Лучше наоборот 2/1 или 3/1
ParallelGCThreads = число ядер

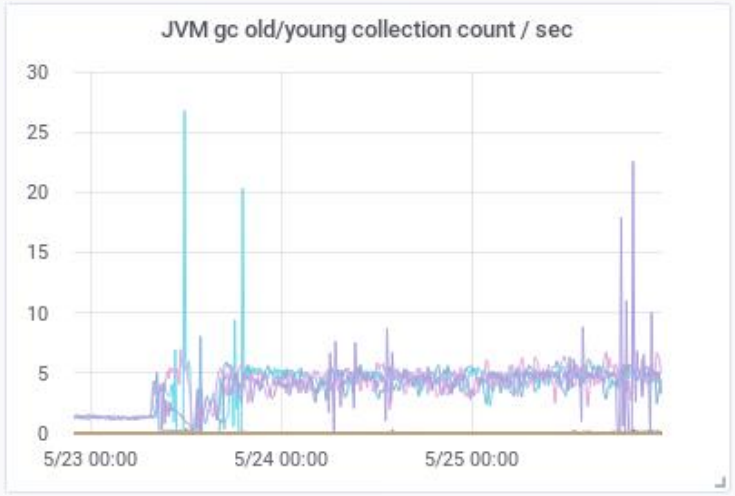


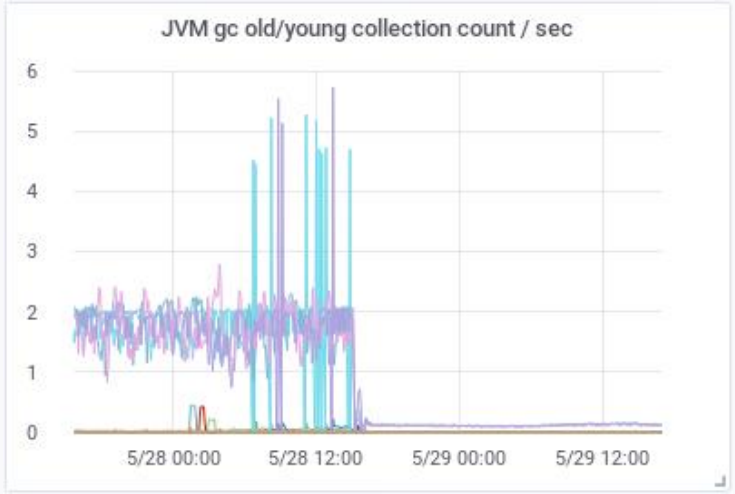
***-XX:NewRatio**=2*

```
NCPU = `grep 'cpu cores' /proc/cpuinfo | uniq
| cut -f 2 -d “:”`
```

-XX:*ParallelGCThreads*=NCPU

indices.memory.index_buffer_size = 20%
index.refresh_interval = 30s (осторожно)



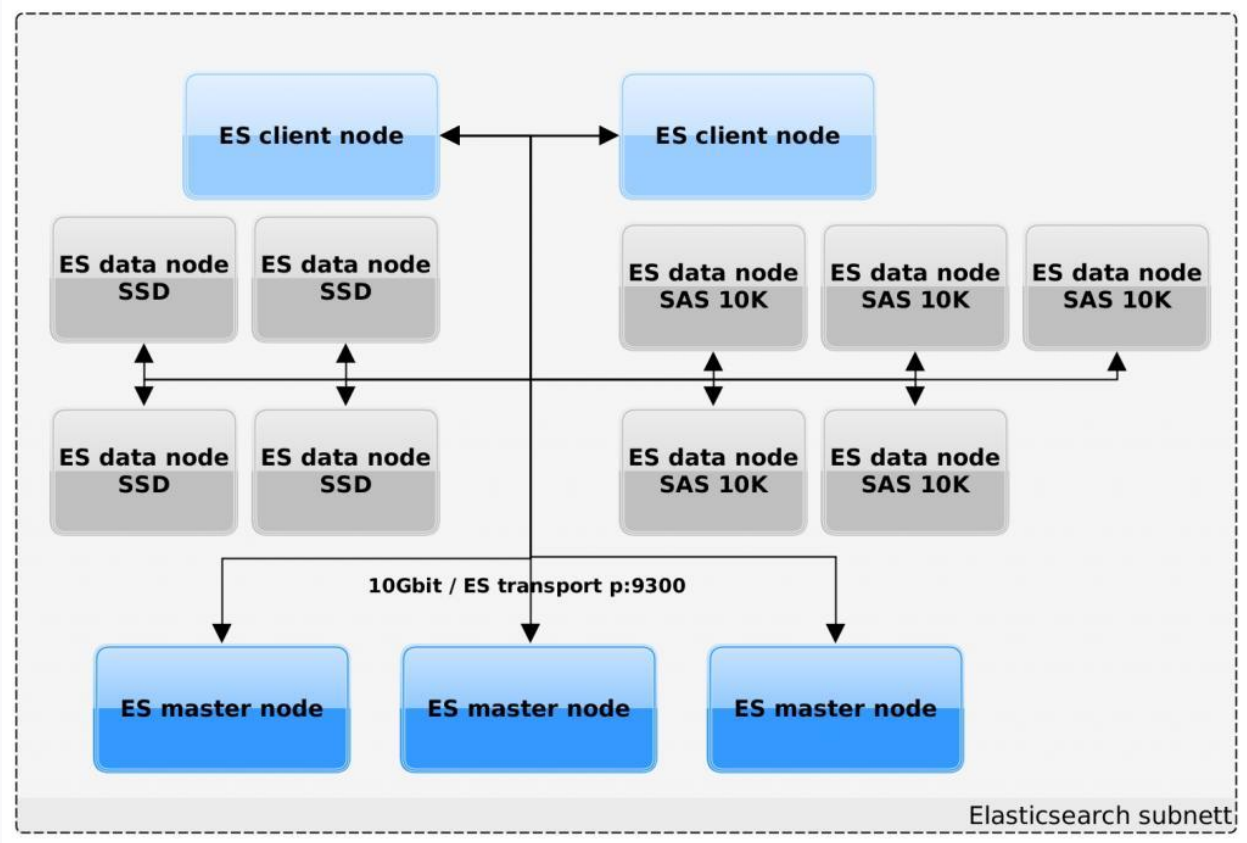


Elasticsearch топология нод

Типы нод в Elasticsearch

- Master – своего рода маршрутизаторы, хранят полную информацию о состоянии кластера, владеют информацией о нахождении каждого индекса и шарда
- Data – занимаются собственно хранением данных и выполнением запросов.
- Ingest – балансировщики нагрузки, проксирующие запросы к кластеру, агрегирующие и кэширующие ответы.

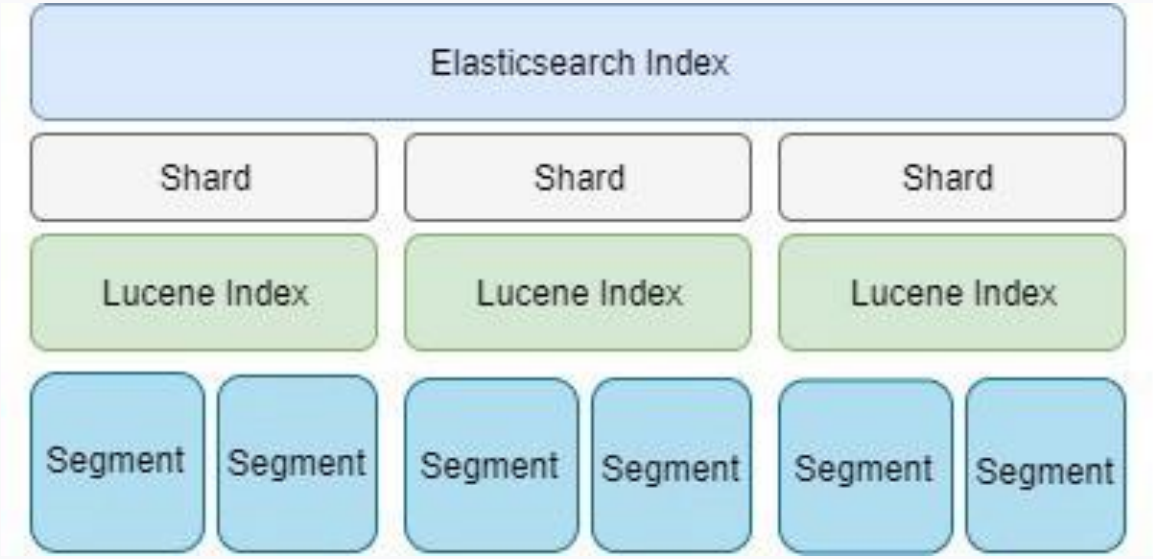
Master: *node.master:true node.data: false node.ingest: false*
Data: *node.master:false node.data: true node.ingest: false*
Client: *node.master: false node.data: false node.ingest: true*



Elasticsearch и хранение данных

Shard – единица хранения данных внутри кластера. Каждый запрос порождает поток, который обращается к конкретному шарду.

- Размер шарда <= Скорость обработки запросов => Нагрузка на CPU
- Каждый шард – структура в памяти, даже если сами данные на диске.
- Количество шардов нельзя поменять для созданного индекса.

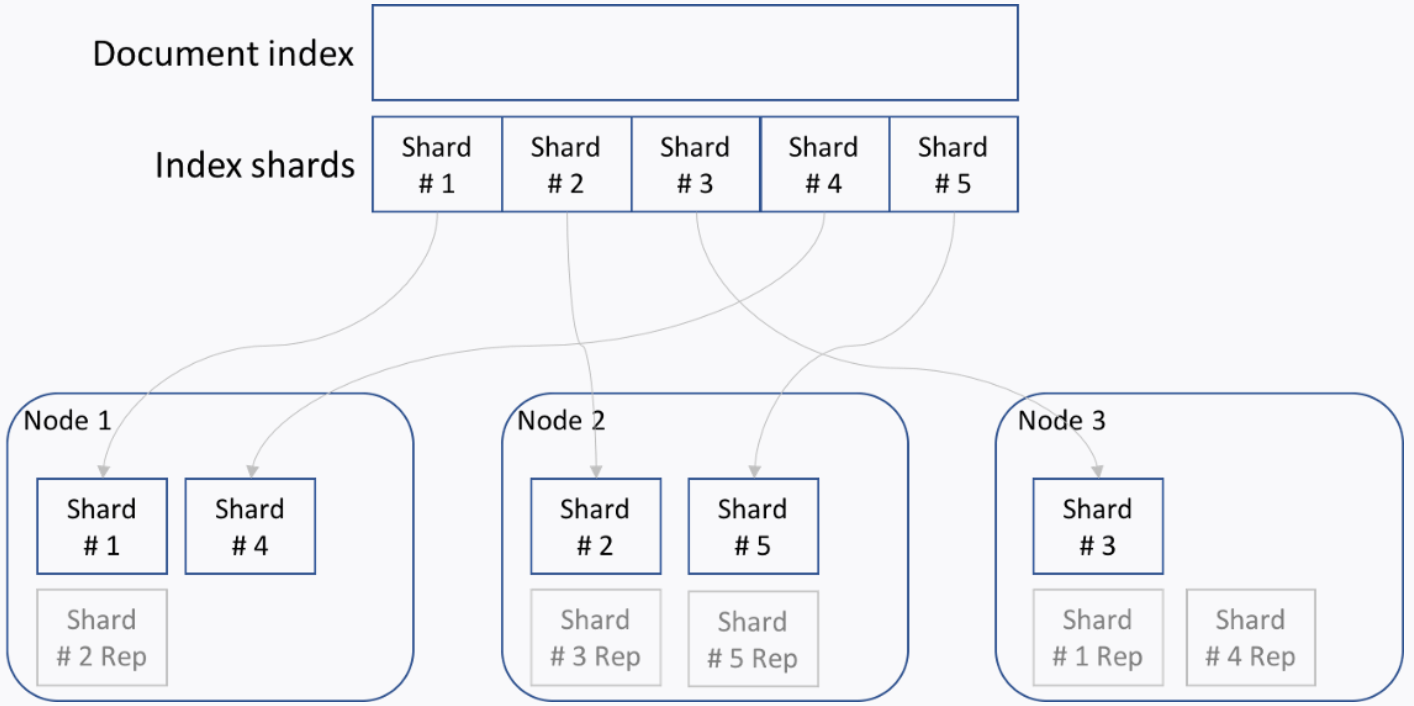


Elasticsearch и репликация

Replication factor – количество реплик шарда, которые мы храним

- *(primary + number_of_replicas) / 2 + 1*

Больше реплик – выше сохранность данных, выше производительность, больше потребляемых ресурсов.



Маршрутизация данных в Elasticsearch

- *Нодам назначаются метки*
  node.attr.size: medium
- *Индекс маршрутизируется согласно меткам*
  ```
  curl -X PUT "localhost:9200/test/_settings?pretty" -H 'Content-Type:
  application/json' -d' {"index.routing.allocation.include.size": "big,medium"}'
  ```

Горячие и холодные данные, разделяем ноды по ролям:

- Индексирующие ноды. Много CPU, памяти, быстрые SSD. [node.attr.data: hot]
- Хранилища холодных данных – много дискового пространства. [node.attr.data: cold]
- Как? Сначала в Logstash, потом - Curator & cron / ILM policy

```
curl -X PUT “localhost:9200/logs_2020-09-02” -H 'Content-Type: application/json’ -d
‘{"settings": {"index.routing.allocation.require.data": "hot"}}’

curl -X PUT “localhost:9200/logs_2020-08-02/settings” -H 'Content-Type:
application/json’ -d ‘{"settings": {"index.routing.allocation.require.data":
“cold"}}’
```

Как вообще распределять данные?

Масштабирование данных внутри кластера:

- **Количество документов на шард** определяет как долго будет выполняться запрос к шарду
- **Количество документов на ноду** определяет требование к памяти для ноды

- **Количество шардов** не меняет количество документов на ноду, но **уменьшает** количество документов на шард > время поиска.
- **Replication factor** не влияет на количество документов на ноду, повышает общую параллельность запросов, но увеличивает потребление ресурсов.
- **Количество нод** уменьшает количество документов на ноду и увеличивает масшрабируемость.

Метрики:

- **Shard search time** – время на поиск внутри шарда, зависит от размера шарда и типа запроса (term/filter)
- **# of concurrent searches** – количество параллельных запросов, выполняемых на одной ноде

20-25 шардов на 1Gb heap

Как вообще распределять данные?

	Scenario 1	More shards	More replicas	More nodes	More nodes & shards	More docs	More searches
# of documents	3,000,000	3,000,000	3,000,000	3,000,000	3,000,000	5,000,000	3,000,000
# of shards	10	**15**	10	10	15	**15**	15
# of replicas	2	2	**3**	2	2	2	2
# of cluster nodes	30	30	30	**40**	**40**	30	30
searches/second	2,500	2,500	2,500	2,500	2,500	2,500	**5,000**
search thread pool size	40	40	40	40	40	40	40
result consolidation time (ms)	3	3	3	3	3	3	3
Shard search time param: indep	3	3	3	3	3	3	3
Shard search time param: slope	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001
effective # of cluster nodes	30	30	30	30	40	30	30
docs per shard	300,000	200,000	300,000	300,000	200,000	333,333	200,000
shard search time (ms)	6.0	5.0	6.0	6.0	5.0	6.3	5.0
shards per node	1	1.5	1.333333333	1	1.125	1.5	1.5
docs per node	300,000	300,000	400,000	300,000	225,000	500,000	300,000
shard searches / second	25,000	37,500	25,000	25,000	37,500	37,500	75,000
shard searches / node / second	833	1,250	833	833	938	1,250	2,500
shard searches / thread / second	21	31	21	21	23	31	63
max searches per thread per second	167	200	167	167	200	158	200
average busy threads	30	31	30	30	23	50	63
average shard search wait time (ms)	0.0	0.0	0.0	0.0	0.0	1.6	2.8
estimated query response time (ms)	**9**	**8**	**9**	**9**	**8**	**25**	**36**

Общие советы по оптимизации

_source – хранит «сырые» данные в индексе, потребляя ресурсы, и обычно ненужно.

```
curl -X PUT "localhost:9200/my-index-000001?pretty" -H 'Content-Type: application/json' -d'
{"mappings": {"_source": {"enabled": false}}}’
```

path.data: /var/lib/elasticsearch/ssd1, /var/lib/elasticsearch/ssd2 – эффективнее отдавать несколько независимых дисков под хранение

processors: при большом количестве ядер elasticsearch может неправильно определить их количество

bootstrap.mlockall: true - отключаем своп

discovery.zen.ping.multicast.enabled : false - отключаем мультикаст

discovery.zen.ping.unicast.hosts: [“node1”, “node2”, “node3”] – добавляем ноды в кластер статически

cluster.routing.allocation.cluster_concurrent_rebalance: 1 – указываем количество одновременно ребалансящихся шардов. Зависит от кейса, но обычно лучше – меньше.

cluster.routing.allocation.disk.watermark.low: 85% - заполнение диска, после которого данные не будут приезжать на ноду

cluster.routing.allocation.disk.watermark.low: 90% - заполнение диска, после которого шарды будут распределяться на другие ноды

Шаблоны индексов и предварительное создание

```
PUT _index_template/template_1
{
  "index_patterns": ["logstash-*"],
  "template": {
    "settings": {
      "number_of_shards": 10
    },
    "mappings": {
      "_source": {
        "enabled": false
      },
      "properties": {
        "host_name": {
          "type": "keyword"
        },
        "created_at": {
          "type": "date",
          "format": "EEE MMM dd HH:mm:ss Z yyyy"
        },
        "ip_address": {
          "type": "ip"
        },
        "@timestamp": {
          "type": "date"
        }
      }
    },
    "aliases": {
      "mydata": { }
    }
  },
  "priority": 10,
  "version": 3,
  "_meta": {
    "description": "my template"
  }
}
```

Предварительное создание индексов
https://github.com/adel-s/ElasticSearch/blob/master/es_precreate_indices.py

Elasticsearch maintenance

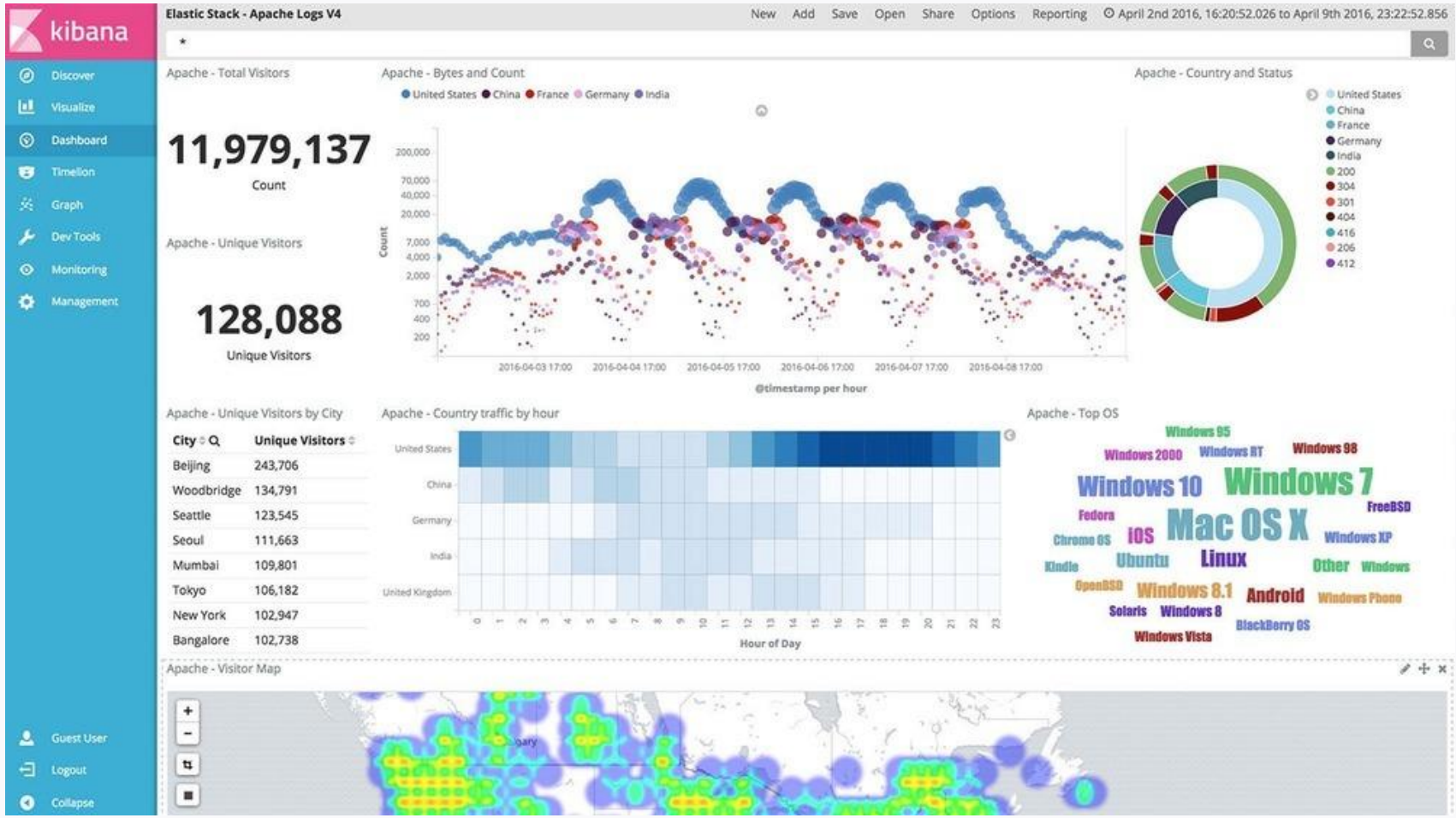
```
curl -XPUT 'http://localhost:9200/_cluster/settings' -d '{
"transient" : {
    "cluster.routing.rebalance.enable" : "none“
    "cluster.routing.allocation.enable" : "none"
    "cluster.routing.allocation.exclude._ip" : “10.10.10.10"
}}’

Curl 'http://localhost:9200/_cluster/settings' -d ‘{
  "transient" : {
     "cluster.routing.rebalance.enable" : "all“
     "cluster.routing.allocation.enable" : “all"
}}’
```

Kibana

Secure and balance:

```
console.enabled: false
elasticsearch.username: user
elasticsearch.password: pass
elasticsearch.hosts:
- http://client-01:9200
- http://client-02:9200
- http://client-03:9200
```



Рефлексия

С какими основными мыслями и инсайтами уходите с вебинара?

Достигли ли вы цели вебинара?

Список материалов для изучения

- Книга
- Сайт
- Мануал
- Статья
- Видео
- Приложение/Сервис
- Пример кода/конфига и др. на github OTUS
- …

Студенты рады, когда к занятию есть что почитать и изучить дополнительно, кроме вебинара


Заполните, пожалуйста,
опрос о занятии по ссылке в чате

Спасибо за внимание!
Приходите на следующие вебинары

фото

Напишите ваше ФИО

Должность
Компанию
Контакты