

ELK日志分析系统 20:30开始

讲师：君临天下

12月26日周末班12月21日全日制班欢迎您的到来！

需要代码、PPT、视频等资料请加以下几位老师

QQ:

贾老师：1786418286

何老师：1926106490

詹老师：2805048645

讨论技术可以加入以下QQ群： 172599077 ,
156927834

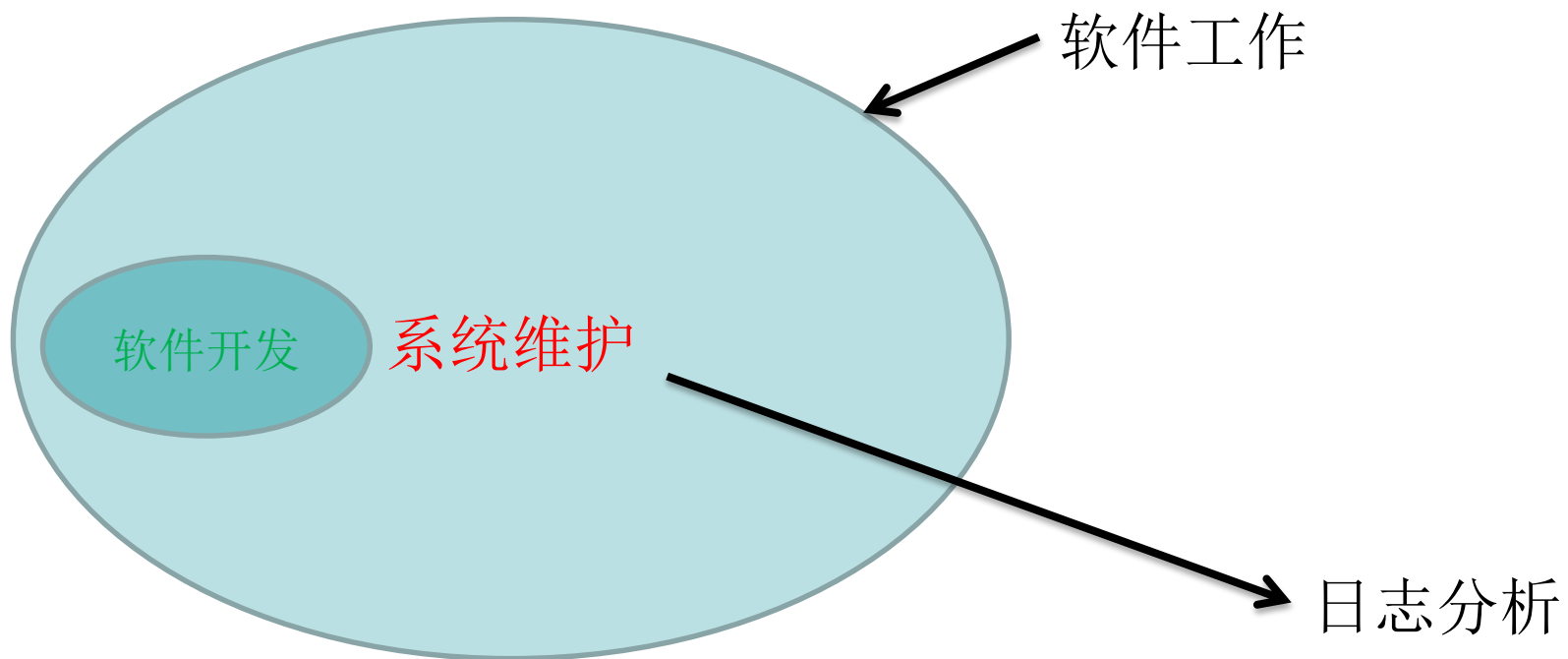
春节前最后一期班级 16年1月1日学费上调，提前报名预订座位，不管何时过来学习费用以报名时费用为准。

年后开班时间

2016.2.26日脱产班

2016.3.12日周末班

2016.3.19日线上周末班



- 一般日志分析遇到的问题：
 - 日志太多不好定位异常
 - 通过日志分析一些数据（如pv、uv）的时候过程太麻烦

思考：有没有一种框架以最少的开发，得到最灵活的日志分析功能？

- ELK就是一套完整的日志分析系统
- ELK=**Logstash+Elasticsearch+Kibana**
- 统一官网<https://www.elastic.co/products>



- **Logstash**

- 用于处理传入的日志，负责收集、过滤和写出日志

- **安装Logstash**

- 下载解压即完成
- Logstash的hallo word

- **Logstash的输入**

- 标准输入
- 文件输入
- TCP输入
- Syslog输入
- Collectd输入

- **Logstash的过滤器配置**

- Date时间处理
- Grok正则捕获
- GeoIP地址查询
- Json编解码
-

- **Logstash的输出**

- elasticserch
- 发送到email
- 保存为文件
- 输出到HDFS
- 标准输出
- TCP发送数据
-

- **Logstash的配置文件**

- 以花括号来分块
- 输出配置Input块
- 过滤器配置Filter
- 输出配置output

- Logstash的配置文件

```
input {
  file {
    path => "/opt/sxt/soft/hadoop-2.7.1/logs/hadoop-root-journalnode-master.log"
    start_position => beginning
  }
}
filter {
  grok {
    match => {
      "message" => "(?m)%{TIMESTAMP_ISO8601:date} %{WORD:log_type} %{DATA:classPath}: %{DATA:data}"
    }
  }
  date {
    match => [ "date", "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}
output {
  elasticsearch {
    hosts => ["master", "slave1"]
  }
  stdout { codec => rubydebug }
}
~
```

ELK日志分析系统

- **Logstash的配置文件**

- input {
- file { #通过Input配置，从文件中读取数据。
- path => "/tmp/access_log" #日志文件位置
- start_position => "beginning"
- #是否从头部开始读取。
- #Logstash启动后，会在系统中记录一个隐藏文件，记录处理过的行号，
- #当进行挂掉，重新启动后，根据该行号记录续读。
- #所以start_position只会生效一次。
- }
- }
-

- **Logstash的配置文件**

- filter {
- **if** [path] =~ "access" {#当路径包含access时，才会执行以下处理逻辑
- mutate { replace => { "type" => "apache_access" } }
- grok {
- match => { "message" => "%{COMBINEDAPACHELOG}" }
- }
- **else if** [path] =~ "error" { #IF-ELSE 配置方式。
- mutate { replace => { type => "apache_error" } }
- #使用Mutate 替换type的值为 "apache_error"
- **else** {
- mutate { replace => { type => "random_logs" } }
- }
- date {
- match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
- }
- }

- **Logstash的配置文件**

- output {
- elasticsearch {
- host => localhost
- }
- stdout { codec => rubydebug }
- }

ELK日志分析系统

• Grok正则捕获

- 基于正则表达式的匹配，完全支持正则表达式
- 增加了一百多个内置变量
- 调用%{USER: user}
- 匹配上的内容放变量中

```

USERNAME [a-zA-Z0-9_-]+
USER %{USERNAME}
INT (?:[+-]?(?:[0-9]+))
BASE10NUM (?<![0-9.+-]) (?>[+-]? (?:(?:[0-9]+(?!\. [0-9]+) ?) | (?
NUMBER (?:%{BASE10NUM})
BASE16NUM (?<![0-9A-Fa-f]) (?:[+-]? (?:(?:[0-9A-Fa-f]+) ?)
BASE16FLOAT \b(?<![0-9A-Fa-f.]) (?:[+-]? (?:(?:[0-9A-Fa-f.]+) ?)

POSINT \b(?:[1-9][0-9]*)\b
NONNEGINT \b(?:[0-9]+)\b
WORD \b\w+\b
NOTSPACE \S+
SPACE \s*
DATA .*?
GREEDYDATA .*
#QUOTEDSTRING (?:(?!\\" ) (?:"(?:\\" . | [^\\""])"*) | (?:'(?:\\" . | [^\\"']
QUOTEDSTRING (?>(?!\" ) (?>"(?:\\" . | [^\\""])"*) | "" | (?>'(?:\\" . | [^\\"']
UUID [A-Fa-f0-9]{8}-(?:[A-Fa-f0-9]{4}-){3}[A-Fa-f0-9]{12}

# Networking
MAC (?:%{CISCOMAC}|%{WINDOWSMAC}|%{COMMONMAC})
CISCOMAC (?:(?:[A-Fa-f0-9]{4}\.){2}[A-Fa-f0-9]{4})
WINDOWSMAC (?:(?:[A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2})
COMMONMAC (?:(?:[A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2})
    
```

- Grok
 - 匹配上的内容放变量中，那么原消息呢？

答案是也存在

可以用remove_field或者overwrite来重写message，比如
Overwrite => ["message"]

- Date
 - 将日志中的时间字符串转成默认的LogStash的Timestamp对象转存到@timestamp
 - 不使用的默认是logstash读取该日志的时间

- geoip
 - 免费的IP地址归类查询库，可以根据ip地址查询对应地域的信息，包括国别，省市，经纬度等

- Logstash支持的操作符

You can use the following comparison operators:

- equality: `==`, `!=`, `<`, `>`, `<=`, `>=`
- regexp: `=~`, `!~`
- inclusion: `in`, `not in`

The supported boolean operators are:

- `and`, `or`, `nand`, `xor`

The supported unary operators are:

- `!`

- Logstash支持的操作符

You can use the following comparison operators:

- equality: `==`, `!=`, `<`, `>`, `<=`, `>=`
- regexp: `=~`, `!~`
- inclusion: `in`, `not in`

The supported boolean operators are:

- `and`, `or`, `nand`, `xor`

The supported unary operators are:

- `!`

- Logstash常用输入输出
- 输入：
 - 日志文件，redis，kafka，指定端口
- 输出：
 - Elasticsearch，hdfs，redis，kafka

- **elasticsearch**

- 用于将导入数据建立动态倒排索引，建立磁盘缓存，提供磁盘同步控制，达到准实时检索

- **Elasticsearch数据流向**

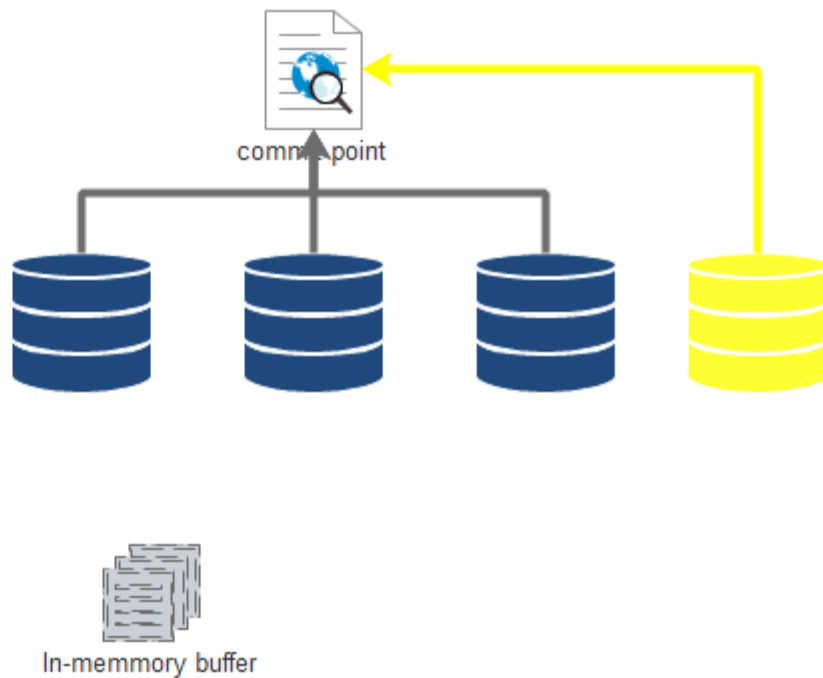
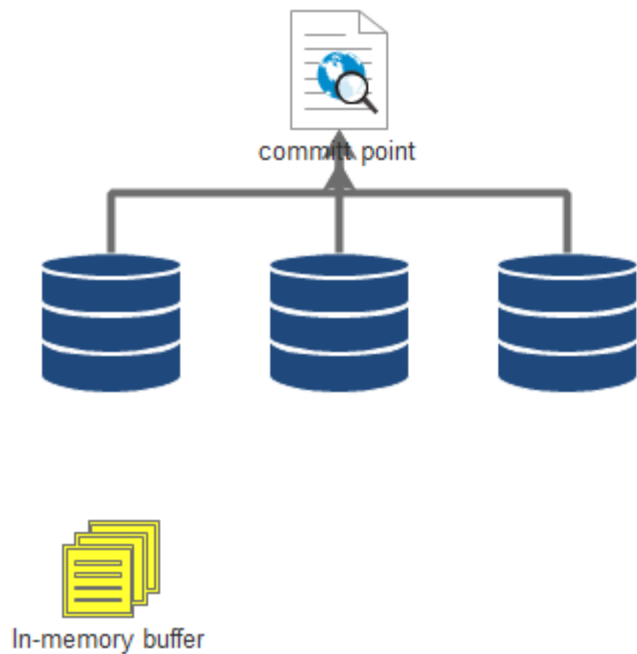
思考：写入的数据是如何变成elasticsearch里面可以被检索和聚合的索引内容呢？

- **Elasticsearch数据流向**

- 动态更新lucene索引，规则：**新收到的数据写入到新的索引文件里面**

- **步骤：**

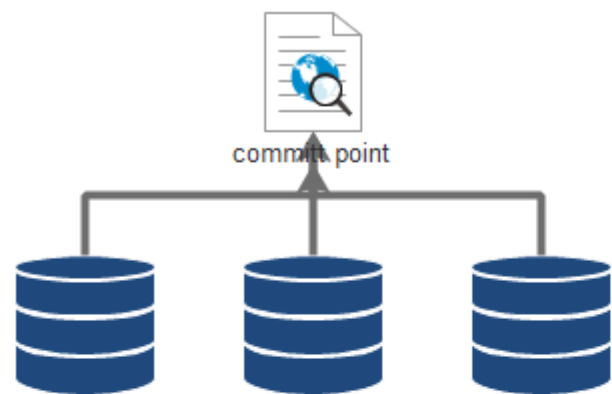
- 每次生成的倒排索引叫一个段（segment）然后另外使用一个commit文件记录索引内所有的segment，生成segment的数据来源是内存buffer




默认每隔一秒刷一次到文件系统缓存，文件系统缓存再到磁盘，可以调用 `/_refresh` 手动刷新

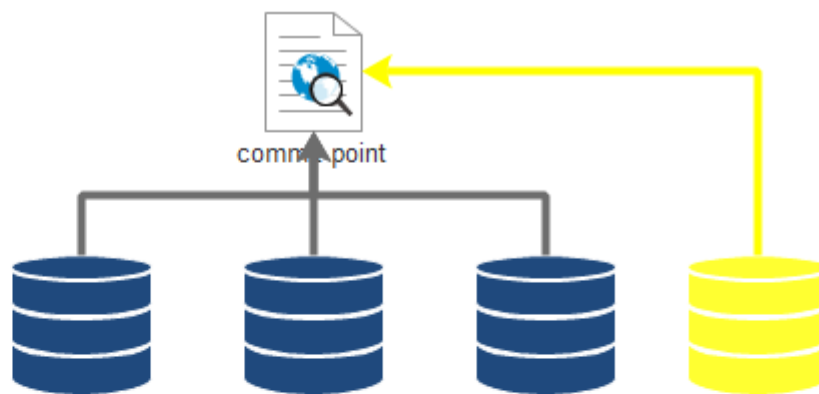
- **思考：既然每隔一秒只是写到文件系统缓存，那么最后一步写到实际磁盘是什么来控制的？如果中间出现主机错误、硬件故障等异常，数据会不会丢失呢？**

- **Translog提供磁盘同步控制**
 - 数据写入内存buffer同时记录了一个translog日志





In-memory buffer


Translog




In-memory buffer


Translog

- **Translog提供磁盘同步控制**

- 等到commit文件更新的时候，translog才清空，这一步叫做flush，默认每半小时刷新一次，也可以手动调用
- 也可以通过配置index.translog.flush_threshold_ops参数，控制每多少条刷新一次

- **Segment merge**
 - 独立线程做merge工作



- **Segment merge**

- 可以通过`indices.store.throttle.max_bytes_per_sec`设置调整速度限制，比如SSD可以调整到100mb
- 线程数公式`Math.min (3,Runtime.getRuntime().availableProcessors()/2)`
- 也可以调整`index.merge.scheduler.max_thread_count`来配置

- **Elasticsearch XPUT**

- 动态修改配置
- **Curl -XPUT http://127.0.0.1:9200/_cluster/settings -d ‘**
- **{**
- **“persistent” :{**
- **“indices.store.throttle.max_bytes_per_sec” : “100mb”**
- **}**
- **}**