# 开源实时日志分析平台ELK

## 学习目标：

1、了解ELK平台的作用

2、掌握ElasticSearch部署及监控

3、掌握LogStash部署及应用

4、掌握Kibana部署及应用

5、掌握filebeat部署及应用

6、掌握Kafka集群部署及应用

在日常运维过程中，运维人员需要对系统和业务日志进行精准把控，便于分析系统和业务状态，做出合理的运维预估，减少企业损失。

日志主要包括系统日志、应用程序日志和安全日志。系统运维和开发人员可以通过日志了解服务器软、硬件信息、检查配置过程中的错误及错误发生的原因。经常分析日志可以了解服务器的负荷，性能安全性，从而及时采取措施纠正错误。

通常，日志被分散在储存不同的设备上。如果你管理数十上百台服务器，你还在使用依次登录每台机器的传统方法查阅日志。这样是不是感觉很繁琐和效率低下。当务之急我们使用集中化的日志管理，例如：开源的rsyslog，将所有服务器上的日志收集汇总。

集中化管理日志后，日志的统计和检索又成为一件比较麻烦的事情，一般我们使用grep、awk和wc等Linux命令能实现检索和统计，但是对于要求更高的查询、排序和统计等要求和庞大的机器数量依然使用这样的方法难免有点力不从心。

通过我们需要对日志进行集中化管理，将所有机器上的日志信息收集、汇总到一起。完整的日志数据具有非常重要的作用：

1）信息查找。通过检索日志信息，定位相应的bug，找出解决方案。

2）服务诊断。通过对日志信息进行统计、分析，了解服务器的负荷和服务运行状态，找出耗时请求进行优化等等。

3）数据分析。如果是格式化的log，可以做进一步的数据分析，统计、聚合出有意义的信息，比如根据请求中的商品id，找出TOP10用户感兴趣商品。

4)安全提示。随着互联网安全性意识越来越高，通过分析日志可以了解服务器或网络的安全状态，为网络安全决策者提供安全类方案的依据。

开源实时日志分析ELK平台能够完美的解决我们上述的问题，ELK由ElasticSearch、Logstash和Kiabana三个开源工具组成：

1）Elastic Search是一个基于Lucene的开源分布式搜索服务器。它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，RESTful风格接口，多数据源，自动搜索负载等。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是非常流行的企业搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。

在elasticsearch中，所有节点的数据是均等的。

2）Logstash是一个完全开源的工具，它可以对你的日志进行收集、过滤、分析，支持大量的数据获取方法，并将其存储供以后使用（如搜索）。说到搜索，logstash带有一个web界面，搜索和展示所有日志。一般工作方式为c/s架构，client端安装在需要收集日志的主机上，server端负责将收到的各节点日志进行过滤、修改等操作在一并发往elasticsearch上去。 input filter out

3）Kibana 是一个基于浏览器页面的Elasticsearch前端展示工具，也是一个开源和免费的工具，Kibana可以为 Logstash 和 ElasticSearch 提供的日志分析友好的 Web 界面，可以帮助您汇总、分析和搜索重要数据日志。

一般我们需要进行日志分析场景是：直接在日志文件中 grep、awk 就可以获得自己想要的信息。但在规模较大的场景中，此方法效率低下，面临问题包括日志量太大如何归档、文本搜索太慢怎么办、如何多维度查询。需要集中化的日志管理，所有服务器上的日志收集汇总。常见解决思路是建立集中式日志收集系统，将所有节点上的日志统一收集，管理，访问。

一般大型系统是一个分布式部署的架构，不同的服务模块部署在不同的服务器上，问题出现时，大部分情况需要根据问题暴露的关键信息，定位到具体的服务器和服务模块，构建一套集中式日志系统，可以提高定位问题的效率。

一个完整的集中式日志系统，需要包含以下几个主要特点：

1）收集－能够采集多种来源的日志数据

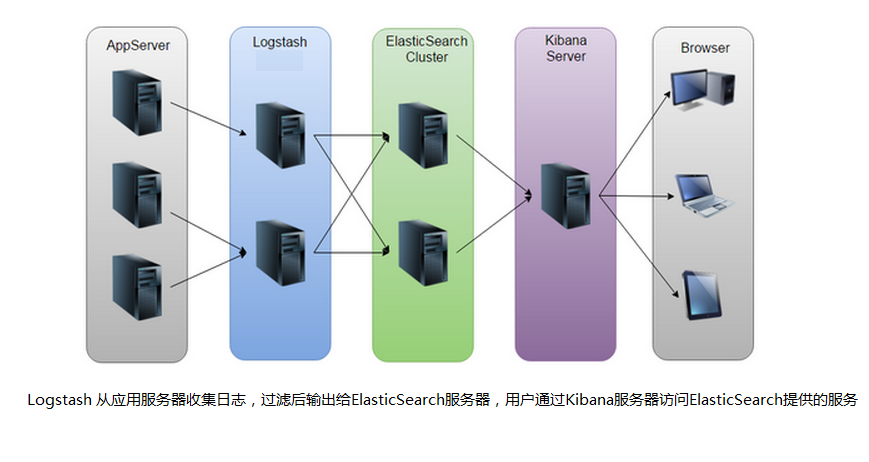
2）传输－能够稳定的把日志数据传输到中央系统

3）存储－如何存储日志数据

4）分析－可以支持 UI 分析

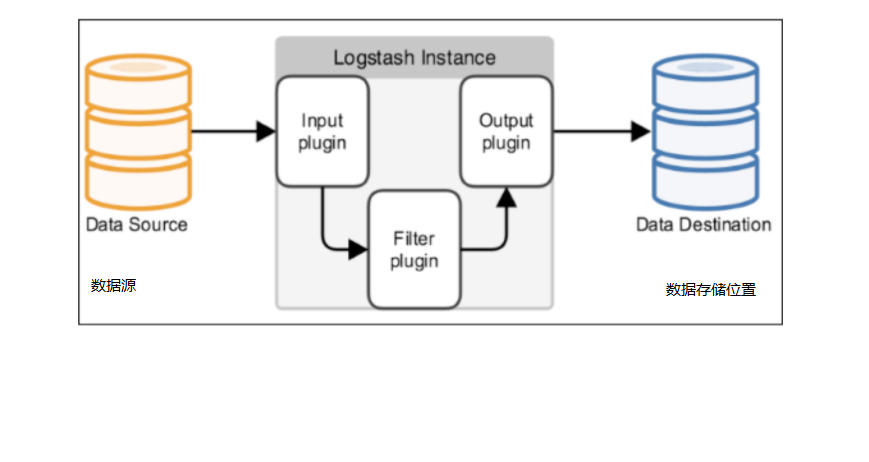
5）警告－能够提供错误报告，监控机制

ELK提供了一整套解决方案，并且都是开源软件，之间互相配合使用，完美衔接，高效的满足了很多场合的应用。目前主流的一种日志系统。



Logstash工作原理

Logstash事件处理有三个阶段：inputs → filters → outputs。是一个接收，处理，转发日志的工具。支持系统日志，webserver日志，错误日志，应用日志，总之包括所有可以抛出来的日志类型。



Logstash组成部分：

Input：输入到Logstash

一些常用的输入为：

file：从文件系统的文件中读取，类似于tial -f命令

rsyslog：在tcp/udp 514端口上监听系统日志消息，并根据RFC3164标准进行解析

redis：从redis service中读取

beats：从filebeat中读取

filters：数据中间处理，对数据进行操作

kafka:从kafka集群中读取

Filter：在Logstash服务器上对数据进行过滤

一些常用的过滤器为：

grok：解析任意文本数据，Grok 是 Logstash 最重要的插件。它的主要作用就是将文本格式的字符串，转换成为具体的结构化的数据，配合正则表达式使用。内置120多个解析语法。

mutate：对字段进行转换。例如对字段进行删除、替换、修改、重命名等。

drop：丢弃一部分events不进行处理。

clone：拷贝 event，这个过程中也可以添加或移除字段。

geoip：添加地理信息(为前台kibana图形化展示使用)

Output：output是logstash处理管道的最末端组件。一个event可以在处理过程中经过多重输出，但是一旦所有的outputs都执行结束，这个event也就完成生命周期。

一些常见的output为：

elasticsearch：可以高效的保存数据，并且能够方便和简单的进行查询。

file：将event数据保存到文件中。

graphite：将event数据发送到图形化组件中，一个很流行的开源存储图形化展示的组件。

Codecs：codecs 是基于数据流的过滤器，它可以作为input，output的一部分配置。

Codecs：可以帮助你轻松的分割发送过来已经被序列化的数据。

一些常见的codecs：

json：使用json格式对数据进行编码/解码。

multiline：将汇多个事件中数据汇总为一个单一的行。比如：java异常信息和堆栈信息。

ELK中的三个系统分别扮演不同的角色，组成了一个整体的解决方案。

Logstash是一个ETL工具，负责从每台机器抓取日志数据，对数据进行格式转换和处理后，输出到Elasticsearch中存储。

Elasticsearch是一个分布式搜索引擎和分析引擎，用于数据存储，可提供实时的数据查询。

Kibana是一个数据可视化服务，根据用户的操作从Elasticsearch中查询数据，形成相应的分析结果，以图表的形式展现给用户。

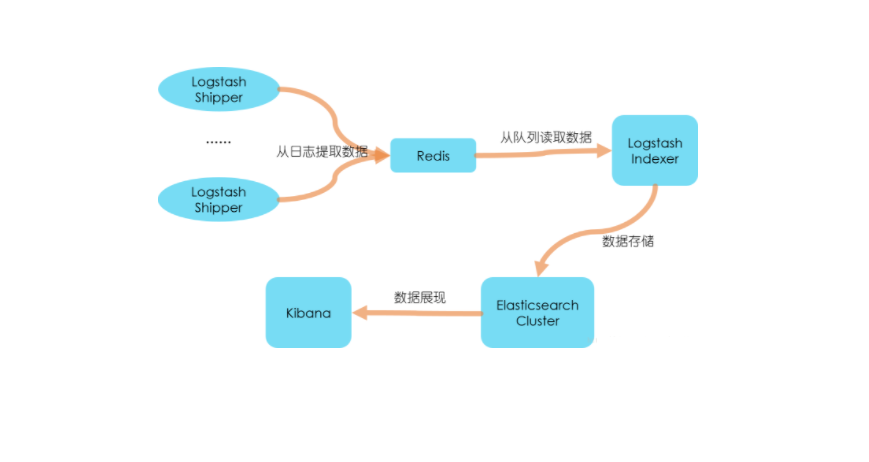
ELK的安装很简单，可以按照"下载->修改配置文件->启动"方法分别部署三个系统，也可以使用docker来快速部署。具体的安装方法这里不详细介绍，下面来看一个常见的部署方案，如下图所示，部署思路是：

1）在每台生成日志文件的机器上，部署Logstash，作为Shipper的角色，负责从日志文件中提取数据，但是不做任何处理，直接将数据输出到Redis队列(list)中；

2）需要一台机器部署Logstash，作为Indexer的角色，负责从Redis中取出数据，对数据进行格式化和相关处理后，输出到Elasticsearch中存储；

3）部署Elasticsearch集群，当然取决于你的数据量了，数据量小的话可以使用单台服务，如果做集群的话，最好是有3个以上节点，同时还需要部署相关的监控插件；

4）部署Kibana服务，提供Web服务。



在前期部署阶段，主要工作是Logstash节点和Elasticsearch集群的部署，而在后期使用阶段，主要工作就是Elasticsearch集群的监控和使用Kibana来检索、分析日志数据了，当然也可以直接编写程序来消费Elasticsearch中的数据。

在上面的部署方案中，我们将Logstash分为Shipper和Indexer两种角色来完成不同的工作，中间通过Redis做数据管道，为什么要这样做？为什么不是直接在每台机器上使用Logstash提取数据、处理、存入Elasticsearch？

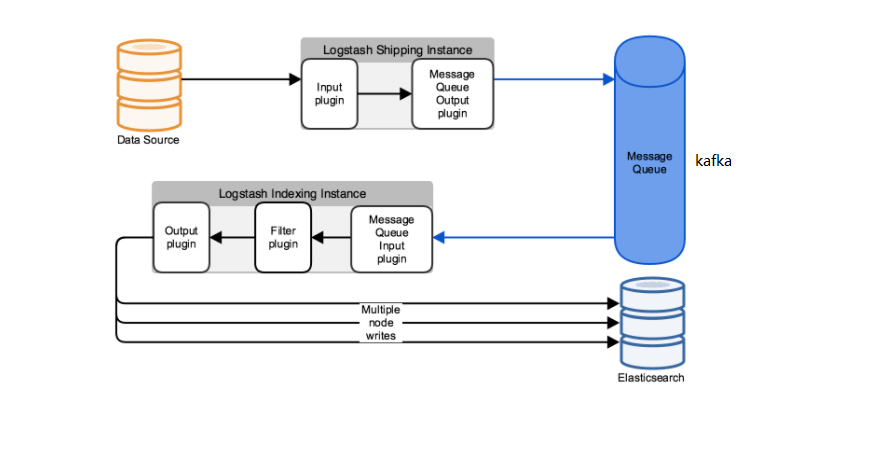
首先，采用这样的架构部署，有三点优势：

第一，降低对日志所在机器的影响，这些机器上一般都部署着反向代理或应用服务，本身负载就很重了，所以尽可能的在这些机器上少做事；

第二，如果有很多台机器需要做日志收集，那么让每台机器都向Elasticsearch持续写入数据，必然会对Elasticsearch造成压力，因此需要对数据进行缓冲，同时，这样的缓冲也可以一定程度的保护数据不丢失；

第三，将日志数据的格式化与处理放到Indexer中统一做，可以在一处修改代码、部署，避免需要到多台机器上去修改配置。

其次，我们需要做的是将数据放入一个消息队列中进行缓冲，所以Redis只是其中一个选择，也可以是RabbitMQ、Kafka等等，在实际生产中，Redis与Kafka用的比较多。由于Redis集群一般都是通过key来做分片，无法对list类型做集群，在数据量大的时候必然不合适了，而Kafka天生就是分布式的消息队列系统。



1）配置nginx日志格式

首先需要将nginx日志格式规范化，便于做解析处理。在nginx.conf文件中设置：

**[**root**@**localhost **~]**#vim **/**etc**/**nginx**/**nginx.conf

log\_format main '$remote\_addr "$time\_iso8601" "$request" $status $body\_bytes\_sent "$http\_user\_agent" "$http\_referer" "$http\_x\_forwarded\_for" "$request\_time" "$upstream\_response\_time" "$http\_cookie" "$http\_Authorization" "$http\_token"'**;**

access\_log **/**var**/**log**/**nginx**/**example.access.log main**;**

2）nginx日志–>>Logstash–>>消息队列

这部分是Logstash Shipper的工作，涉及input和output两种插件。input部分，由于需要提取的是日志文件，一般使用file插件，该插件常用的几个参数是：

path：指定日志文件路径。

type：指定一个名称，设置type后，可以在后面的filter和output中对不同的type做不同的处理，适用于需要消费多个日志文件的场景。

start\_position：指定起始读取位置，“beginning”表示从文件头开始，“end”表示从文件尾开始（类似tail -f）。

sincedb\_path：与Logstash的一个坑有关。通常Logstash会记录每个文件已经被读取到的位置，保存在sincedb中，如果Logstash重启，那么对于同一个文件，会继续从上次记录的位置开始读取。如果想重新从头读取文件，需要删除sincedb文件，sincedb\_path则是指定了该文件的路径。为了方便，我们可以根据需要将其设置为“/dev/null”，即不保存位置信息。

input **{**

**file** **{**

**type** **=>** "example\_nginx\_access"

path **=>** **[**"/var/log/nginx/example.access.log"**]**

start\_position **=>** "beginning"

sincedb\_path **=>** "/dev/null"

**}**

**}**

output部分，将数据输出到消息队列，以redis为例，需要指定redis server和list key名称。另外，在测试阶段，可以使用stdout来查看输出信息。

# 输出到redis

output **{**

**if** **[**type**]** **==** "example\_nginx\_access" **{**

redis **{**

host **=>** "127.0.0.1"

port **=>** "6379"

data\_type **=>** "list"

key **=>** "logstash:example\_nginx\_access"

**}**

# stdout {codec => rubydebug}

**}**

**}**

3）消息队列–>>Logstash–>>Elasticsearch

这部分是Logstash Indexer的工作，涉及input、filter和output三种插件。在input部分，我们通过redis插件将数据从消息队列中取出来。在output部分，我们通过elasticsearch插件将数据写入Elasticsearch。

# 从redis输入数据

input **{**

redis **{**

host **=>** "127.0.0.1"

port **=>** "6379"

data\_type **=>** "list"

key **=>** "logstash:example\_nginx\_access"

**}**

**}**

output **{**

elasticsearch **{**

index **=>** "logstash-example-nginx-%{+YYYY.MM.dd}"

hosts **=>** **[**"127.0.0.1:9200"**]**

**}**

**}**

这里，需要重点关注filter部分，下面列举几个常用的插件，实际使用中根据自身需求从官方文档中查找适合自己业务的插件并使用即可，当然也可以编写自己的插件。

grok：是Logstash最重要的一个插件，用于将非结构化的文本数据转化为结构化的数据。grok内部使用正则语法对文本数据进行匹配，为了降低使用复杂度，其提供了一组pattern，我们可以直接调用pattern而不需要自己写正则表达式，参考源码grok-patterns。grok解析文本的语法格式是%{SYNTAX:SEMANTIC}，SYNTAX是pattern名称，SEMANTIC是需要生成的字段名称，使用工具Grok Debugger可以对解析语法进行调试。例如，在下面的配置中，我们先使用grok对输入的原始nginx日志信息（默认以message作为字段名）进行解析，并添加新的字段request\_path\_with\_verb（该字段的值是verb和request\_path的组合），然后对request\_path字段做进一步解析。

kv：用于将某个字段的值进行分解，类似于编程语言中的字符串Split。在下面的配置中，我们将request\_args字段值按照“&”进行分解，分解后的字段名称以“request\_args\_”作为前缀，并且丢弃重复的字段。

geoip：用于根据IP信息生成地理位置信息，默认使用自带的一份GeoLiteCity database，也可以自己更换为最新的数据库，但是需要数据格式需要遵循Maxmind的格式（参考GeoLite），似乎目前只能支持legacy database，数据类型必须是.dat。下载GeoLiteCity.dat.gz后解压， 并将文件路径配置到source中即可。

translate：用于检测某字段的值是否符合条件，如果符合条件则将其翻译成新的值，写入一个新的字段，匹配pattern可以通过YAML文件来配置。例如，在下面的配置中，我们对request\_api字段翻译成更加易懂的文字描述。

filter **{**

grok **{**

match **=>** **{**"message" **=>** "%{IPORHOST:client\_ip} \"%{TIMESTAMP\_ISO8601:timestamp}\" \"%{WORD:verb} %{NOTSPACE:request\_path} HTTP/%{NUMBER:httpversion}\" %{NUMBER:response\_status:int} %{NUMBER:response\_body\_bytes:int} \"%{DATA:user\_agent}\" \"%{DATA:http\_referer}\" \"%{NOTSPACE:http\_x\_forwarder\_for}\" \"%{NUMBER:request\_time:float}\" \"%{DATA:upstream\_resopnse\_time}\" \"%{DATA:http\_cookie}\" \"%{DATA:http\_authorization}\" \"%{DATA:http\_token}\""**}**

add\_field **=>** **{**"request\_path\_with\_verb" **=>** "%{verb} %{request\_path}"**}**

**}**

grok **{**

match **=>** **{**"request\_path" **=>** "%{URIPATH:request\_api}(?:\?%{NOTSPACE:request\_args}|)"**}**

add\_field **=>** **{**"request\_annotation" **=>** "%{request\_api}"**}**

**}**

kv **{**

prefix **=>** "request\_args\_"

field\_split **=>** "&"

source **=>** "request\_args"

allow\_duplicate\_values **=>** false

**}**

geoip **{**

source **=>** "client\_ip"

database **=>** "/home/elktest/geoip\_data/GeoLiteCity.dat"

**}**

translate **{**

field **=>** request\_path

destination **=>** request\_annotation

regex **=>** true

exact **=>** true

dictionary\_path **=>** "/home/elktest/api\_annotation.yaml"

override **=>** true

**}**

**}**

ElasticSearch工作原理

Elasticsearch承载了数据存储和查询的功能，这里主要介绍些实际生产中的问题和方法：

1）关于集群配置，重点关注三个参数：

第一，discovery.zen.ping.unicast.hosts，Elasticsearch默认使用Zen Discovery来做节点发现机制，推荐使用unicast来做通信方式，在该配置项中列举出Master节点。

第二，discovery.zen.minimum\_master\_nodes，该参数表示集群中可工作的具有Master节点资格的最小数量，默认值是1。为了提高集群的可用性，避免脑裂现象（所谓脑裂，就是同一个集群中的不同节点，对集群的状态有不一致的理解。），官方推荐设置为(N/2)+1，其中N是具有Master资格的节点的数量。

第三，discovery.zen.ping\_timeout，表示节点在发现过程中的等待时间，默认值是3秒，可以根据自身网络环境进行调整，一定程度上提供可用性。

**[**root**@**localhost **~]**#vim **/**etc**/**elasticsearch**/**elasticsearch.yml

discovery.zen.ping.unicast.hosts**:** **[**"master1"**,** "master2"**,** "master3"**]**

discovery.zen.minimum\_master\_nodes**:** 2

discovery.zen.ping\_timeout**:** 10

2）关于集群节点，第一，节点类型包括：候选Master节点、数据节点和Client节点。通过设置两个配置项node.master和node.data为true或false，来决定将一个节点分配为什么类型的节点。第二，尽量将候选Master节点和Data节点分离开，通常Data节点负载较重，需要考虑单独部署。

3）关于内存，Elasticsearch默认设置的内存是1GB，对于任何一个业务部署来说，这个都太小了。通过指定ES\_HEAP\_SIZE环境变量，可以修改其堆内存大小，服务进程在启动时候会读取这个变量，并相应的设置堆的大小。建议设置系统内存的一半给Elasticsearch，但是不要超过32GB。参考官方文档。

4）关于硬盘空间，Elasticsearch默认将数据存储在/var/lib/elasticsearch路径下，随着数据的增长，一定会出现硬盘空间不够用的情形，此时就需要给机器挂载新的硬盘，并将Elasticsearch的路径配置到新硬盘的路径下。通过“path.data”配置项来进行设置，比如“path.data: /data1,/var/lib/elasticsearch,/data”。需要注意的是，同一分片下的数据只能写入到一个路径下，因此还是需要合理的规划和监控硬盘的使用。

5）关于Index的划分和分片的个数，这个需要根据数据量来做权衡了，Index可以按时间划分，比如每月一个或者每天一个，在Logstash输出时进行配置，shard的数量也需要做好控制。

6）关于监控，笔者使用过head和marvel两个监控插件，head免费，功能相对有限，marvel现在需要收费了。另外，不要在数据节点开启监控插件。

Kibana工作原理

Kibana提供的是数据查询和显示的Web服务，有丰富的图表样板，能满足大部分的数据可视化需求，这也是很多人选择ELK的主要原因之一。UI的操作没有什么特别需要介绍的，经常使用就会熟练，这里主要介绍经常遇到的三个问题。

a）查询语法

在Kibana的Discover页面中，可以输入一个查询条件来查询所需的数据。查询条件的写法使用的是Elasticsearch的Query String语法，而不是Query DSL，参考官方文档query-string-syntax，这里列举其中部分常用的：

.单字段的全文检索，比如搜索args字段中包含first的文档，写作 args:first；

.单字段的精确检索，比如搜索args字段值为first的文档，写作 args: “first”；

.多个检索条件的组合，使用 NOT, AND 和 OR 来组合，注意必须是大写，比如 args:(“first” OR “second”) AND NOT agent: “third”；

.字段是否存在，\_exists\_:agent表示要求agent字段存在，\_missing\_:agent表示要求agent字段不存在；

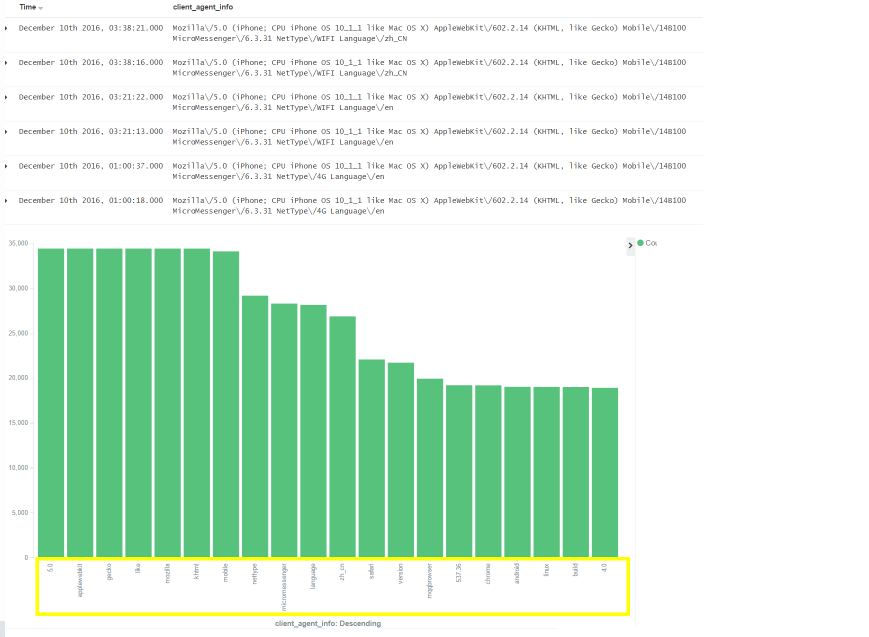
.通配符：用 ? 表示单字母，\* 表示任意个字母。

b）错误“Discover: Request Timeout after 30000ms”

这个错误经常发生在要查询的数据量比较大的情况下，此时Elasticsearch需要较长时间才能返回，导致Kibana发生Timeout报错。解决这个问题的方法，就是在Kibana的配置文件中修改elasticsearch.requestTimeout一项的值，然后重启Kibana服务即可，注意单位是ms。

c）疑惑“字符串被分解了”

经常碰到这样一个问题：为什么查询结果的字段值是正确的，可是做图表时却发现字段值被分解了，不是想要的结果？如下图所示的client\_agent\_info字段。



得到这样一个不正确结果的原因是使用了Analyzed字段来做图表分析，默认情况下Elasticsearch会对字符串数据进行分析，建立倒排索引，所以如果对这么一个字段进行terms聚合，必然会得到上面所示的错误结果了。那么应该怎么做才对？默认情况下，Elasticsearch还会创建一个相对应的没有被Analyzed的字段，即带“.raw”后缀的字段，在这样的字段上做聚合分析即可。

又会有很多人问这样的问题：为什么我的Elasticsearch没有自动创建带“.raw”后缀的字段？然而在Logstash中输出数据时，设置index名称前缀为“logstash-”就有了这个字段。这个问题的根源是Elasticsearch的dynamic template在捣鬼，dynamic temlate用于指导Elasticsearch如何为插入的数据自动建立Schema映射关系，默认情况下，Logstash会在Elasticsearch中建立一个名为“logstash”的模板，所有前缀为“logstash-”的index都会参照这个模板来建立映射关系，在该模板中申明了要为每个字符串数据建立一个额外的带“.raw”后缀的字段。可以向Elasticsearch来查询你的模板，使用API：GET http://localhost:9200/\_template。

以上便是对ELK日志系统的总结介绍，还有一个重要的功能没有提到，就是如何将日志数据与自身产品业务的数据融合起来。举个例子，在nginx日志中，通常会包含API请求访问时携带的用户Token信息，由于Token是有时效性的，我们需要及时将这些Token转换成真实的用户信息存储下来。这样的需求通常有两种实现方式，一种是自己写一个Logstash filter，然后在Logstash处理数据时调用；另一种是将Logstash Indexer产生的数据再次输出到消息队列中，由我们自己的脚本程序从消息队列中取出数据，做相应的业务处理后，输出到Elasticsearch中。

ELK集群部署

（0）环境介绍

系统：centos7.5

防火墙：关闭

Selinux：关闭

时间：同步

服务器：N台

es-node1: #master

es-node2: #slave

注明：

master-slave模式：

master收集到日志后，会把一部分数据碎片到salve上（随机的一部分数据）；同时，master和slave又都会各自做副本，并把副本放到对方机器上，这样就保证了数据不会丢失。

如果master宕机了，那么客户端在日志采集配置中将elasticsearch主机指向改为slave，就可以保证ELK日志的正常采集和web展示。

logstash-node:

kibana-node:

app-node:

kafka-node1:

kafka-node2:

kafka-node3:

（1）elasticsearch安装配置

基础环境安装（es-node1与es-node2配置一致）：

1）获取软件包

**[**root**@**localhost **~]**#wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.4.2.rpm

2）安装

**[**root**@**es-node1 **~]**# yum **-**y install elasticsearch-6.4.2.rpm

**[**root**@**es-node2 **~]**# yum **-**y install elasticsearch-6.4.2.rpm

3）修改配置文件

es-node1:

**[**root**@**es-node1 **~]**# vim **/**etc**/**elasticsearch**/**elasticsearch.yml

cluster.name**:** myescluster

node.name**:** es-node1

path.data**:** **/**var**/**lib**/**elasticsearch

path.logs**:** **/**var**/**log**/**elasticsearch

network.host**:** 192**.**168**.**208**.**178

http.port**:** 9200

discovery.zen.ping.unicast.hosts**:** **[**"192.168.208.178"**,** "192.168.208.183"**]**

discovery.zen.minimum\_master\_nodes**:** 1

**[**root**@**es-node1 **~]**#systemctl enable elasticsearch

**[**root**@**es-node1 **~]**#systemctl start elasticsearch

es-node2:

**[**root**@**es-node2 **~]**# vim **/**etc**/**elasticsearch**/**elasticsearch.yml

cluster.name**:** myescluster

node.name**:** es-node2

path.data**:** **/**var**/**lib**/**elasticsearch

path.logs**:** **/**var**/**log**/**elasticsearch

network.host**:** 192**.**168**.**208**.**183

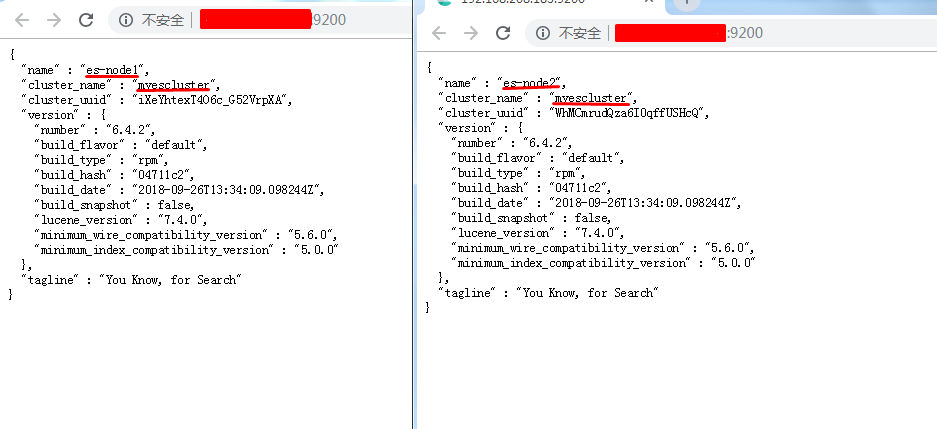
http.port**:** 9200

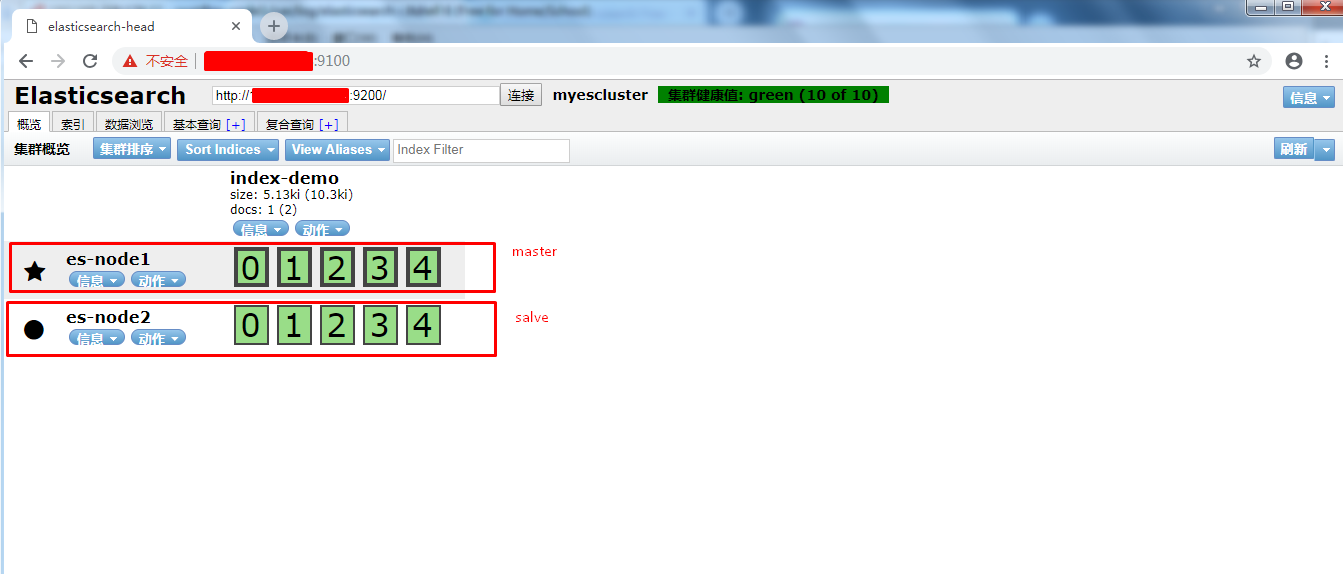
discovery.zen.ping.unicast.hosts**:** **[**"192.168.208.178"**,** "192.168.208.183"**]**

discovery.zen.minimum\_master\_nodes**:** 1

**[**root**@**es-node2 **~]**#systemctl enable elasticsearch

**[**root**@**es-node2 **~]**#systemctl start elasticsearch





（3）部署ElasticStash-head插件

elasticsearch Head是集群管理、数据可视化、增删改查、查询语句可视化工具,在最新的ES5中安装方式和ES2以上的版本有很大的不同,在ES2中可以直接在bin目录下执行plugin install xxxx 来进行安装,但是在ES5中这种安装方式变了,要想在ES5中安装Elasticsearch Head必须要安装NodeJs,然后通过NodeJS来启动Head。

具体安装步骤如下:

1)github下载源码:https://github.com/mobz/elasticsearch-head

**[**root**@**es-node1 **~]**# git clone https://github.com/mobz/elasticsearch-head

2)安装NodeJS

https://github.com/nodesource/distributions#rpm

**[**root**@**es-node1 **~]**# curl **-**sL https://rpm.nodesource.com/setup\_10.x **|** **bash** **-**

**[**root**@**es-node1 **~]**# yum **-**y install nodejs

**[**root**@**es-node1 **~]**# node **-**v

**[**root**@**es-node1 **~]**# npm **-**v

**[**root**@**es-node1 **~]**# npm install -g grunt **-**cli

3)安装elasticsearch-head

**[**root**@**es-node1 elasticsearch-head**]**# npm install

4)修改Elasticsearch.yml文件

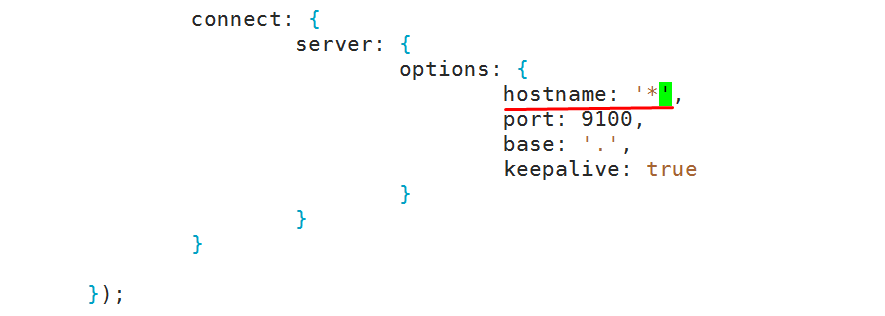
http.cors.enabled: true

http.cors.allow-origin: "\*"

在cluster上方添加以上两行内容

5）编辑~/elasticsearch-head/Gruntfile.js文件

**[**root**@**es-node1 elasticsearch-head**]**# vim Gruntfile.js



6）编辑~/elasticsearch-head/\_site/app.js文件

**[**root**@**es-node1 elasticsearch-head**]**# vim \_site**/**app.js

把文件中4354行中的的localhost修改为elasticsearch服务器地址

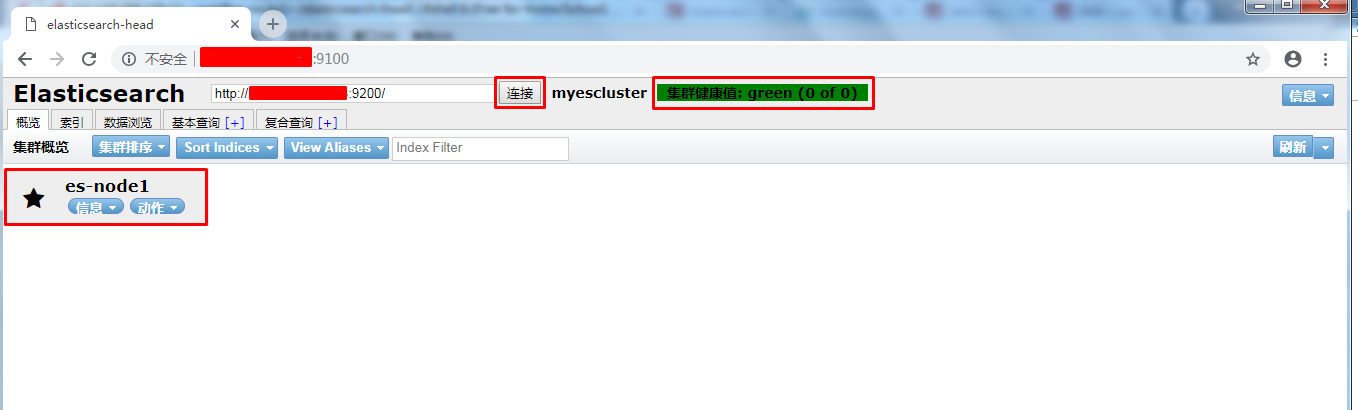
7）启动elasticsearch-head

在elasticsearch-head启动之前要把elasticsearch先进行重启。

在elasticsearch-head目录下启动服务

**[**root**@**es-node1 elasticsearch-head**]**# grunt server

8）访问elasticsearch-head Web前端页面



注意：

① 此时elasticsearch-head为前台启动，如果终端退出，那么elasticsearch-head服务也会随之关闭。

② 在非elasticsearch-head目录中启动server会失败！因为grunt需要读取目录下的Gruntfile.js。

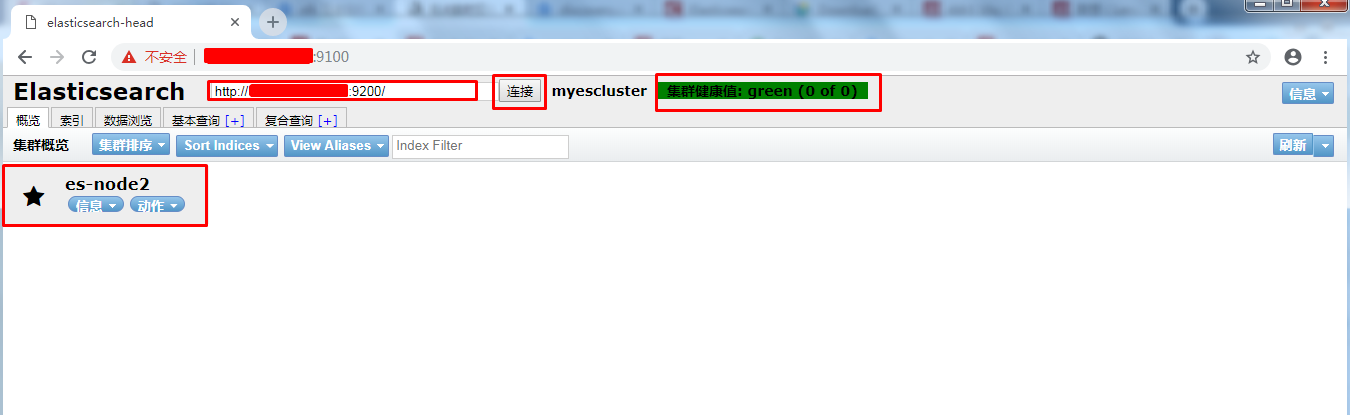
9)添加第二台elasticsearch到elasticsearch-head页面

**[**root**@**es-node2 **~]**# vim **/**etc**/**elasticsearch**/**elasticsearch.yml

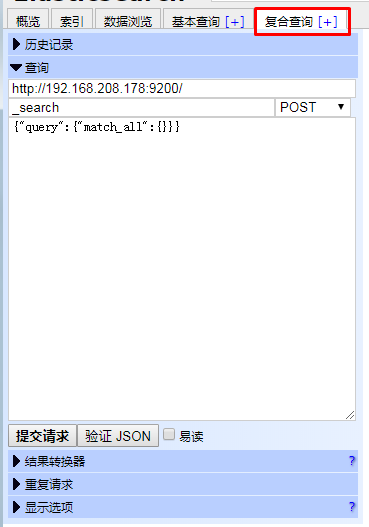
http.cors.enabled**:** true

http.cors.allow-origin**:** "\*"

**[**root**@**es-node2 **~]**# systemctl restart elasticsearch

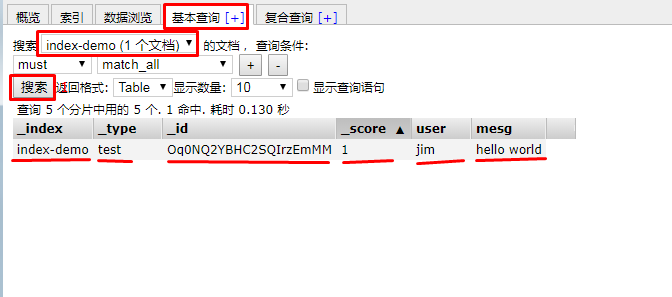


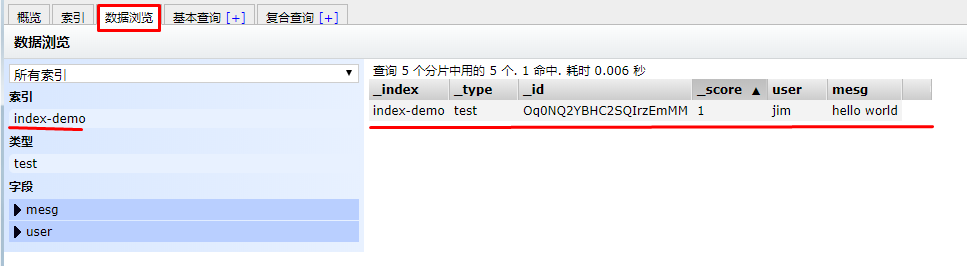
10)验证elasticsearch-head可用性











（4）Logstash部署

Logstash is an open source data collection engine with real-time pipelining capabilities。

简单说logstash就是一根具备实时数据传输能力的管道，负责将数据信息从管道的输入端传输到管道的输出端；与此同时这根管道还可以让你根据自己的需求在中间加上滤网，Logstash提供里很多功能强大的滤网以满足你的各种应用场景。

1）获取软件包

**[**root**@**localhost **~]**# wget https://artifacts.elastic.co/downloads/logstash/logstash-6.4.2.rpm

2）部署

**[**root**@**localhost **~]**# yum **-**y install logstash-6.4.2.rpm

3）启动

**[**root**@**localhost **~]**# systemctl enable logstash**;systemctl** start logstash

**[**root**@**localhost **~]**# ps aux **|** **grep** logstash

4）验证可用性

4.1）快速启动，标准输入输出作为input和output，没有filter

**[**root**@**localhost **~]**# cd **/**usr**/**share**/**logstash

**[**root**@**localhost logstash**]**# **./**bin**/**logstash -e 'input { stdin{} } output{ stdout {} }'

4.2）添加codec进行格式化输出

**[**root**@**localhost logstash**]**# **./**bin**/**logstash -e 'input{ stdin {} } output{ stdout {codec=>rubydebug} }'

5）Logstash参数与配置

5.1）常用启动参数



5.2）配置文件结构及语法

●区段

  Logstash通过{}来定义区域，区域内可以定义插件，一个区域内可以定义多个插件，如下：

input **{**

stdin **{**

**}**

beats **{**

port **=>** 5044

**}**

**}**

●数据类型

Logstash仅支持少量的数据类型：

​ Boolean：ssl\_enable **=>** true

​ Number：port **=>** 33

​ String：name **=>** “Hello world”

​ Commonts：# this is a comment

●字段引用

Logstash数据流中的数据被称之为Event对象，Event以JSON结构构成，Event的属性被称之为字段，如果你像在配置文件中引用这些字段，只需要把字段的名字写在中括号**[]**里就行了，如**[**type**]**，对于嵌套字段每层字段名称都写在**[]**里就可以了，比如：**[**tags**][**type**]**；除此之外，对于Logstash的arrag类型支持下标与倒序下表，如：**[**tags**][**type**][**0**],[**tags**][**type**][-**1**]**。

●条件判断

Logstash支持下面的操作符：

​ equality：**==,** **!=,** **<,** **>,** **<=,** **>=**

​ regexp：**=~,** **!~**

​ inclusion：in**,** not in

​ boolean：and**,** or**,** nand**,** xor

​ unary：**!**

例如：

**if** EXPRESSION **{**

**...**

**}** **else** **if** EXPRESSION **{**

**...**

**}** **else** **{**

**...**

**}**

环境变量引用

●Logstash支持引用系统环境变量，环境变量不存在时可以设置默认值，例如：

**[**root**@**localhost **~]**# export TCP\_PORT**=**12345

input **{**

tcp **{**

port **=>** "${TCP\_PORT:54321}"

**}**

**}**

5.3）常用输入插件（Input plugin）

5.3.1）File读取插件

文件读取插件主要用来抓取文件的变化信息，将变化信息封装成Event进程处理或者传递。

配置实例：

input

**file** **{**

path **=>** **[**"/var/log/\*.log"**,** "/var/log/message"**]**

**type** **=>** "system"

start\_position **=>** "beginning"

**}**

**}**

常用参数：





5.3.2）Beats监听插件

Beats插件用于建立监听服务，接收Filebeat或者其他beat发送的Events；

配置实例：

input **{**

beats **{**

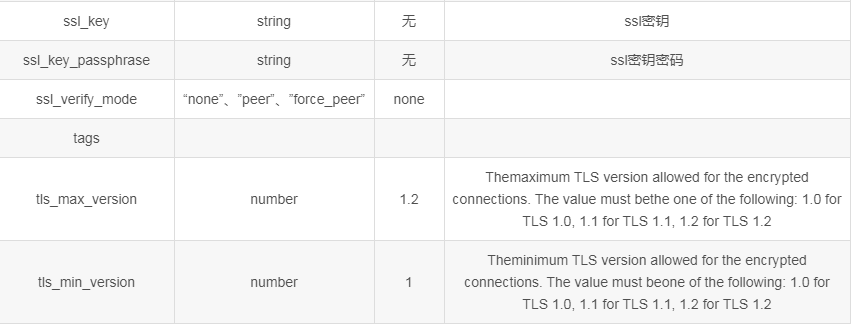
port **=>** 5044

**}**

**}**

常用参数：





5.3.3）TCP监听插件

TCP插件有两种工作模式，“Client”和“Server”，分别用于发送网络数据和监听网络数据。

配置实例：

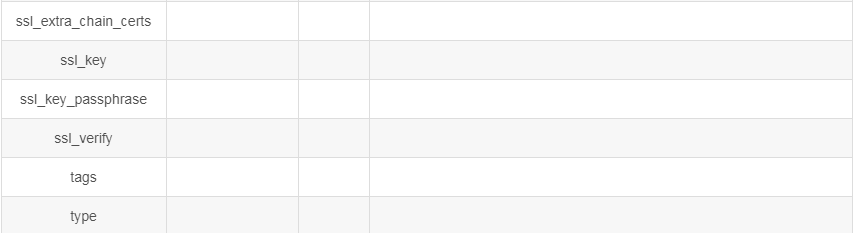
tcp **{**

port **=>** 41414

**}**

常用参数：





5.3.4）Redis读取插件

用于读取Redis中缓存的数据信息。

配置实例：

input **{**

redis **{**

host **=>** "127.0.0.1"

port **=>** 6379

data\_type **=>** "list"

key **=>** "logstash-list"

**}**

**}**

常用参数：





5.3.5）Syslog监听插件

  监听操作系统syslog信息

配置实例：

syslog {

}

常用参数：





5.4）常用过滤插件（Filter plugin）

  丰富的过滤器插件的是 logstash威力如此强大的重要因素，过滤器插件主要处理流经当前Logstash的事件信息，可以添加字段、移除字段、转换字段类型，通过正则表达式切分数据等，也可以根据条件判断来进行不同的数据处理方式。

5.4.1）grok正则捕获

  grok 是Logstash中将非结构化数据解析成结构化数据以便于查询的最好工具，非常适合解析syslog logs，apache log， mysql log，以及一些其他的web log

●预定义表达式调用

Logstash提供120个常用正则表达式可供安装使用，安装之后你可以通过名称调用它们，语法如下：**%{**SYNTAX**:**SEMANTIC**}**

SYNTAX：表示已经安装的正则表达式的名称

SEMANTIC：表示从Event中匹配到的内容的名称

例如：

Event的内容为“**[**debug**]** 127**.**0**.**0**.**1 **-** test log content”，匹配**%{**IP**:**client**}**将获得“client**:** 127**.**0**.**0**.**1”的结果，前提安装了IP表达式；如果你在捕获数据时想进行数据类型转换可以使用**%{**NUMBER**:**num**:**int**}**这种语法，默认情况下，所有的返回结果都是string类型，当前Logstash所支持的转换类型仅有“int”和“float”；

一个稍微完整一点的事例：

​ 日志文件http.log内容：55**.**3**.**244**.**1 GET **/**index.html 15824 0**.**043

​ 表达式：**%{**IP**:**client**}** **%{**WORD**:**method**}** **%{**URIPATHPARAM**:**request**}** **%{**NUMBER**:**bytes**}** **%{**NUMBER**:**duration**}**

配置文件内容：

input **{**

**file** **{**

path **=>** "/var/log/http.log"

**}**

**}**

filter **{**

grok **{**

match **=>** **{**"message" **=>** "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration}"**}**

**}**

**}**

输出结果：

client**:** 55**.**3**.**244**.**1

method**:** GET

request**:** **/**index.html

bytes**:** 15824

duration**:** 0**.**043

●自定义表达式调用

语法：**(?<**field\_name**>**the pattern here**)**

举例：捕获10或11和长度的十六进制queue\_id可以使用表达式**(?<**queue\_id**>[**0**-**9A-F**]{**10**,**11**})**

●安装自定义表达式

与预定义表达式相同，你也可以将自定义的表达式配置到Logstash中，然后就可以像于定义的表达式一样使用；以下是操作步骤说明：

1、在Logstash根目录下创建文件夹“patterns”，在“patterns”文件夹中创建文件“extra”（文件名称无所谓，可自己选择有意义的文件名称）；

2、在文件“extra”中添加表达式，格式：patternName regexp，名称与表达式之间用空格隔开即可，如下：

# contents of ./patterns/postfix:

POSTFIX\_QUEUEID **[**0**-**9A-F**]{**10**,**11**}**

3、使用自定义的表达式时需要指定“patterns\_dir”变量，变量内容指向表达式文件所在的目录，举例如下：

## 日志内容 ##

Jan 1 06**:**25**:**43 mailserver14 postfix**/**cleanup**[**21403**]:** BEF25A72965**:** message-id**=<**20130101142543**.**5828399CCAF**@**mailserver14.example.com**>**

## Logstash配置 ##

filter **{**

grok **{**

patterns\_dir **=>** **[**"./patterns"**]**

match **=>** **{** "message" **=>** "%{SYSLOGBASE} %{POSTFIX\_QUEUEID:queue\_id}: %{GREEDYDATA:syslog\_message}" **}**

**}**

**}**

## 运行结果 ##

timestamp**:** Jan 1 06**:**25**:**43

logsource**:** mailserver14

program**:** postfix**/**cleanup

pid**:** 21403

queue\_id**:** BEF25A72965

常用参数：





**-** 其他

**-** 一般的正则表达式只能匹配单行文本，如果一个Event的内容为多行，可以在pattern前加“**(?**m**)**”

**-** 对于Hash和Array类型，Hash表示键值对，Array表示数组

**-** Grok表达式在线debug地址：http://grokdebug.herokuapp.com

**-** 预定义正则表达式参考地址：https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns

5.4.2）date时间处理插件

该插件用于时间字段的格式转换，比如将“Apr 17 09:32:01”（MMM dd HH:mm:ss）转换为“MM-dd HH:mm:ss”。而且通常情况下，Logstash会为自动给Event打上时间戳，但是这个时间戳是Event的处理时间（主要是input接收数据的时间），和日志记录时间会存在偏差（主要原因是buffer），我们可以使用此插件用日志发生时间替换掉默认是时间戳的值。

常用参数：



5.4.3）mutate 数据修改插件

mutate 插件是 Logstash另一个重要插件。它提供了丰富的基础类型数据处理能力。可以重命名，删除，替换和修改事件中的字段。

常用参数：





5.4.4）JSON插件

JSON插件用于解码JSON格式的字符串，一般是一堆日志信息中，部分是JSON格式，部分不是的情况下

配置实例：

json **{**

source **=>** **...**

**}**

## 事例配置，message是JSON格式的字符串："{\"uid\":3081609001,\"type\":\"signal\"}" ##

filter **{**

json **{**

source **=>** "message"

target **=>** "jsoncontent"

**}**

**}**

## 输出结果 ##

**{**

"@version"**:** "1"**,**

"@timestamp"**:** "2014-11-18T08:11:33.000Z"**,**

"host"**:** "web121.mweibo.tc.sinanode.com"**,**

"message"**:** "{\"uid\":3081609001,\"type\":\"signal\"}"**,**

"jsoncontent"**:** **{**

"uid"**:** 3081609001**,**

"type"**:** "signal"

**}**

**}**

## 如果从事例配置中删除`target`，输出结果如下 ##

**{**

"@version"**:** "1"**,**

"@timestamp"**:** "2014-11-18T08:11:33.000Z"**,**

"host"**:** "web121.mweibo.tc.sinanode.com"**,**

"message"**:** "{\"uid\":3081609001,\"type\":\"signal\"}"**,**

"uid"**:** 3081609001**,**

"type"**:** "signal"

**}**

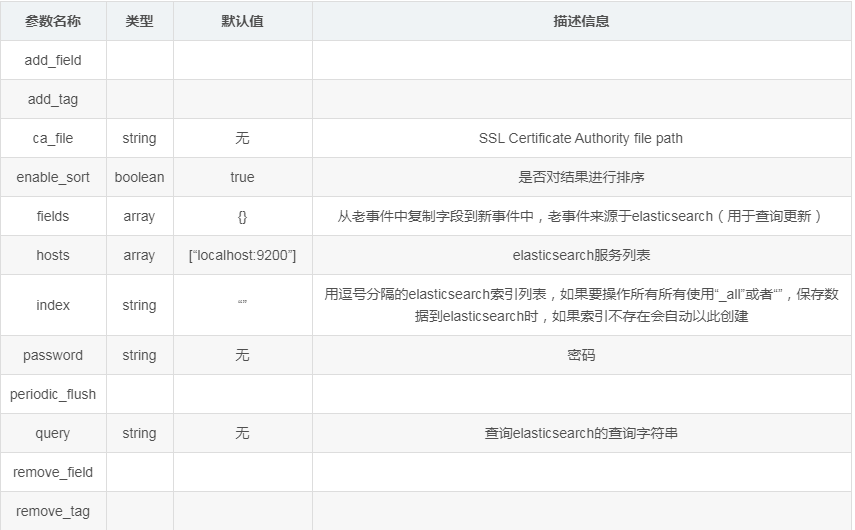
常用参数：



5.4.5）elasticsearch查询过滤插件

  用于查询Elasticsearch中的事件，可将查询结果应用于当前事件中

常用参数：





5.4.6）其他

  还有很多其他有用插件，如：Split、GeoIP、Ruby等等

5.5）常用输出插件（Output plugin）

5.5.1）ElasticSearch输出插件

  用于将事件信息写入到Elasticsearch中，官方推荐插件，ELK必备插件

配置实例：

output **{**

elasticsearch **{**

hosts **=>** **[**"127.0.0.1:9200"**]**

index **=>** "filebeat-%{type}-%{+yyyy.MM.dd}"

template\_overwrite **=>** true

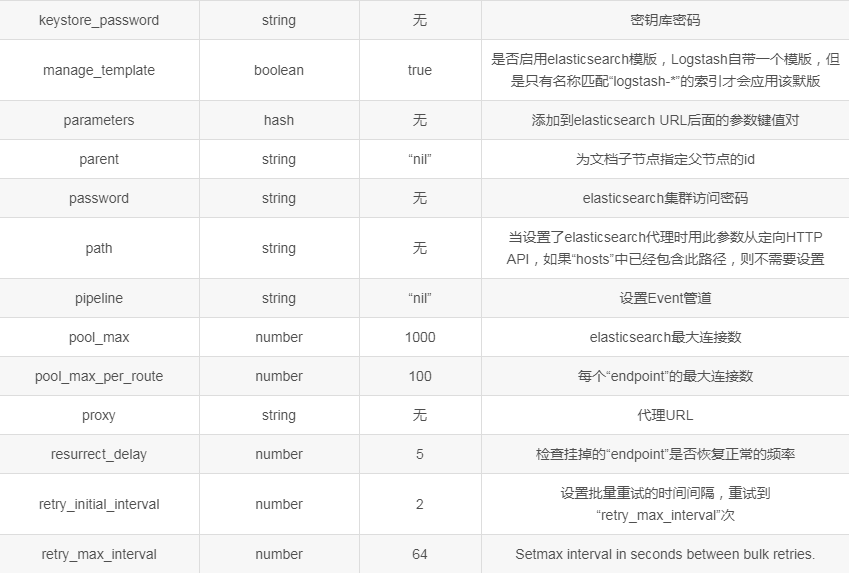
**}**

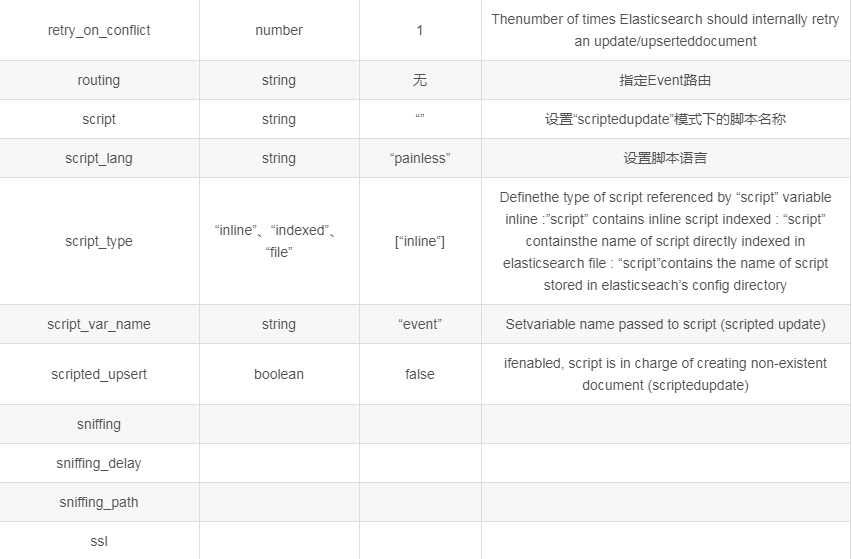
**}**

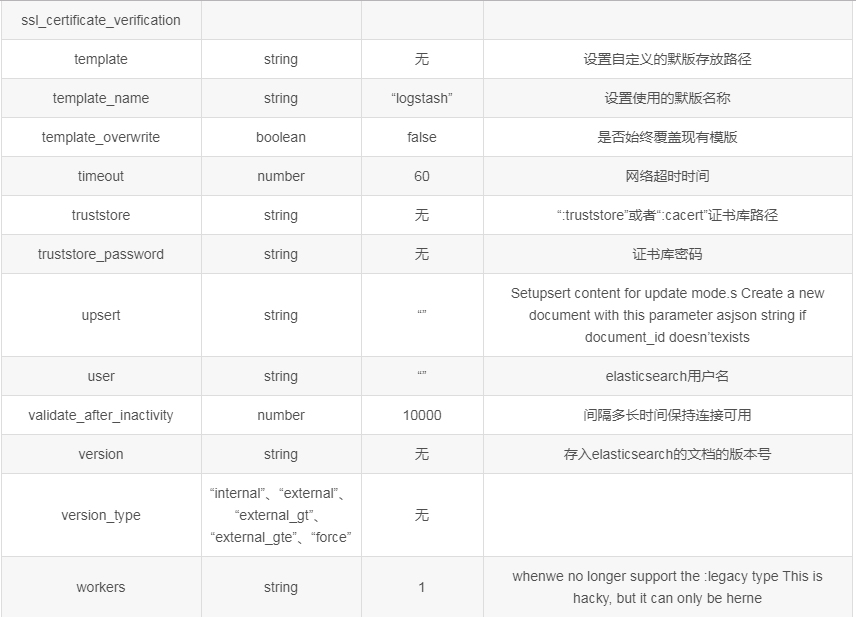
常用参数：











5.5.2）Redis输出插件

  用于将Event写入Redis中进行缓存，通常情况下Logstash的Filter处理比较吃系统资源，复杂的Filter处理会非常耗时，如果Event产生速度比较快，可以使用Redis作为buffer使用

配置实例：

output **{**

redis **{**

host **=>** "127.0.0.1"

port **=>** 6379

data\_type **=>** "list"

key **=>** "logstash-list"

**}**

**}**

常用配置：





5.5.3）File输出插件

  用于将Event输出到文件内

配置实例：

output **{**

**file** **{**

path **=>** **...**

codec **=>** line **{** format **=>** "custom format: %{message}"**}**

**}**

**}**

常用参数：



5.5.4）TCP插件

Write events over a TCP socket.Each event json is separated by a newline.Can either accept connections from clients or connect to a server, depending on mode.

配置实例：

tcp {

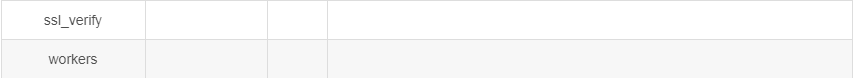
host => ...

port => ...

}

常用参数：





5.6）常用编码插件（Codec plugin）

JSON编码插件

直接输入预定义好的 JSON 数据，这样就可以省略掉 filter/grok 配置

配置实例：

json {

}

常用参数：



6)Logstash实例

6.1）接收Filebeat事件，输出到Redis

input **{**

beats **{**

port **=>** 5044

**}**

**}**

output **{**

redis **{**

host **=>** "127.0.0.1"

port **=>** 6379

data\_type **=>** "list"

key **=>** "logstash-list"

**}**

**}**

6.2）读取Redis数据，根据“type”判断，分别处理，输出到ES

input **{**

redis **{**

host **=>** "127.0.0.1"

port **=>** 6379

data\_type **=>** "list"

key **=>** "logstash-list"

**}**

**}**

filter **{**

**if** **[**type**]** **==** "application" **{**

grok **{**

match **=>** **[**"message"**,** "(?m)-(?<systemName>.+?):(?<logTime>(?>\d\d){1,2}-(?:0?[1-9]|1[0-2])-(?:(?:0[1-9])|(?:[12][0-9])|(?:3[01])|[1-9]) (?:2[0123]|[01]?[0-9]):(?:[0-5][0-9]):(?:(?:[0-5][0-9]|60)(?:[:.,][0-9]+)?)) \[(?<level>(\b\w+\b)) \*\] (?<thread>(\b\w+\b)) \((?<point>.\*?)\) - (?<content>.\*)"**]**

**}**

**date** **{**

match **=>** **[**"logTime"**,** "yyyy-MM-dd HH:mm:ss,SSS"**]**

**}**

json **{**

source **=>** "message"

**}**

**date** **{**

match **=>** **[**"timestamp"**,** "yyyy-MM-dd HH:mm:ss,SSS"**]**

**}**

**}**

**if** **[**type**]** **==** "application\_bizz" **{**

json **{**

source **=>** "message"

**}**

**date** **{**

match **=>** **[**"timestamp"**,** "yyyy-MM-dd HH:mm:ss,SSS"**]**

**}**

**}**

mutate **{**

remove\_field **=>** **[**"@version"**,** "beat"**,** "logTime"**]**

**}**

**}**

output **{**

stdout**{**

**}**

elasticsearch **{**

hosts **=>** **[**"127.0.0.1:9200"**]**

index **=>** "filebeat-%{type}-%{+yyyy.MM.dd}"

document\_type **=>** "%{documentType}"

template\_overwrite **=>** true

**}**

**}**

6）kafka集群部署

6.1）zookeeper集群

ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，它是集群的管理者，监视着集群中各个节点的状态根据节点提交的反馈进行下一步合理操作。最终，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

6.1.1）环境

kafka-node1:

kafka-node2:

6.1.2）zookeeper包获取

**[**root**@**localhost **~]**# wget http://mirrors.shu.edu.cn/apache/zookeeper/zookeeper-3.4.13/zookeeper-3.4.13.tar.gz

6.1.3）zookeeper部署

**[**root**@**localhost **~]**# tar xf zookeeper-3.4.13.tar.gz

**[**root**@**localhost **~]**# mv zookeeper-3.4.13 **/**usr**/**local**/**zookeeper

6.1.4）配置文件

**[**root**@**localhost **~]**# cd **/**usr**/**local**/**zookeeper**/**

**[**root**@**localhost zookeeper**]**# mkdir data

**[**root**@**localhost zookeeper**]**# cd conf

**[**root**@**localhost conf**]**# cp zoo\_sample.cfg zoo.cfg

**[**root**@**localhost conf**]**# vim zoo.cfg

dataDir**=/**usr**/**local**/**zooker**/**data

server.1**=**192**.**168**.**208**.**185**:**2888**:**3888

server.2**=**192**.**168**.**208**.**170**:**2888**:**3888

**[**root**@**localhost conf**]**# cd **/**usr**/**local**/**zookeeper**/**data

**[**root**@**localhost data**]**# echo "1" **>>** myid

6.1.5）启动

**[**root**@**localhost **~]**# **/**usr**/**local**/**zookeeper**/**bin**/**zkServer.sh start

6.1.6）验证

**[**root**@**localhost **~]**# **/**usr**/**local**/**zookeeper**/**bin**/**zkServer.sh status

注：

验证完成后，请把zookeeper部署目录远程复制到集群其它主机，修改data/myid后启动即可。

6.2）kafka集群

名词解释：

Broker：Kafka 集群包含一个或多个服务器，这种服务器被称为 broker。

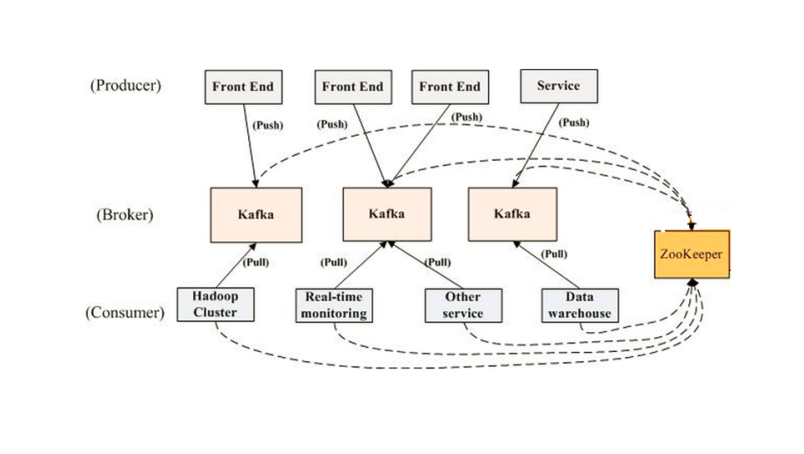
Topic：每条发布到 Kafka 集群的消息都有一个类别，这个类别被称为 Topic。（物理上不同 Topic 的消息分开存储，逻辑上一个 Topic 的消息虽然保存于一个或多个 broker 上，但用户只需指定消息的 Topic 即可生产或消费数据而不必关心数据存于何处）。

Partition：Partition 是物理上的概念，每个 Topic 包含一个或多个 Partition。

Producer：负责发布消息到 Kafka broker。

Consumer：消息消费者，向 Kafka broker 读取消息的客户端。

Consumer Group：每个 Consumer 属于一个特定的 Consumer Group（可为每个 Consumer 指定 group name，若不指定 group name 则属于默认的 group）。



如上图所示，一个典型的kafka集群中包含若干producer（可以是web前端产生的page view，或者是服务器日志，系统CPU、memory等），若干broker（Kafka支持水平扩展，一般broker数量越多，集群吞吐率越高），若干consumer group，以及一个Zookeeper集群。Kafka通过Zookeeper管理集群配置，选举leader，以及在consumer group发生变化时进行rebalance。producer使用push模式将消息发布到broker，consumer使用pull模式从broker订阅并消费消息。

6.2.1）环境

kafka-node1:

kafka-node2:

kafka版本：kafka\_2.10-0.10.1.0.tgz

6.2.2）kafka包获取

**[**root**@**localhost **~]**# wget http://archive.apache.org/dist/kafka/2.0.0/kafka\_2.10-0.10.1.0.tgz

6.2.3）部署kafka集群

**[**root**@**kafka-node1 **~]**# tar xf kafka\_2.10-0.10.1.0.tgz

**[**root**@**kafka-node1 **~]**# mv kafka\_2.10-0.10.1.0 **/**usr**/**local**/**kafka

**[**root**@**kafka-node1 **~]**# cd **/**usr**/**local**/**kafka**/**config

**[**root**@**kafka-node1 config**]**# vim server.properties

broker.id**=**1 #当前机器在集群中的唯一标识，和zookeeper的myid性质一样

listeners**=**PLAINTEXT://:9092 #当前kafka对外提供服务的端口默认是9092

num.network.threads**=**3 # 这个是borker进行网络处理的线程数

num.io.threads**=**8 # 这个是borker进行I/O处理的线程数

socket.send.buffer.bytes**=**102400 #发送缓冲区buffer大小，数据不是一下子就发送的，先回存储到缓冲区了到达一定的大小后在发送，能提高性能

socket.receive.buffer.bytes**=**102400 #kafka接收缓冲区大小，当数据到达一定大小后在序列化到磁盘

socket.request.max.bytes**=**104857600 #这个参数是向kafka请求消息或者向kafka发送消息的请请求的最大数，这个值不能超过java的堆栈大小

log.dirs**=/**var**/**log**/**kafka-logs

#消息存放的目录，这个目录可以配置为“，”逗号分割的表达式，上面的num.io.threads要大于这个目录的个数这个目录，如果配置多个目录，新创建的topic他把消息持久化的地方是，当前以逗号分割的目录中，那个分区数最少就放那一个

num.partitions**=**1 # 默认的分区数，一个topic默认1个分区数

num.recovery.threads.per.data.dir**=**1

offsets.topic.replication.factor**=**1

transaction.state.log.replication.factor**=**1

transaction.state.log.min.isr**=**1

log.retention.hours**=**168 # 默认消息的最大持久化时间，168小时，7天

log.segment.bytes**=**1073741824

log.retention.check.interval.ms**=**300000 #每隔300000毫秒去检查上面配置的log失效时间（默认168 ），到目录查看是否有过期的消息如果有，删除

message.max.byte**=**5242880 #消息保存的最大值5M

default.replication.factor**=**2 #kafka保存消息的副本数，如果一个副本失效了，另一个还可以继续提供服务

replica.fetch.max.bytes**=**5242880 #取消息的最大直接数

zookeeper.connect**=**192**.**168**.**208**.**185**:**2181**,**192**.**168**.**208**.**170**:**2181 #设置zookeeper的连接端口

zookeeper.connection.timeout.ms**=**6000

group.initial.rebalance.delay.ms**=**0

**[**root**@**kafka-node2 **~]**# vim **/**usr**/**local**/**kafka**/**config**/**server.properties

**[**root**@**kafka-node2 **~]** egrep **-**v '^#|^$' **/**usr**/**local**/**kafka**/**config**/**server.properties

broker.id**=**2

listeners**=**PLAINTEXT://:9092

num.network.threads**=**3

num.io.threads**=**8

socket.send.buffer.bytes**=**102400

socket.receive.buffer.bytes**=**102400

socket.request.max.bytes**=**104857600

log.dirs**=/**var**/**log**/**kafka-logs

num.partitions**=**1

num.recovery.threads.per.data.dir**=**1

offsets.topic.replication.factor**=**1

transaction.state.log.replication.factor**=**1

transaction.state.log.min.isr**=**1

log.retention.hours**=**168

log.segment.bytes**=**1073741824

log.retention.check.interval.ms**=**300000

message.max.byte**=**5242880

default.replication.factor**=**2

replica.fetch.max.bytes**=**5242880

zookeeper.connect**=**node71**:**2181**,**node72**:**2181**,**node73**:**2181

zookeeper.connection.timeout.ms**=**6000

group.initial.rebalance.delay.ms**=**0

6.2.4）启动kafka集群

在2台机器上都执行启动Kafka集群：

**[**root**@**kafka-nodex **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-server-start.sh **-**daemon **/**usr**/**local**/**kafka**/**config**/**server.properties

**[**root**@**kafka-nodex **~]**# ps aux **|** **grep** kafka #验证是否启动

**[**root**@**kafka-nodex **~]**# netstat **-**anput **|** **grep** ":9092"

6.2.5)测试kafka集群可用性

6.2.5.1）创建Topic

**[**root**@**kafka-node1 **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-topics.sh **--**create **--**zookeeper 192**.**168**.**208**.**185**:**2181 **--**replication-factor 2 **--**partitions 3 **--**topic mysqlone

OpenJDK 64**-**Bit Server VM warning**:** If the number of processors is expected to increase from one**,** then you should configure the number of parallel GC threads appropriately using **-**XX**:**ParallelGCThreads**=**N

Created topic "mysqlone"**.**

注：

**--**create 创建话题

**--**zookeeper 192**.**168**.**208**.**185**:**2181 指定zookeeper服务器

**--**replication-factor 2 复制2份

**--**partitions 3 3个分区，即3个文件夹

**--**topic mysqlone 话题名称为mysqlone

**[**root**@**kafka-node1 **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-topics.sh **--**create **--**zookeeper 192**.**168**.**208**.**185**:**2181 **--**replication-factor 2 **--**partitions 2 **--**topic mysqltwo

OpenJDK 64**-**Bit Server VM warning**:** If the number of processors is expected to increase from one**,** then you should configure the number of parallel GC threads appropriately using **-**XX**:**ParallelGCThreads**=**N

Created topic "mysqltwo"**.**

6.2.5.2）在一台服务器上创建一个发布者broker

**[**root**@**kafka-node2 **~]**#**/**usr**/**local**/**kafka**/**bin**/**kafka-console-producer.sh **--**broker-list 192**.**168**.**208**.**185**:**9092 **--**topic mysqlone

OpenJDK 64**-**Bit Server VM warning**:** If the number of processors is expected to increase from one**,** then you should configure the number of parallel GC threads appropriately using **-**XX**:**ParallelGCThreads**=**N

**>**abc test

注：

**--**broker-list 192**.**168**.**208**.**185**:**9092 产生broker服务器IP，集群中任意服务器IP地址均可

6.2.5.3）在一台服务器上创建一个订阅者consumer

**[**root**@**kafka-node2 **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-console-consumer.sh **--**bootstrap-server 192**.**168**.**208**.**185**:**9092 **--**from-beginning **--**topic mysqlone

OpenJDK 64**-**Bit Server VM warning**:** If the number of processors is expected to increase from one**,** then you should configure the number of parallel GC threads appropriately using **-**XX**:**ParallelGCThreads**=**N

abc test

注：

对于消费者，kafka中有两个设置的地方：对于老的消费者，由**--**zookeeper参数设置；对于新的消费者，由**--**bootstrap-server参数设置

如果使用了**--**zookeeper参数**,**那么consumer的信息将会存放在zk之中

查看的方法：

**./**zookeeper-client**,**然后 ls **/**consumers**/[**group\_id**]/**offsets**/[**topic**]/[**broker\_id-part\_id**],**这个是查看某个group\_id的某个topic的offset

如果使用了**--**bootstrap-server参数**,**那么consumer的信息将会存放在kafka之中

6.2.5.4）列出全部的topic

**[**root**@**kafka-node1 **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-topics.sh **--**list **--**zookeeper 192**.**168**.**208**.**xxx**:**2181

注：

**--**zookeeper 192**.**168**.**208**.**xxx**:**2181 为zookeeper集群中任意服务器IP地址

6.2.5.5）查看指定的topic状态

**[**root**@**kafka-node1 **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-topics.sh **--**describe **--**zookeeper 192**.**168**.**208**.**185**:**2181 **--**topic mysqlone

Topic**:**mysqlone PartitionCount**:**3 ReplicationFactor**:**2 Configs**:**

Topic**:** mysqlone Partition**:** 0 Leader**:** 1 Replicas**:** 1**,**2 Isr**:** 1**,**2

Topic**:** mysqlone Partition**:** 1 Leader**:** 2 Replicas**:** 2**,**1 Isr**:** 2**,**1

Topic**:** mysqlone Partition**:** 2 Leader**:** 1 Replicas**:** 1**,**2 Isr**:** 1**,**2

注：

Topic: 主题名称

Partition: 分片编号

Leader: 该分区的leader节点

Replicas: 该副本存在于哪个broker节点

Isr: 活跃状态的broker

6.2.5.6)使用zkCli.sh查看kafka brokers状态

**[**root**@**kafka-node1 **~]**# **/**usr**/**local**/**zookeeper**/**bin**/**zkCli.sh **-**server 192**.**168**.**208**.**185**:**2181

**[**zk**:** 192**.**168**.**208**.**185**:**2181**(**CONNECTED**)** 1**]** ls **/**brokers

**[**zk**:** 192**.**168**.**208**.**185**:**2181**(**CONNECTED**)** 2**]** get **/**brokers**/**ids

**[**zk**:** 192**.**168**.**208**.**185**:**2181**(**CONNECTED**)** 3**]** get **/**brokers**/**topics**/**mysqlone**/**partitions**/**0

**[**zk**:** 192**.**168**.**208**.**185**:**2181**(**CONNECTED**)** 4**]** get **/**brokers**/**topics**/**mysqlone**/**partitions**/**1

**[**zk**:** 192**.**168**.**208**.**185**:**2181**(**CONNECTED**)** 5**]** get **/**brokers**/**topics**/**mysqlone**/**partitions**/**2

注：

1、kafka工作原理、选举方法

2、删除topic方法

3、关闭kafka方法

7）filebeat部署

7.0）环境

**[**root**@**kafka-node1 **~]**# vim **/**etc**/**hosts

192**.**168**.**208**.**185 kafka-node1

192**.**168**.**208**.**170 kafka-node2

192**.**168**.**208**.**162 filebeat-node

**[**root**@**kafka-node2 **~]**# vim **/**etc**/**hosts

192**.**168**.**208**.**185 kafka-node1

192**.**168**.**208**.**170 kafka-node2

192**.**168**.**208**.**162 filebeat-node

**[**root**@**filebeat-node **~]**# vim **/**etc**/**hosts

192**.**168**.**208**.**185 kafka-node1

192**.**168**.**208**.**170 kafka-node2

192**.**168**.**208**.**162 filebeat-node

7.1）获取软件包

**[**root**@**filebeat-node **~]**# wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-6.4.2-x86\_64.rpm

7.2)部署

**[**root**@**filebeat-node **~]**# yum **-**y install filebeat-6.4.2-x86\_64.rpm

7.3）配置

7.3.1）创建topic

**[**root**@**kafka-node1 **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-topics.sh **--**create **--**zookeeper 192**.**168**.**208**.**185**:**2181 192**.**168**.**208**.**170**:**2181 **--**replication-factor 2 **--**partitions 2 **--**topic filebeatone

OpenJDK 64**-**Bit Server VM warning**:** If the number of processors is expected to increase from one**,** then you should configure the number of parallel GC threads appropriately using **-**XX**:**ParallelGCThreads**=**N

Created topic "filebeatone"**.**

7.3.2)修改filebeat.yml文件，使用kafka

**[**root**@**filebeat-node **~]**# vim **/**etc**/**filebeat**/**filebeat.yml

#=========================== Filebeat inputs =========================

filebeat.inputs**:**

**-** type**:** log

enabled**:** true

paths**:**

**-** **/**var**/**log**/**nginx**/\*.**log

#-------------------------- kafka output -----------------------------

output.kafka**:**

# Array of hosts to connect to.

hosts**:** **[**"kafka-node1:9092"**]**

topic**:** filebeatone

required\_acks**:** 1

#-------------------------- kafka finish -----------------------------

注：

hosts:位置中主机标识仅为主机名称，不能为主机IP地址。多主机的写法：

["kafka-node1:9092,kafka-node2:9092"]

7.4）启动

**[**root**@**filebeat-node **~]**# systemctl enable filebeat

**[**root**@**filebeat-node **~]**# systemctl start filebeat

7.5）验证

7.5.1）在kafka集群中任意一台服务器执行如下命令，等待消息生成后读出

**[**root**@**kafka-node1 **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-console-consumer.sh **--**bootstrap-server 192**.**168**.**208**.**185**:**9092 **--**from-beginning **--**topic filebeatone

7.5.2）任意客户端访问filebeat-node上web服务

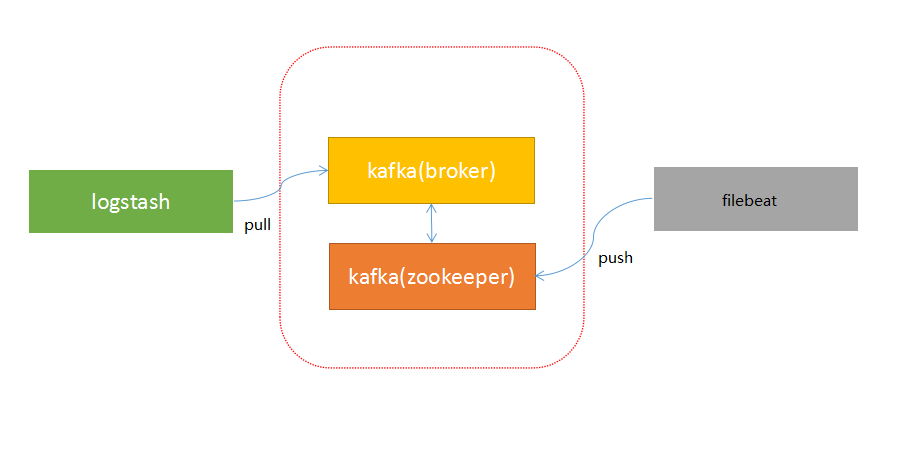
http://filebeat-node

7.5.3)得到7.5.1小节输出结果

**[**root**@**kafka-node1 **~]**# **/**usr**/**local**/**kafka**/**bin**/**kafka-console-consumer.sh **--**bootstrap-server 192**.**168**.**208**.**170**:**9092 **--**from-beginning **--**topic filebeatone

**{**"@timestamp"**:**"2018-10-06T01:30:28.975Z"**,**"@metadata"**:{**"beat"**:**"filebeat"**,**"type"**:**"doc"**,**"version"**:**"6.4.2"**,**"partition"**:**0**,**"topic"**:**"filebeatone"**},**"beat"**:{**"name"**:**"filebeat-node"**,**"hostname"**:**"filebeat-node"**,**"version"**:**"6.4.2"**},**"host"**:{**"name"**:**"filebeat-node"**},**"source"**:**"/var/log/nginx/access.log"**,**"offset"**:**0**,**"message"**:**"192.168.208.1 - - [06/Oct/2018:09:30:24 +0800] \"GET / HTTP/1.1\" 200 15 \"-\" \"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3551.3 Safari/537.36\" \"-\""**,**"prospector"**:{**"type"**:**"log"**},**"input"**:{**"type"**:**"log"**}}**

8）Logstash应用kafka集群实现日志收集



**[**root**@**localhost **~]**# vim **/**etc**/**logstash**/**conf.d**/**kafka.conf

input **{**

kafka **{**

bootstrap\_servers **=>** "192.168.208.185:9092,192.168.208.170:9092"

topics **=>** **[**"filebeatone"**]**

**}**

**}**

output **{**

elasticsearch **{**

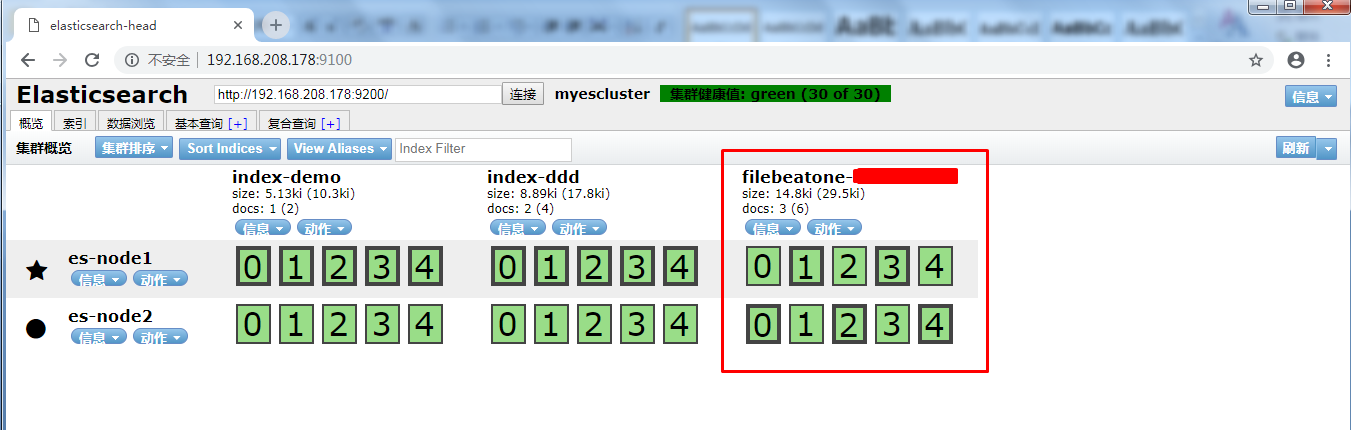
hosts **=>** **[**"192.168.208.178:9200"**,**"192.168.208.183:9200"**]**

index **=>** "filebeatone-%{+YYYY.MM.dd}"

manage\_template **=>** false

**}**

**}**



9）Kibana

9.0）环境

9.1)软件获取

**[**root**@**kibana-node **~]**# wget https://artifacts.elastic.co/downloads/kibana/kibana-6.4.2-x86\_64.rpm

9.2）部署

**[**root**@**kibana-node **~]**# yum **-**y install kibana-6.4.2-x86\_64.rpm

9.3）修改配置文件

**[**root**@**kibana-node **~]**# vim **/**etc**/**kibana**/**kibana.yml

server.port**:** 5601

server.host**:** "本地IP地址"

server.name**:** "kibana-node"

elasticsearch.url**:** "http://ES-IP:9200" #如果是多ES架构，请填写master主机IP

9.4）汉化

下载汉化包

**[**root**@**kibana-node **~]**#wget https://github.com/anbai-inc/Kibana\_Hanization/archive/master.zip

**[**root**@**kibana-node **~]**#unzip master.zip

**[**root**@**kibana-node **~]**#cd Kibana\_Hanization-master**/**

执行汉化

**[**root**@**kibana-node **~]**#python main.py **/**usr**/**share**/**kibana （查询kibana安装目录：rpm **-**ql kibana）

9.5）启动

**[**root**@**es-node2 **~]**# systemctl enable kibana

**[**root**@**es-node2 **~]**# systemctl start kibana

9.6）访问

http://kibana-node：5601

9.7）系统管理

9.7.1）添加索引







