2017年2月14日 星期二

Logstash配置说明

LOGSTASH的配置可以分为三块去展开说明，INPUT，CODEC，FILTER，OUTPUT。

input配置说明

1.读取文件(File)

配置示例：

```
input {
        file {
                        path => [ “/data/rdd/multipaths/afa/**/*.log”, “/var/log/access.log” ]
                        sincedb_path => "/tmp/logstashdb"
                        start_position => "beginning"
                        stat_interval => 3
                        discover_interval => 3
                        ignore_older => 864000
                        close_older => 1800
                        max_open_files => 1024000
        }
}
```

解释

path

指定监控的文件路径，可一次性监控多个目录。

说明：LogStash::Inputs::File 只是在进程运行的注册阶段初始化一个 FileWatch 对象。所以它不能支持类似 fluentd 那样的 path => "/path/to/%{+yyyy/MM/dd/hh}.log" 写法。达到相同目的，你只能写成 path => "/path/to/*/*/*/*.log"。FileWatch 模块提供了一个稍微简单一点的写法：/path/to/**/*.log，用 ** 来缩写表示递归全部子目录。

discover_interval

logstash 每隔多久去检查一次被监听的 path 下是否有新文件。默认值是 15 秒。

说明：对于常规服务产生的日志，使用默认配置即可。但是对于类似afa，afe的超多日志文件，经过测试发现，同时处理afa日志400各文件（单条日志约12kb），最大延迟是2分钟。故为了兼顾线上日志的低延迟，此值需要适当调整。

此值的大小“直接”影响了数据的延迟时间，故需要设置合理值。

exclude

不想被监听的文件可以排除出去，这里跟 path 一样支持 glob 展开。

close_older

一个已经监听中的文件，如果超过这个值的时间内没有更新内容，就关闭监听它的文件句柄。默认是 3600 秒，即一小时。

说明：如果收集的服务日志文件数目过多（大于1024），此值需要合理调整，保障logstash同时监控的文件数目不要太多。

ignore_older

在每次检查文件列表的时候，如果一个文件的最后修改时间超过这个值，就忽略这个文件。默认是 86400 秒，即一天。

sincedb_path

logstash会记录一个叫 .sincedb 的数据库文件来跟踪被监听的日志文件的当前读取位置。所以，不要担心 logstash 会漏过你的数据。如果不想用默认的路径可以通过这个配置定义 sincedb 文件到其他位置。

说明：如果是测试环境，想要每次重新（从头）开始收集文件，把此目录配置为/dev/null即可，只需要再次启动进程即可完成重复收集。但是请确定你要这么做，且在生产环境，给定需要的合理配置。

sincedb_write_interval

logstash 每隔多久写一次 sincedb 文件，默认是 15 秒。

说明：此值一般不用修改，因为是收集日志数据，但是你依然可以把值调小，防止logstash出现故障时，下次启动后会重复收集部分数据。

stat_interval

logstash 每隔多久检查一次被监听文件状态（是否有更新），默认是 1 秒。

说明：此值一般不用修改。但是收集的日志文件数目过多，应该适当调大此值。测试afa的80万+文件时，此值设置的为15秒。而此值的大小“直接”影响了数据的延迟时间，故需要设置合理值。

start_position

logstash 从什么位置开始读取文件数据，默认是结束位置，也就是说 logstash 进程会以类似 tail -F 的形式运行。如果你是要导入原有数据，把这个设定改成 "beginning"，logstash 进程就从头开始读取，类似 less +F 的形式运行。

说明：仅在该文件从未被监听过的时候起作用。如果 sincedb 文件中已经有这个文件的 inode 记录了，那么 logstash 依然会从记录过的 pos 开始读取数据。

因为 windows 平台上没有 inode 的概念，Logstash 某些版本在 windows 平台上监听文件不是很靠谱。

2.读取网络数据(TCP)

配置示例

```
input {
    tcp {
        port => 8888
        mode => "server"
        ssl_enable => false
    }
}
```

常见场景

目前来看，LogStash::Inputs::TCP 最常见的用法就是配合 nc 命令导入旧数据。在启动 logstash 进程后，在另一个终端运行如下命令即可导入数据：

```
# nc 127.0.0.1 8888 < olddata
```

这种做法比用 LogStash::Inputs::File 好，因为当 nc 命令结束，就知道数据导入完毕了。而用 input/file 方式，logstash 进程还会一直等待新数据输入被监听的文件，不能直接看出是否任务完成了。

说明：虽然 LogStash::Inputs::TCP 用 Ruby 的 Socket 和 OpenSSL 库实现了高级的 SSL 功能，但 Logstash 本身只能在 SizedQueue 中缓存 20 个事件。故建议在生产环境中换用其他消息队列。

codec配置说明

编码插件(Codec)

Codec 是 logstash 从 1.3.0 版开始新引入的概念(Codec 来自 Coder/decoder 两个单词的首字母缩写)。在此之前，logstash 只支持纯文本形式输入，然后以过滤器处理它。但现在，我们可以在输入 期处理不同类型的数据。codec 的引入，使得 logstash 可以更好更方便的与其他有自定义数据格式的运维产品共存，比如 graphite、fluent、netflow、collectd，以及使用 msgpack、json、edn 等通用数据格式的其他产品等。

Logstash 不只是一个input | filter | output 的数据流，而是一个 input | decode | filter | encode | output 的数据流！

1.采用 JSON 编码

作为一款Agent部署在生产环境的节点上收集日志的服务，改组件自身对资源的消耗就应该得到很好控制。故直接输入预定义好的 JSON 数据，这样就可以省略掉 filter/grok 配置，从而大大节约CPU资源。
配置示例

```
input {
    file {
        path => "/var/log/nginx/access.log_json""
        codec => "json"
    }
}
```

2.合并多行数据(Multiline)

有些时候，应用程序调试日志会包含非常丰富的内容，为一个事件打印出很多行内容。这种日志通常都很难通过命令行解析的方式做分析。logstash 正为此提供了 codec/multiline 插件。

配置示例

```
input {
        file {
                path => ["/data/rdd/multipaths/afa/**/*.log"]
                sincedb_path => "/tmp/logstashdb"
                start_position => "beginning"
                stat_interval => 3
                discover_interval => 3
                ignore_older => 864000
                close_older => 1800
                max_open_files => 1024000
                codec => multiline {
                                pattern => "交易耗时"
                                negate => true
                                what => "next"
                                max_lines => 1000
                                max_bytes => "10MiB"
                }
        }
```

}

解释

其实这个插件的原理很简单，就是把当前行的数据添加到前面一行后面，，直到新进的当前行匹配“交易耗时”正则为止。

pattern
要匹配的正则表达式。

说明：其他主流产品的配置参考： https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns

negate
否定正则表达式（如果没有匹配的话）

what
可以为 previous 或 next。如果正则表达式匹配了，那么该事件是属于下一个或是前一个事件。

max_lines
单次匹配最大行数。此值默认是500，可以合理调整来控制（防止）出现特殊异常产生的回调stack信息。

filter配置说明

过滤器插件(Filter)，丰富的过滤器插件的存在是 logstash 威力如此强大的重要因素。名为过滤器，其实提供的不单单是过滤的功能。可以提供对数据的多种加工和预操作。

1.Grok 正则捕获

Grok 是 Logstash 最重要的插件。你可以在 grok 里预定义好命名正则表达式，在稍后(grok参数或者其他正则表达式里)引用它。

具体介绍请移步： https://github.com/hzruandd/bigdata/blob/master/logstash/grok.md

2.JSON 编解码

上面已经介绍了在 codec 中使用 JSON 编码。但是，有些日志可能是一种复合的数据结构，其中只是一部分记录是 JSON 格式的。这时候，我们依然需要在 filter 阶段，单独启用 JSON 解码插件。

配置示例

```
filter {
    json {
        source => "message"
        target => "jsoncontent"
    }
}
```

如果不打算使用多层结构的话，删掉 target 配置即可。

3.数据修改(Mutate)

filters/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。包括类型转换，字符串处理和字段处理等。

类型转换filters/mutate 插件最初诞生时的唯一功能。可以设置的转换类型包括："integer"，"float" 和 “string"。

配置示例

```
filter {
    mutate {
        convert => ["request_time", "float"]
    }
}
```

说明：mutate 除了转换简单的字符值，还支持对数组类型的字段进行转换，即将 ["1","2"] 转换成 [1,2]。

字符串支持的操作有：gsub，spilt，join，merge；

字段处理支持的操作有：
rename，重命名某个字段，如果目的字段已经存在，会被覆盖掉：

```
filter {
    mutate {
```

```
        rename => ["syslog_host", "host"]
    }
}
```

update，更新某个字段的内容。如果字段不存在，不会新建。
replace，作用和 update 类似，但是当字段不存在的时候，它会起到 add_field 参数一样的效果，自动添加新的字段。

执行次序：

需要注意的是，filter/mutate 内部是有执行次序的。其次序如下：

```
rename(event) if @rename
update(event) if @update
replace(event) if @replace
convert(event) if @convert
gsub(event) if @gsub
uppercase(event) if @uppercase
lowercase(event) if @lowercase
strip(event) if @strip
remove(event) if @remove
split(event) if @split
join(event) if @join
merge(event) if @merge
```

4.随心所欲的Ruby 处理

filters/ruby 插件将会是一个非常有用的工具，它支持更加丰富的数据预处理，节约服务器资源。
这部分的介绍后续继续补充。

5.split 拆分事件

上面介绍了multiline 插件将多行数据合并进一个事件里，那么反过来，也可以把一行数据，拆分成多个事件。这就是 split 插件完成的工作内容。

配置示例

```
filter {
    split {
        field => "message"
```

```
        terminator => "#"
    }
}
```

output配置说明

1.数据流入Elasticsearch

配置示例

```
output {
        elasticsearch {
                hosts => ["10.8.6.170:9200"]
                index => "logstash-to-es-%{+YYYY.MM.dd}"
        }
}
```

解释

index
写入的 ES 索引的名称，这里可以使用变量。为了更贴合日志场景，Logstash 提供了 %{+YYYY.MM.dd} 这种写法。

说明：在语法解析的时候，看到以 + 号开头的，就会自动认为后面是时间格式，尝试用时间格式来解析后续字符串。所以，之前处理过程中不要给自定义字段取个加号开头的名字。此外，注意索引名中不能有大写字母，否则 ES 在日志中会报 InvalidIndexNameException，但是 Logstash 不会报错，这个错误比较隐晦，也容易掉进这个坑中。

host

Logstash 1.4.2 在 transport 和 http 协议的情况下是固定连接指定 host 发送数据。从 1.5.0 开始，host 可以设置数组，它会从节点列表中选取不同的节点发送数据，达到 Round-Robin 负载均衡的效果。

2.数据流入kafka

配置示例

```
output {
    kafka {
```

```
        bootstrap_servers => "10.8.6.170:9092"
        topic_id => "3w-test-afa"
    }
}
```

解释

bootstrap_servers
kafka节点信息（ip：port），可以配置多个，使用逗号“，”分割即可。

topic_id
写入kafka集群的topic名称