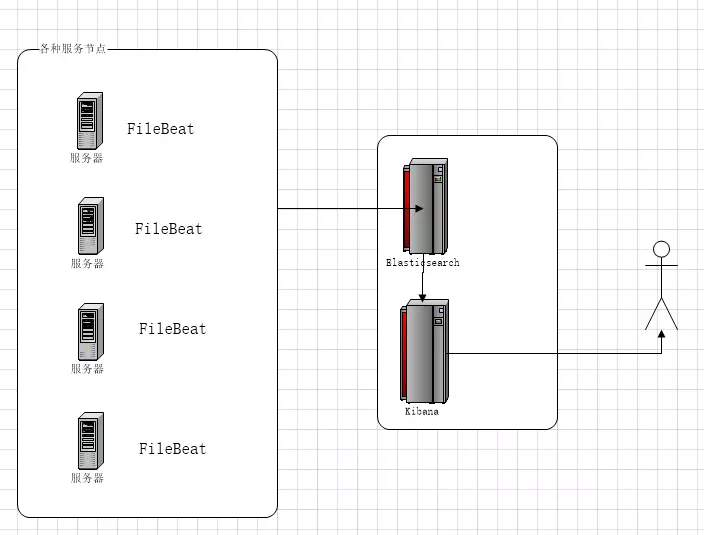
EFK详解

EKF架构图:



1. EFK简介

EFK不是一个软件，而是一套解决方案，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，是目前主流的一种日志系统。

EFK是三个开源软件的缩写，分别表示：Elasticsearch , FileBeat, Kibana , 其中ELasticsearch负责日志保存和搜索，FileBeat负责收集日志，Kibana 负责界面

Elasticsearch：是个开源分布式搜索引擎，提供搜集、分析、存储数据三大功能。

特点：分布式，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。

kibana：是一个开源和免费的工具，它可以为 Logstash 、Beats和 ElasticSearch提供友好的web可视化界面，帮助您汇总、分析和搜索重要数据日志。

Filebeat由两个主要组件组成：prospector 和harvester。

启动Filebeat时，它会启动一个或多个查找器(prospector)，查看您为日志文件指定的本地路径。 对于prospector 所在的每个日志文件，prospector 启动harvester。 每个harvester都会为新内容读取单个日志文件，并将新日志数据发送到libbeat，后者将聚合事件并将聚合数据发送到您为Filebeat配置的输出。

Filebeat隶属于Beats。目前Beats包含六种工具：

Packetbeat（搜集网络流量数据）

Metricbeat（搜集系统、进程和文件系统级别的 CPU 和内存使用情况等数据）

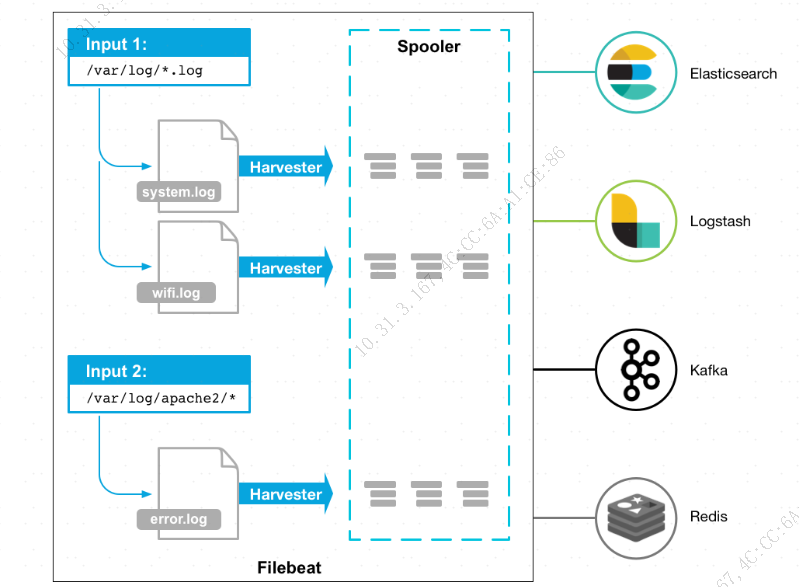
Filebeat（搜集文件数据）

Winlogbeat（搜集 Windows 事件日志数据）

Auditbeat（ 轻量型审计日志采集器）

Heartbeat（轻量级服务器健康采集器）

filebeat原理



Filebeat 有两大部分，inputs 和 harvesters，inputs 负责找文件（类似 find 命令）和管理 harvesters，一个 harvester 则和一个文件一一对应，一行行读然后发送给 output（类似tail -f）

[https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html](https://links.jianshu.com/go?to=https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html" \t "https://www.jianshu.com/p/_blank)

Logstash替换成了FileBeat，因为Filebeat相对于Logstash来说有2个好处：

1、侵入低，无需修改程序目前任何代码和配置

2、相对于Logstash来说性能高，Logstash对于IO占用很大

1. docker安装efk

安装es

sudo docker run --name elasticsearch --privileged=true -p 9200:9200 -p 9300:9300

-e "discovery.type=single-node" -v

/docker/elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml

-v /docker/elasticsearch/data:/usr/share/elasticsearch/data

-v /docker/elasticsearch/plugins:/usr/share/elasticsearch/plugins

-d elasticsearch:7.12.0

不加--privileged=true会报以下错误

Exception in thread "main" SettingsException[Failed to load settings from /usr/share/elasticsearch/config/elasticsearch.yml]; nested: AccessDeniedException[/usr/share/elasticsearch/config/elasticsearch.yml];

at org.elasticsearch.node.InternalSettingsPreparer.prepareEnvironment(InternalSettingsPreparer.java:74)

at org.elasticsearch.cli.EnvironmentAwareCommand.createEnv(EnvironmentAwareCommand.java:89)

at org.elasticsearch.cli.EnvironmentAwareCommand.createEnv(EnvironmentAwareCommand.java:80)

at org.elasticsearch.cli.EnvironmentAwareCommand.execute(EnvironmentAwareCommand.java:75)

at org.elasticsearch.cli.Command.mainWithoutErrorHandling(Command.java:116)

at org.elasticsearch.cli.MultiCommand.execute(MultiCommand.java:80)

at org.elasticsearch.cli.Command.mainWithoutErrorHandling(Command.java:116)

at org.elasticsearch.cli.Command.main(Command.java:79)

at org.elasticsearch.common.settings.KeyStoreCli.main(KeyStoreCli.java:32)

Caused by: java.nio.file.AccessDeniedException: /usr/share/elasticsearch/config/elasticsearch.yml

at java.base/sun.nio.fs.UnixException.translateToIOException(UnixException.java:90)

at java.base/sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:106)

at java.base/sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:111)

at java.base/sun.nio.fs.UnixFileSystemProvider.newByteChannel(UnixFileSystemProvider.java:218)

at java.base/java.nio.file.Files.newByteChannel(Files.java:375)

at java.base/java.nio.file.Files.newByteChannel(Files.java:426)

at java.base/java.nio.file.spi.FileSystemProvider.newInputStream(FileSystemProvider.java:420)

at java.base/java.nio.file.Files.newInputStream(Files.java:160)

at org.elasticsearch.common.settings.Settings$Builder.loadFromPath(Settings.java:1075)

at org.elasticsearch.node.InternalSettingsPreparer.prepareEnvironment(InternalSettingsPreparer.java:72)

... 8 more

Exception in thread "main" java.nio.file.AccessDeniedException: /usr/share/elasticsearch/plugins

at java.base/sun.nio.fs.UnixException.translateToIOException(UnixException.java:90)

at java.base/sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:106)

at java.base/sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:111)

at java.base/sun.nio.fs.UnixFileSystemProvider.newDirectoryStream(UnixFileSystemProvider.java:431)

at java.base/java.nio.file.Files.newDirectoryStream(Files.java:476)

at java.base/java.nio.file.Files.list(Files.java:3765)

at org.elasticsearch.tools.launchers.BootstrapJvmOptions.getPluginInfo(BootstrapJvmOptions.java:43)

at org.elasticsearch.tools.launchers.BootstrapJvmOptions.bootstrapJvmOptions(BootstrapJvmOptions.java:34)

at org.elasticsearch.tools.launchers.JvmOptionsParser.jvmOptions(JvmOptionsParser.java:137)

at org.elasticsearch.tools.launchers.JvmOptionsParser.main(JvmOptionsParser.java:86)

安裝kibana

sudo docker run --restart=always --link f070ba380dd9:elasticsearch --name=kibana --privileged=true -p 5601:5601

-itd elastic/kibana:7.7.1

安裝filebeat

docker run -d --name filebeat docker.elastic.co/beats/filebeat:7.7.1

docker ps

拷贝容器内部的filebeat目录到/docker/filebeat

docker cp filebeat:/usr/share/filebeat /docker/filebeat

docker run -d --restart=always --name=filebeat --privileged=true -v /docker/filebeat/:/usr/share/filebeat/ -v /docker/log/test/:/var/log/test/

-d docker.elastic.co/beats/filebeat:7.7.1

鏡像重啓報錯,用docker logs filebeat查看

Exiting: error loading config file: config file (“/opt/filebeat/filebeat.yml“) can only be writable

报错是因为安全原因不要其他用户写的权限，去掉写的权限就可以了，把777修改成755，转码后是493

Chmod 755 filebeat.yml

docker restart filebeat

filebeat.yml文件配置

filebeat.inputs:

- type: log

enabled: true

paths:

- /var/log/test/\*.log //收集日志的路徑

multiline.pattern: '^\d{4}-\d{2}-\d{2}' //pattern：正则表达式

//negate：true 或 false；默认是false，匹配pattern的行合并到上一行；true，不匹配pattern的行合并到上一行

multiline.negate: true

multiline.match: after //after 或 before，合并到上一行的末尾或开头

setup.kibana.host: "http://172.16.0.16:5601" //kibana地址

setup.dashboards.enabled: true

output.elasticsearch:

hosts: ["http://172.16.0.17:9200"] //es服務器主機

index: "filebeat-%{+yyyy.MM.dd}" //索引名稱

setup.template.name: "my-log"

setup.template.pattern: "my-log-\*"

//keys\_under\_root可以让字段位于根节点，默认为false

json.keys\_under\_root: false

#对于同名的key，覆盖原有key值

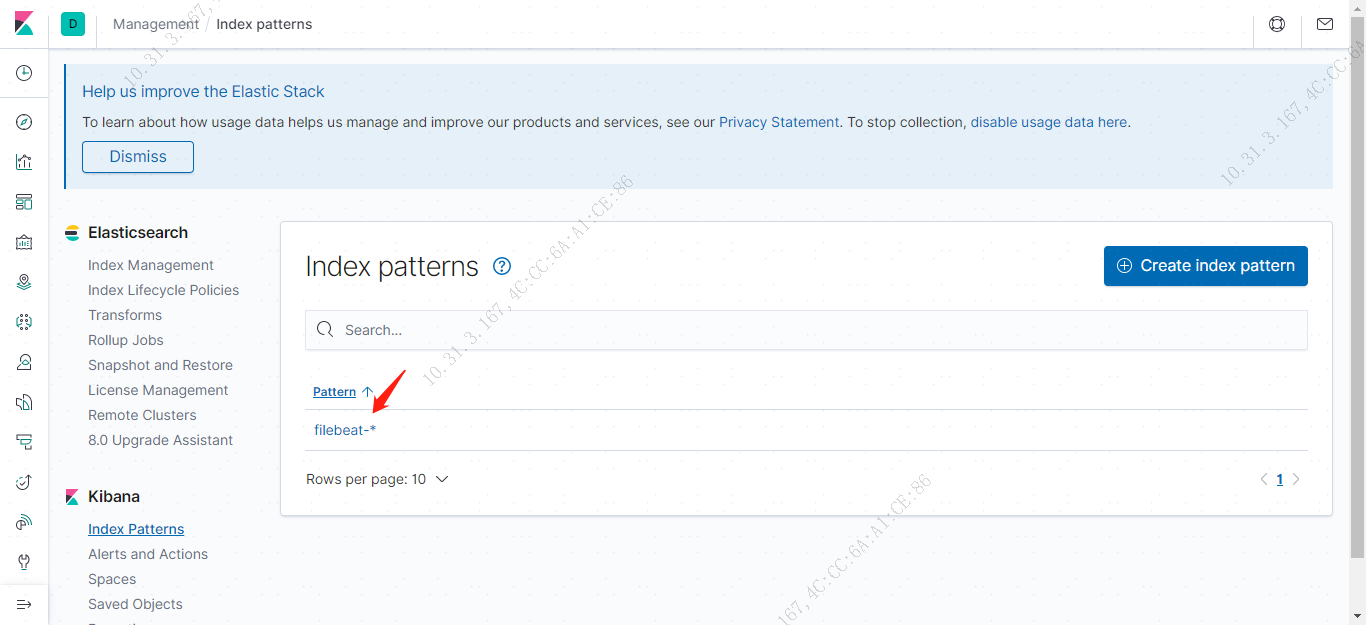
json.overwrite\_keys: true

processors:

- decode\_json\_fields:

fields: [""]

target: json



在docker/log/test/目录下写入一些内容测试

While true;do echo “filebeat”>>a.log;sleep 1;done



添加依赖

<dependency>

<groupId>net.logstash.logback</groupId>

<artifactId>logstash-logback-encoder</artifactId>

<version>6.6</version>

</dependency>

logback-spring.xml的配置

<?xml version="1.0" encoding="UTF-8"?>

<configuration scan="true" scanPeriod="60 seconds" debug="false">

<!-- 定义参数常量 -->

<!-- TRACE<DEBUG<INFO<WARN<ERROR -->

<!-- logger.trace("msg") logger.debug... -->

<property name="log.level" value="debug"/>

<property name="log.maxHistory" value="2"/>

<!--修改此处-->

<property name="log.filePath" value="./docker/log/test"/>

<property name="log.pattern" value="%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n"/>

<appender name="stash" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>${log.filePath}/elastic.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>${log.filePath}/elastic.%d{yyyy-MM-dd}.log.gz</fileNamePattern>

<totalSizeCap>20GB</totalSizeCap>

</rollingPolicy>

<encoder class="net.logstash.logback.encoder.LogstashEncoder"/>

</appender>

<root name="com.example" level="${log.level}" additivity="true">

<appender-ref ref="stash" />

</root>

</configuration>

nohup java -jar springbootdemo.jar>springboot.log & 把springboot的日志輸出到filebeat收集的日志目錄下

FileBeat的配置参考

<https://www.elastic.co/guide/en/beats/filebeat/current/configuring-howto-filebeat.html>

Log input配置详解

[https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-input-log.html](https://links.jianshu.com/go?to=https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-input-log.html" \t "https://www.jianshu.com/p/_blank)

基本例子

filebeat.inputs:

- type: log

paths:

- /var/log/system.log

- /var/log/wifi.log

- type: log

paths:

- "/var/log/apache2/\*"

fields:

apache: true

fields\_under\_root: true

inputs 可以配置多块（block）

paths 可以配置多个文件，文件路径和文件名都支持通配。

# ignore\_older 和 scan\_frequency

## 场景问题

场景1：路径下的历史文件可能很多，比如配置了按天分割，显然旧文件我们一般是不需要的。  
场景2：扫描频率如何控制，通配设置复杂的话频繁扫文件也是很大的开销。

## 解决方案

场景1是通过 ignore\_older 参数解决，意思是多久前的旧文件不扫描。  
比如设置为 1h，表示文件时间在 1h 之前的日志都不会被 input 模块搜集，直到有新日志产生。

场景2是通过 scan\_frequency 参数控制，表示多久扫描一次是否有新文件产生。  
比如设置 10s（默认），一个新文件产生 10s 后会被发现，或者一个旧文件（上面 ignore\_older）新产生了一行日志 10s 才发现这个文件。

# close\_\* 和 clean\_\*

被 harvester 获取的文件就一直拿着不放吗？文件重命名或者被删除后怎么办呢？

## close\_\* 配置簇

The close\_\* configuration options are used to close the harvester after a certain criteria or time. Closing the harvester means closing the file handler.

close\_inactive  
多久关闭文件，比如一个日志文件，10 分钟都没有读到新的内容就把文件句柄关闭。

这里的时间不是取决于文件的最后更新时间，而是 Filebeat 内部记录的时间，上次读到文件和这次尝试读文件的时间差。

官方建议设置的时间是比文件产生数据频率高一个数量级（默认5m），比如每秒都有日志产生，这个值就可以设置为 1m。

close\_renamed  
是否关闭 rename 的文件。

close\_removed  
默认开启。  
文件被删除后，就关闭文件句柄。

这个符合正常场景，一般日志清理程序都是清理很多天前的日志，这个时间远远大于 ignore\_older 和 close\_inactive。

## clean\_\* 配置簇

The clean\_\* options are used to clean up the state entries in the registry file.

Filebeat 内部记录了很多文件状态，保存在 data/registry/filebeat/data.json。如果不清理的话这个文件会越来越大，影响效率。

[{"source":"/var/log/test/a.log","offset":7794,"timestamp":"2022-04-19T08:58:58.144313065Z","ttl":-1,"type":"log","meta":null,"FileStateOS":{"inode":38606784,"device":64768}}]

~

clean\_inactive  
多久清理一次注册信息。默认值是0（不开启clean\_\*相关功能）

清理的文件信息需要保证这个文件已经不活跃了，所以这个值需要大于 ignore\_older + scan\_frequency。  
不然的话清理后这个文件又被发现，则会重头开始读取，这样就重了。

clean\_removed  
文件被删除后是否清理注册信息，默认开启。  
需要和 close\_removed 值保持一致

简单总结几个时间配置：  
clean\_inactive > ignore\_older + scan\_frequency > close\_inactive

推荐配置：

tail\_files: false

scan\_frequency: 10s

ignore\_older: 60m

close\_inactive: 10m

close\_renamed: true

close\_removed: true

clean\_inactive: 70m

clean\_removed: true

资源限制

在日志非常多机器负载高的时候加重机器负担，建议生产环境上需要对 Filebeat 资源进行限制：  
max\_procs 最多使用多少核，默认会全部使用，按机器情况限制为1-4核，不太会影响推送效率。

配置自动加载

[https://www.elastic.co/guide/en/beats/filebeat/current/\_live\_reloading.html](https://links.jianshu.com/go?to=https://www.elastic.co/guide/en/beats/filebeat/current/_live_reloading.html" \t "https://www.jianshu.com/p/_blank)

filebeat.config.inputs:

enabled: true

path: configs/\*.yml

reload.enabled: true

reload.period: 10s

具体的 input 配置文件放在 configs 文件夹下，例如：

- type: log

paths:

- /var/log/messages

- /var/log/\*.log

**Filebeat是如何工作的**

**Filebeat由两个主要组件组成：inputs 和 harvesters （直译：收割机，采集器）。这些组件一起工作以跟踪文件，并将事件数据发送到你指定的输出。**

**1.1. harvester是什么**

**一个harvester负责读取一个单个文件的内容。**

**harvester逐行读取每个文件（一行一行地读取每个文件），并把这些内容发送到输出。**

**每个文件启动一个harvester。**

**harvester负责打开和关闭这个文件，这就意味着在harvester运行时文件描述符保持打开状态。**

**在harvester正在读取文件内容的时候，文件被删除或者重命名了，那么Filebeat会续读这个文件。这就有一个问题了，就是只要负责这个文件的harvester没用关闭，那么磁盘空间就不会释放。默认情况下，Filebeat保存文件打开直到close\_inactive到达。**

**1.2. input是什么**

**一个input负责管理harvesters，并找到所有要读取的源。**

**如果input类型是log，则input查找驱动器上与已定义的glob路径匹配的所有文件，并为每个文件启动一个harvester。**

**每个input都在自己的Go例程中运行。**

**下面的例子配置Filebeat从所有匹配指定的glob模式的文件中读取行：**

**filebeat.inputs:**

**- type: log**

**paths:**

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

**- /var/path2/\*.log**

**1.3. Filebeat如何保持文件状态**

**Filebeat保存每个文件的状态，并经常刷新状态到磁盘上的注册文件（registry）。状态用于记住harvester读取的最后一个偏移量，并确保所有日志行被发送（到输出）。如果输出，比如Elasticsearch 或者 Logstash等，无法访问，那么Filebeat会跟踪已经发送的最后一行，并只要输出再次变得可用时继续读取文件。当Filebeat运行时，会将每个文件的状态新保存在内存中。当Filebeat重新启动时，将使用注册文件中的数据重新构建状态，Filebeat将在最后一个已知位置继续每个harvester。**

**对于每个输入，Filebeat保存它找到的每个文件的状态。因为文件可以重命名或移动，所以文件名和路径不足以标识文件。对于每个文件，Filebeat存储惟一标识符，以检测文件是否以前读取过。**

**如果你的情况涉及每天创建大量的新文件，你可能会发现注册表文件变得太大了。**

**（画外音：Filebeat保存每个文件的状态，并将状态保存到registry\_file中的磁盘。当重新启动Filebeat时，文件状态用于在以前的位置继续读取文件。如果每天生成大量新文件，注册表文件可能会变得太大。为了减小注册表文件的大小，有两个配置选项可用：clean\_remove和clean\_inactive。对于你不再访问且被忽略的旧文件，建议您使用clean\_inactive。如果想从磁盘上删除旧文件，那么使用clean\_remove选项。）**

**1.4. Filebeat如何确保至少投递一次（at-least-once）？**

**Filebeat保证事件将被投递到配置的输出中至少一次，并且不会丢失数据。Filebeat能够实现这种行为，因为它将每个事件的投递状态存储在注册表文件中。**

**在定义的输出被阻塞且没有确认所有事件的情况下，Filebeat将继续尝试发送事件，直到输出确认收到事件为止。**

**如果Filebeat在发送事件的过程中关闭了，则在关闭之前它不会等待输出确认所有事件。当Filebeat重新启动时，发送到输出（但在Filebeat关闭前未确认）的任何事件将再次发送。这确保每个事件至少被发送一次，但是你最终可能会将重复的事件发送到输出。你可以通过设置shutdown\_timeout选项，将Filebeat配置为在关闭之前等待特定的时间。**