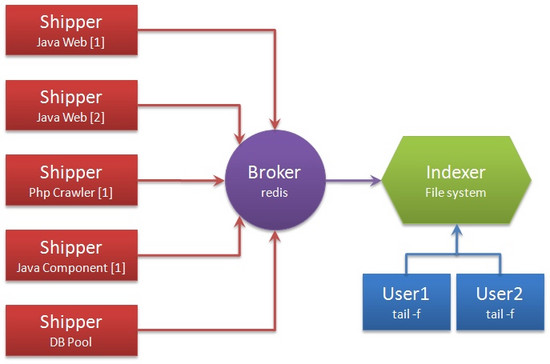
Logstash 是由 JRuby 编写的，使用基于消息的简单架构，在 JVM 上运行。理念非常简单，如果说 MapReduce 框架分为 Mapper 和 Reducer 两大模块，那么 Logstash 有仨：

1、Collect: 数据输入。对应 input

2、Enrich: 数据处理。对应 filter

3、Transport: 数据输出。对应 output

不过这里我们主要还是以日志汇总处理系统的思路来进行介绍，一个典型的架构为：



图中的三个组件为 Shipper, Broker 和 Indexer，这里我们结合上图分别来进行介绍

**Shipper** 负责日志收集。职责是监控本地日志文件的变化，并输出到 Redis 缓存起来。实际部署是我们也可以考虑一些替代方案，比方说用 rsyslog 来取代 Shipper，这样就可以减少在各个服务器配置 Logstash 的工作量，也可以结合 logrotate 进行版本管理，具体方式很多，可以根据需要选择最合适的方式，切换的成本也不会特别高，只要原始日志不丢，其他都好说。

**Broker** 可以看作是日志集线器，可以连接多个 Shipper 和多个 Indexer。实际情况中是否需要采用 Redis 做存储集中，亦或是通过某台服务器来完成日志汇总，都是可以根据工作量来权衡的，不过如果对性能要求较高，可能需要接入 Redis。

**Indexer** 负责日志存储。在这个架构中会从 Redis 接收日志，写入到本地文件。因为架构比较灵活，如果不想用 Logstash 的存储，也可以对接到 Elasticsearch，这也就是前面所说的 ELK 的套路了。

一个 Logstash 进程可以有多个输入源，同时也可以有多个输出源，输入输出均支持过滤和改写，非常灵活。如果需要 Broker 的话，官方推荐 Redis，因为其支持订阅发布和队列两种模式，能够自行根据需求选用。

为了防止异常终止期间的数据丢失，Logstash 具有持久性队列功能，将消息队列存储在磁盘上。 持久队列在 Logstash 中提供数据的持久性。

queue.type：persisted

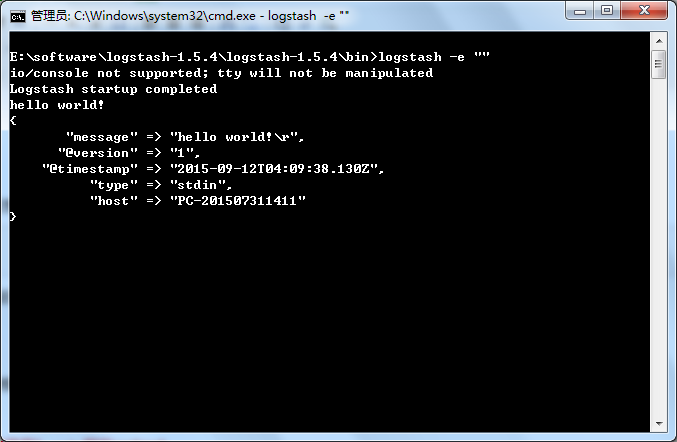
queue.max\_bytes：8gb

指定这些设置后，Logstash 将缓存磁盘上的事件，直到队列的大小达到8gb。当队列充满 unACKed 的事件，并且已达到大小限制时，Logstash 将不再接受新的事件。

通过命令行，进入到logstash/bin目录，执行下面的命令：

logstash -e ""

可以看到提示下面信息（这个命令稍后介绍），输入hello world!

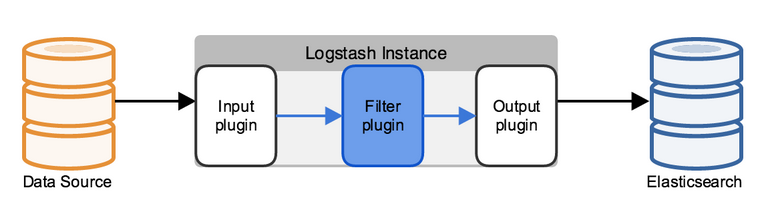


　　可以看到logstash为我们自动添加了几个字段，时间戳@timestamp，版本@version，输入的类型type，以及主机名host。

Logstash使用管道方式进行日志的搜集处理和输出。有点类似\*NIX系统的管道命令 xxx | ccc | ddd，xxx执行完了会执行ccc，然后执行ddd。

在logstash中，包括了三个阶段:

**输入input --> 处理filter（不是必须的） --> 输出output**



-f：通过这个命令可以指定Logstash的配置文件，根据配置文件配置logstash

-e：后面跟着字符串，该字符串可以被当做logstash的配置（如果是“” 则默认使用stdin作为输入，stdout作为输出）

-l：bin/logstash -l logs/logstash.log命令来统一存储日志。

-t：测试配置文件是否正确，然后退出。

## **配置文件说明**

前面介绍过logstash基本上由三部分组成，input、output以及用户需要才添加的filter，因此标准的配置文件格式如下：

input {...}

filter {...}

output {...}

## **常用插件配置：**

input类插件，常用的插件：file、tcp、udp、syslog，beats等。

①.file插件：  
file插件字段解释：  
codec => #可选项，默认是plain，可设置其他编码方式。  
discover\_interval => #可选项，logstash多久检查一下path下有新文件，默认15s。  
exclude => #可选项，排除path下不想监听的文件。  
sincedb\_path => #可选项，记录文件以及文件读取信息位置的数据文件。~/.sincedb\_xxxx  
sincedb\_write\_interval => #可选项，logstash多久写一次sincedb文件，默认15s.  
stat\_interval => #可选项，logstash多久检查一次被监听文件的变化，默认1s。  
start\_position => #可选项，logstash从哪个位置读取文件数据，默认从尾部，值为：end。初次导入，设置为：beginning。  
path => #必选项，配置文件路径，可定义多个。  
tags => #可选项，在数据处理过程中，由具体的插件来添加或者删除的标记。  
type => #可选项，自定义处理时间类型。比如nginxlog。

input{

file{

start\_position => "beginning"

path => ["/tmp/file1.log","/tmp/file2.log"]

type => 'filelog'

}

}

output{

stdout{}

}

②.tcp插件:  
tcp插件字段解释：  
add\_field => #可选项，默认{}。  
codec => #可选项，默认plain。  
data\_timeout => #可选项，默认-1。  
host => #可选项，默认0.0.0.0。  
mode => #可选项，值为["server","client"]之一，默认为server。  
port => #必选，端口。  
ssl\_cacert => #可选项，定义相关路径。  
ssl\_cert => #可选项，定义相关路径。  
ssl\_enable => #可选项，默认为false。  
ssl\_key => #可选项，定义相关路径。  
ssl\_key\_passphrase => #可选项，默认nil  
ssl\_verify => #可选项，默认false。  
tags => #可选项  
type => #可选项

input{

tcp{

host => "127.0.0.1"

port => 8888

type => "tcplog"

}

}

output{

stdout{}

}

③udp插件：  
udp插件字段解释：  
add\_field => #可选项，默认{}。  
host => #可选项，默认0.0.0.0。  
queue\_size => #默认2000  
tags => #可选项  
type => #可选项  
workers => #默认为2

input{

udp{

host => "127.0.0.1"

port => 9999

}

}

output{

stdout{}

}

④.syslog插件：

input{

syslog{

host => "127.0.0.1"

port => 518

type => "syslog"

}

}

output{

stdout{}

}

Mysql增量：

下面使用 增量 来新增数据,需要在jdbc.conf配置文件中做如下修改:

input {

stdin {

}

jdbc {

# mysql 数据库链接,test为数据库名

jdbc\_connection\_string => "jdbc:mysql://127.0.0.1:3306/test"

# 用户名和密码

jdbc\_user => "root"

jdbc\_password => "root"

# 驱动

jdbc\_driver\_library => "G:\Developer\Elasticsearch5.5.1\ES5\logstash-5.5.1\bin\mysql\mysql-connector-java-5.1.9.jar"

# 驱动类名

jdbc\_driver\_class => "com.mysql.jdbc.Driver"

#处理中文乱码问题

codec => plain { charset => "UTF-8"}

#使用其它字段追踪，而不是用时间

use\_column\_value => true

#追踪的字段

tracking\_column => id

record\_last\_run => true

#上一个sql\_last\_value值的存放文件路径, 必须要在文件中指定字段的初始值

last\_run\_metadata\_path => "G:\Developer\Elasticsearch5.5.1\ES5\logstash-5.5.1\bin\mysql\station\_parameter.txt"

#开启分页查询

jdbc\_paging\_enabled => true

jdbc\_page\_size => 300

# 执行的sql 文件路径+名称

statement\_filepath => "G:\Developer\Elasticsearch5.5.1\ES5\logstash-5.5.1\bin\mysql\jdbc.sql"

# 设置监听间隔 各字段含义（由左至右）分、时、天、月、年，全部为\*默认含义为每分钟都更新

schedule => "\* \* \* \* \*"

# 索引类型

type => "jdbc"

}

}

filter {

json {

source => "message"

remove\_field => ["message"]

}

}

output {

elasticsearch {

hosts => ["localhost:9200"] # ES的IP地址及端口

index => "article" # 索引名称

document\_id => "%{id}" # 自增ID

}

stdout {

codec => json\_lines # JSON格式输出

}

}

Mysql全量

input {

stdin {

}

jdbc {

# mysql 数据库链接,test为数据库名

jdbc\_connection\_string => "jdbc:mysql://127.0.0.1:3306/test"

# 用户名和密码

jdbc\_user => "root"

jdbc\_password => "root"

# 驱动

jdbc\_driver\_library => "G:\Developer\Elasticsearch5.5.1\ES5\logstash-5.5.1\bin\mysql\mysql-connector-java-5.1.9.jar"

# 驱动类名

jdbc\_driver\_class => "com.mysql.jdbc.Driver"

jdbc\_paging\_enabled => "true"

jdbc\_page\_size => "50000"

# 执行的sql 文件路径+名称

statement\_filepath => "G:\Developer\Elasticsearch5.5.1\ES5\logstash-5.5.1\bin\mysql\jdbc.sql"

# 设置监听间隔 各字段含义（由左至右）分、时、天、月、年，全部为\*默认含义为每分钟都更新

schedule => "\* \* \* \* \*"

type => "jdbc" # 索引类型

}

}

filter {

json {

source => "message"

remove\_field => ["message"]

}

}

output {

elasticsearch {

hosts => ["localhost:9200"] # ES的IP地址及端口

index => "article" # 索引名称

document\_id => "%{id}" # 自增ID 需要关联的数据库中有有一个id字段，对应索引的id号

}

stdout {

codec => json\_lines # JSON格式长文本输出

}

}

codec类插件，常用的插件：plain、json、json\_lines、rubydebug、multiline等。  
①.plain插件：

input{

stdin{

codec => "plain"

}

}

output{

stdout{}

}

②.json插件：

input{

stdin{}

}

output{

stdout{

codec => "json"

}

}

⑤.multiline插件：(处理错误日志)  
multiline插件字段：  
charset => #字符编码，可选  
max\_bytes => #bytes类型，设置最大的字节数，可选  
max\_lines => #number类型，设置最大的行数，默认是500行，可选  
multiline\_tag => #string类型，设置一个事件标签，默认是"multiline" ，可选  
pattern => #string 类型，设置匹配的正则表达式 ，必选   
patterns\_dir => #array类型，可以设置多个正则表达式，可选  
negate => #boolean类型，设置正向匹配还是反向匹配，默认是false，可选  
what => #设置未匹配的内容是向前合并还是向后合并，previous, next 两个值选择，必选

input{

stdin{

codec => multiline{

pattern => "^\["

negate => true

what => "previous"

}

}

}

output{

stdout{}

}

**filter插件**，常用的filter插件：json、grok等。

①.json插件：  
add\_field => #hash（可选项），默认{}  
add\_tag => #array（可选项），默认[]  
remove\_field => #array（可选项），默认[]  
remove\_tag => #array（可选项），默认[]  
source => #string（必选项）  
target => #string（可选项）

**output插件：**  
①.file插件  
②.tcp/udp插件  
③.redis/kfaka  
④.elasticsearch

input{

redis{

host => 'redis-server'

port => '6379'

data\_type => 'list'

key => 'lb'

codec => 'json'

}

}

**Logstash处理json格式日志文件**

假设日志文件中的每一行记录格式为json的，如：

{"Method":"JSAPI.JSTicket","Message":"JSTicket:kgt8ON7yVITDhtdwci0qeZg4L-Dj1O5WF42Nog47n\_0aGF4WPJDIF2UA9MeS8GzLe6MPjyp2WlzvsL0nlvkohw","CreateTime":"2015/10/13 9:39:59","AppGUID":"cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d","\_PartitionKey":"cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d","\_RowKey":"1444700398710\_ad4d33ce-a9d9-4d11-932e-e2ccebdb726c","\_UnixTS":1444700398710}

默认配置下，logstash处理插入进elasticsearch后，查到的结果是这样的：

{

"\_index": "logstash-2015.10.16",

"\_type": "voip\_feedback",

"\_id": "sheE9eXiQASMDVtRJ0EYcg",

"\_version": 1,

"found": true,

"\_source": {

"message": "{\"Method\":\"JSAPI.JSTicket\",\"Message\":\"JSTicket:kgt8ON7yVITDhtdwci0qeZg4L-Dj1O5WF42Nog47n\_0aGF4WPJDIF2UA9MeS8GzLe6MPjyp2WlzvsL0nlvkohw\",\"CreateTime\":\"2015/10/13 9:39:59\",\"AppGUID\":\"cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d\",\"\_PartitionKey\":\"cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d\",\"\_RowKey\":\"1444700398710\_ad4d33ce-a9d9-4d11-932e-e2ccebdb726c\",\"\_UnixTS\":1444700398710}",

"@version": "1",

"@timestamp": "2015-10-16T00:39:51.252Z",

"type": "voip\_feedback",

"host": "ipphone",

"path": "/usr1/data/voip\_feedback.txt"

}}

即会将json记录做为一个字符串放到”message”下，但是我是想让logstash自动解析json记录，将各字段放入elasticsearch中。有三种配置方式可以实现。

**第一种，直接设置format => json**

file {

type => "voip\_feedback"

path => ["/usr1/data/voip\_feedback.txt"]

format => json

sincedb\_path => "/home/jfy/soft/logstash-1.4.2/voip\_feedback.access"

}

这种方式查询出的结果是：

{

"\_index": "logstash-2015.10.16",

"\_type": "voip\_feedback",

"\_id": "NrNX8HrxSzCvLl4ilKeyCQ",

"\_version": 1,

"found": true,

"\_source": {

"Method": "JSAPI.JSTicket",

"Message": "JSTicket:kgt8ON7yVITDhtdwci0qeZg4L-Dj1O5WF42Nog47n\_0aGF4WPJDIF2UA9MeS8GzLe6MPjyp2WlzvsL0nlvkohw",

"CreateTime": "2015/10/13 9:39:59",

"AppGUID": "cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d",

"\_PartitionKey": "cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d",

"\_RowKey": "1444700398710\_ad4d33ce-a9d9-4d11-932e-e2ccebdb726c",

"\_UnixTS": 1444700398710,

"@version": "1",

"@timestamp": "2015-10-16T00:16:11.455Z",

"type": "voip\_feedback",

"host": "ipphone",

"path": "/usr1/data/voip\_feedback.txt"

}}

可以看到，json记录已经被直接解析成各字段放入到了\_source中，但是原始记录内容没有被保存

**第二种，使用codec => json**

file {

type => "voip\_feedback"

path => ["/usr1/data/voip\_feedback.txt"]

sincedb\_path => "/home/jfy/soft/logstash-1.4.2/voip\_feedback.access"

codec => json {

charset => "UTF-8"

}

}

**第三种，使用filter json**

filter {

if [type] == "voip\_feedback" {

json {

source => "message"

#target => "doc"

#remove\_field => ["message"]

}

}

}

这种方式查询出的结果是这样的：

{

"\_index": "logstash-2015.10.16",

"\_type": "voip\_feedback",

"\_id": "CUtesLCETAqhX73NKXZfug",

"\_version": 1,

"found": true,

"\_source": {

"message": "{\"Method222\":\"JSAPI.JSTicket\",\"Message\":\"JSTicket:kgt8ON7yVITDhtdwci0qeZg4L-Dj1O5WF42Nog47n\_0aGF4WPJDIF2UA9MeS8GzLe6MPjyp2WlzvsL0nlvkohw\",\"CreateTime\":\"2015/10/13 9:39:59\",\"AppGUID\":\"cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d\",\"\_PartitionKey\":\"cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d\",\"\_RowKey\":\"1444700398710\_ad4d33ce-a9d9-4d11-932e-e2ccebdb726c\",\"\_UnixTS\":1444700398710}",

"@version": "1",

"@timestamp": "2015-10-16T00:28:20.018Z",

"type": "voip\_feedback",

"host": "ipphone",

"path": "/usr1/data/voip\_feedback.txt",

"Method222": "JSAPI.JSTicket",

"Message": "JSTicket:kgt8ON7yVITDhtdwci0qeZg4L-Dj1O5WF42Nog47n\_0aGF4WPJDIF2UA9MeS8GzLe6MPjyp2WlzvsL0nlvkohw",

"CreateTime": "2015/10/13 9:39:59",

"AppGUID": "cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d",

"\_PartitionKey": "cb54ba2d-1d38-45f2-9ed1-abff0bf7dd3d",

"\_RowKey": "1444700398710\_ad4d33ce-a9d9-4d11-932e-e2ccebdb726c",

"\_UnixTS": 1444700398710,

"tags": [

"111",

"222"

]

}}

可以看到，原始记录被保存，同时字段也被解析保存。如果确认不需要保存原始记录内容，可以加设置：remove\_field => [“message”]