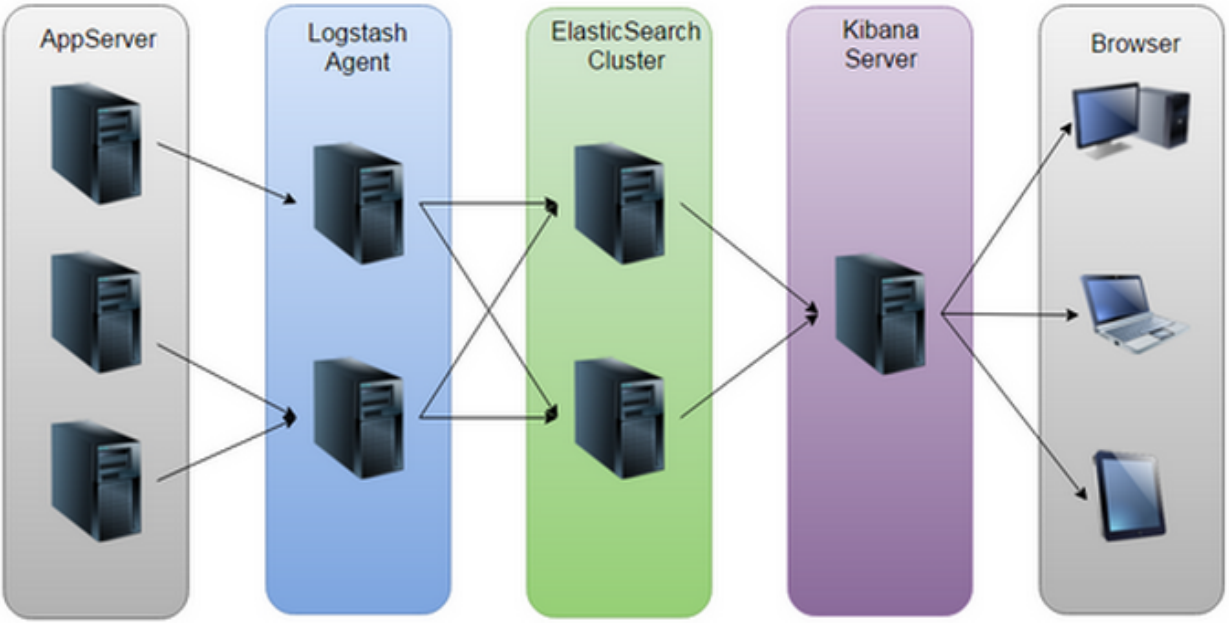
一.概念

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syslog，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。
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）ElasticSearch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，是第二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。
Elasticsearch本身就是分布式的，因此即便你只有一个节点，Elasticsearch默认也会对你的数据进行分片和副本操作，当你向集群添加新数据时，数据也会在新加入的节点中进行平衡。
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。
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。

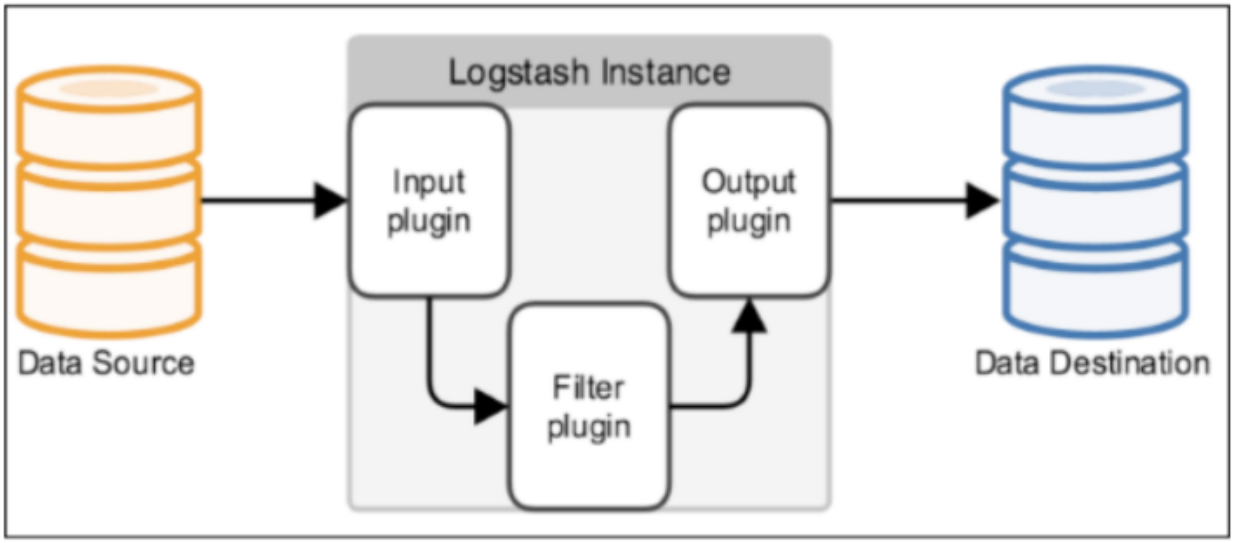
ELK工作原理展示图：



如上图：Logstash收集AppServer产生的Log，并存放到ElasticSearch集群中，而Kibana则从ES集群中查询数据生成图表，再返回给Browser。

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。



Input：输入数据到logstash。

一些常用的输入为：

file：从文件系统的文件中读取，类似于tail -f命令

syslog：在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。

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)
Outputs：outputs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outputs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。

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
    }
}
```

==========ELK整体方案==========

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。

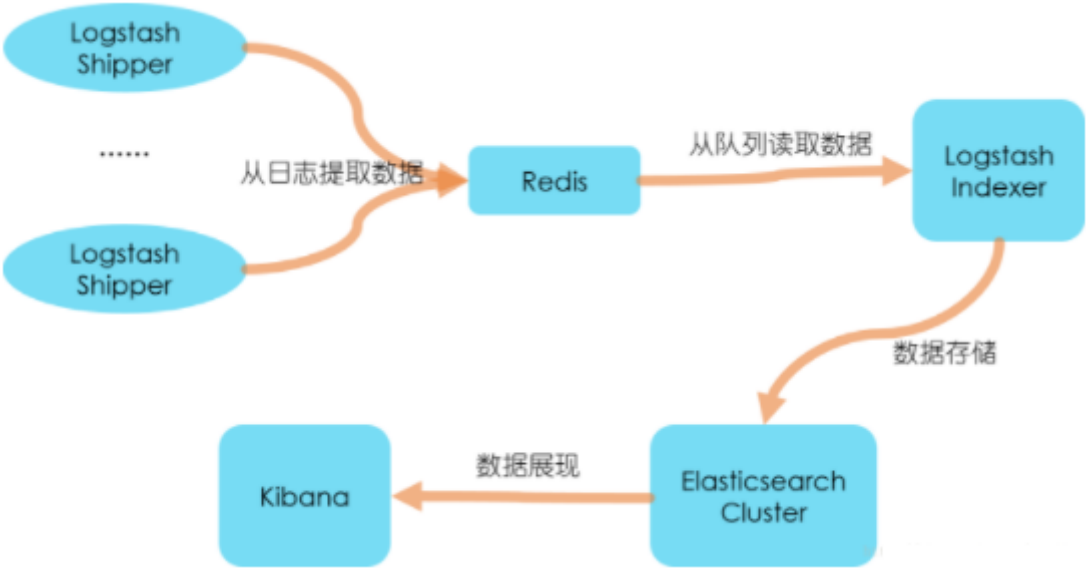
ELK的安装很简单，可以按照"下载->修改配置文件->启动"方法分别部署三个系统，也可以使用docker来快速部署。具体的安装方法这里不详细介绍，下面来看一个常见的部署方案，如下图所示，部署思路是：

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？

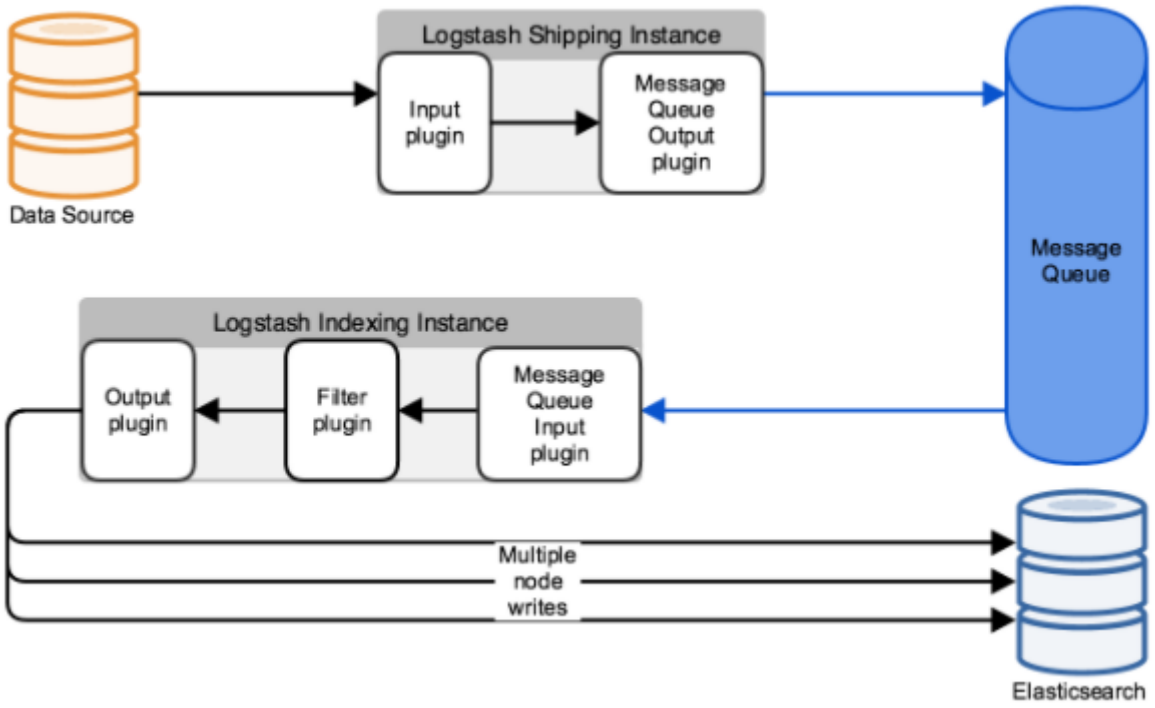
首先，采用这样的架构部署，有三点优势：

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。



https://www.cnblogs.com/kevingrace/p/5919021.html