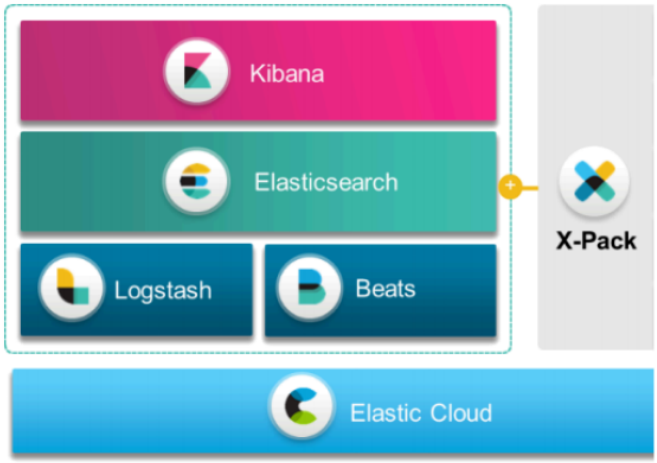
集中化日志平台Elastic Stack

一、Elastic Stack概念

1 Elastic Stack介绍

日志在项目中起到了性能监控、异常定位、数据分析等非常重要的作用。虽然主流工具如Tomcat，Nginx等都会生成日志文件。但在分布式架构中，不同的服务部署在不同的服务器上，这样就产生了日志量大，数据分散，搜索不方便等问题。于是我们需要搭建集中式日志平台，将所有节点上的日志进行统一管理，让日志数据最大限度的发挥作用。

Elastic公司提供了一整套搭建集中式日志平台的解决方案。最开始由Elasticsearch、Logstash、Kibana三个工具组成，简称ELK。在发展的过程中又有新成员Beats的加入，形成了Elastic Stack。



2 Elastic Stack组件

- Elasticsearch

  Elasticsearch基于java开发的分布式搜索引擎，他可以对日志数据进行存储、搜索、分析等操作。

- Kibana

  Kibana为Elasticsearch提供了友好的日志分析Web界面，通过仪表板等可视化组件展示日志数据。

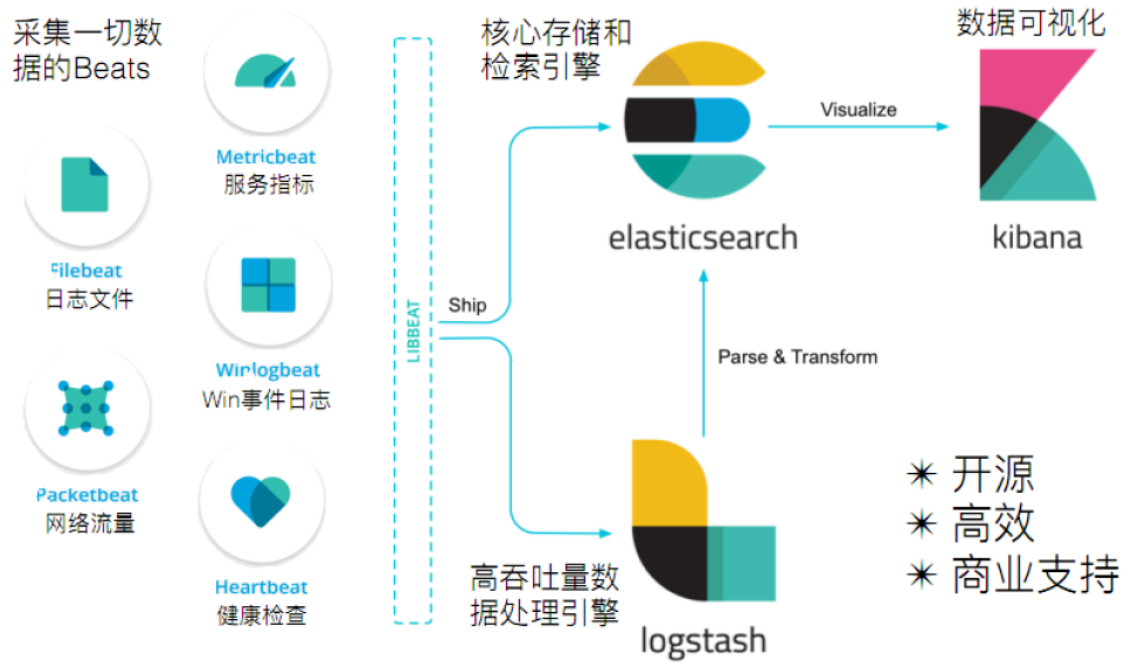
- Logstash

  Logstash可以对日志进行收集、分析处理，然后输出到存储系统如Elasticsearch中。

- Beats

  由于Logstash是在jvm中运行，收集数据时资源消耗比较大，elastic又推出了一系列轻量级的数据采集工具，这些工具统称为Beats，Beats收集的数据可以直接输出到Elasticsearch中，也可以通过Logstash处理后输出到Elasticsearch中。Beats有以下常用工具：

  - Filebeat：用于监控、收集服务器日志文件。
  - Metricbeat：可以监控、收集系统的CPU使用率、内存、磁盘IO等数据，以及 Apache、NGINX、MongoDB、MySQL、Redis 等服务的指标。



二、Beats

1 介绍

轻量型数据采集器Beats是一个免费且开放的平台，集合了多种单一用途数据采集器。它们从成百上千或成千上万台机器和系统向 Logstash 或 Elasticsearch 发送数据。

> 课程中我们使用Beats系列最常用的Filebeat和Metricbeat采集日志文件和指标数据。

Beats 系列

全品类采集器，搞定所有数据类型。

Filebeat
日志文件
→

Metricbeat
指标
→

Packetbeat
网络数据
→

Winlogbeat
Windows 事件日志
→

Auditbeat
审计数据
→

Heartbeat
运行时间监控
→

Functionbeat
无需服务器的采集器
→

> 如果采集数据不需要任何处理，那么可以直接发送到Elasticsearch中。
>
> 如果采集的数据需要处理，那么可以发送到Logstash中，处理完成后再发送到Elasticsearch。
>
> 最后通过Kibana对数据进行一系列的可视化展示。

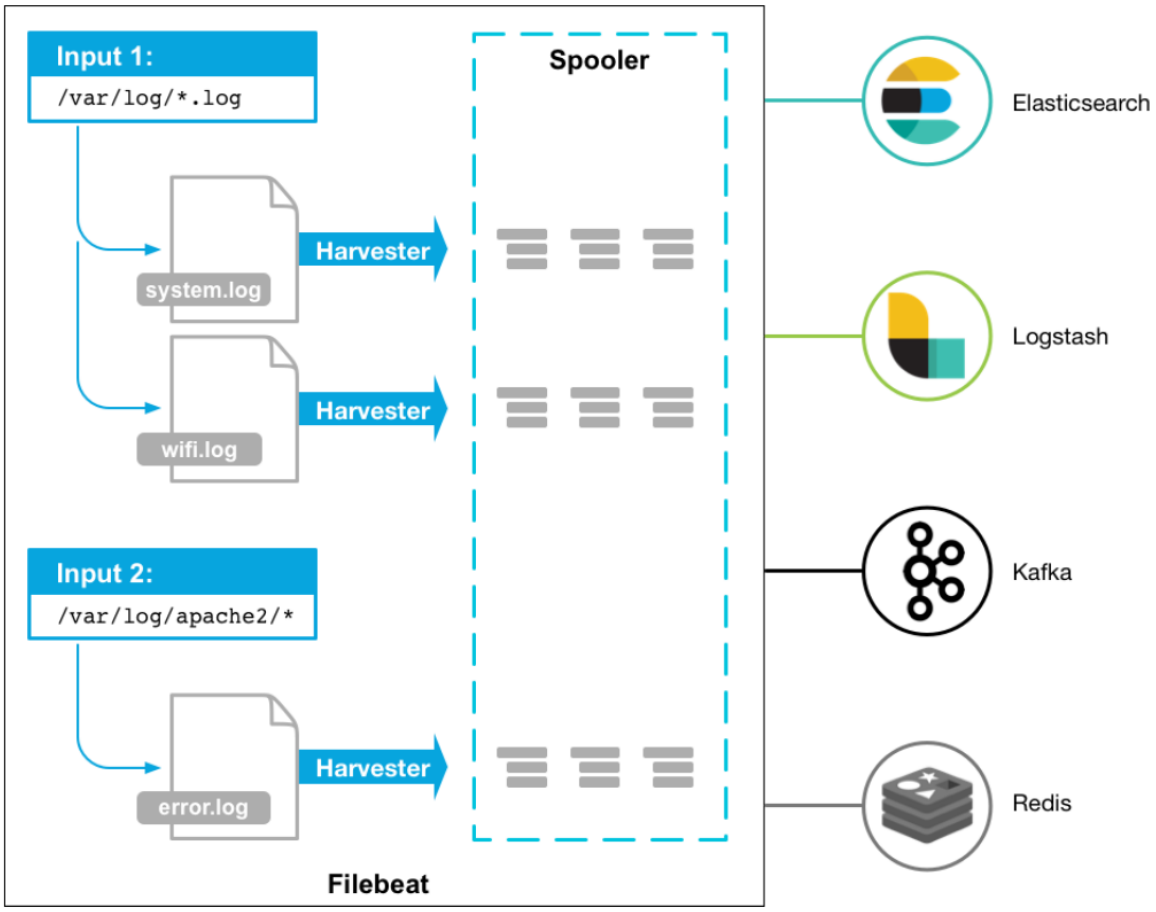
2 Filebeat

2.1 介绍

Filebeat是一款轻量型日志采集器，用于监控、收集服务器日志文件。

2.2 架构

> 首先Filebeat指定一些日志文件为数据输入源，之后使用Harvester（收割机）源源不断的读取日志，最后通过Spooler（卷轴）将日志数据传送到对应的目的地。



2.3 安装

1. 使用rz工具将Filebeat压缩文件上传到Linux虚拟机
2. 解压：

```
tar -zxvf filebeat-7.12.1-linux-x86_64.tar.gz  -C /usr/local/
```

2.4 入门案例

接下来我们使用filebeat读取一个普通的日志文件

1. 创建一个文本文件

```
vim /usr/local/mylog.log
# 为该文件随便添加一句话
```

2. 在filebeat中创建配置文件，配置文本文件的读取参数

```
# 进入filebeat文件夹下
cd /usr/local/filebeat-7.12.1-linux-x86_64/

# 创建配置文件
vim mylogconfig.yml

# 配置文件加入以下内容：
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /usr/local/mylog.log
output.console:
  pretty: true
  enable: true
```

3. 基于配置文件启动filebeat

```
./filebeat -e -c mylogconfig.yml
```

> 参数说明：
>
> -e：标准输出，输出到控制台
>
> -c：指定配置文件

4. 向文本文件追加数据，测试filebeat是否能为增量数据生成日志数据

```
# 打开另一个会话窗口，进入文本文件的目录下
cd /usr/local/

# 向文本文件中追加内容，再次查看filebeat的控制台
echo 'world' >> mylog.log
```

2.5 自定义字段

Filebeat读取日志文件后会生成json格式的日志，我们还可以为生成的日志添加一些自定义字段：

```
# 修改配置文件：
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /usr/local/mylog.log
  tags: ["mylog","test"]  #添加自定义标签，便于后续处理
  fields:  #添加自定义字段
    from: mylog
  fields_under_root: true #true为添加到根节点，false为添加到子节点中
output.console:
  pretty: true
  enable: true

# 查询filebeat原进程
ps -ef | grep filebeat
# 关闭filebeat原进程
kill -9 进程号
# 重启filebeat
./filebeat -e -c mylogconfig.yml

# 向文本文件追加数据
echo '123' >> mylog.log

# 我们可以看到生成的日志数据多了两个字段
```

2.6 收集Nginx日志

接下来我们使用filebeat收集Nginx日志

1. 安装Nginx

```
# 使用rz工具将Nginx压缩文件上传到Linux虚拟机
# 解压Nginx压缩文件
tar -zxvf nginx-1.21.1.tar.gz -C /usr/local/

# 安装依赖包
yum -y install gcc gcc-c++ pcre pcre-devel zlib zlib-devel openssl openssl-devel

# 进入Nginx解压路径
cd /usr/local/nginx-1.21.1

# 安装Nginx
./configure
make install

# 启动Nginx
/usr/local/nginx/sbin/nginx

# 访问Nginx
```

> Nginx的日志文件在/usr/local/nginx/logs中，正常日志存在access.log中，异常日志存在error.log中。

2. 读取Nginx日志的配置文件

```
# 在filebeat中创建配置文件
cd /usr/local/filebeat-7.12.1-linux-x86_64/
vim nginxlogconfig.yml

# 添加如下内容
filebeat.inputs:
- type: log
  enabled: true
  paths:
   - /usr/local/nginx/logs/*.log
  tags: ["nginx"]
output.console:
  pretty: true
  enable: true

# 启动filebeat，如果filebeat还在启动，关闭已启动的filebeat
./filebeat -e -c nginxlogconfig.yml
```

2.7 Filebeat模板

在收集Ngnix日志时，日志内容并没有处理，难以阅读其中的具体数据。Filebeat针对常见的服务提供了处理日志的模板。接下来我们讲解Filebeat中Module的使用。

1. 配置Nginx读取模板：

```
# 查看Filebeat的日志处理模板
./filebeat modules list

# 启用模板
./filebeat modules enable nginx

# 配置日志处理模板
cd modules.d/
vim nginx.yml

# 加入如下配置
- module: nginx
  # Access logs
  access:
    enabled: true
    var.paths: ["/usr/local/nginx/logs/access.log"]
  # Error logs
  error:
    enabled: true
    var.paths: ["/usr/local/nginx/logs/error.log"]
```

2. 修改配置文件：

```
vim nginxlogconfig.yml
# 修改为以下内容
filebeat.config.modules:
 path: ${path.config}/modules.d/*.yml
 reload.enabled: true
output.console:
  pretty: true
  enable: true
```

3. 启动filebeat，如果filebeat还在启动，关闭已启动的filebeat

```
./filebeat -e -c nginxlogconfig.yml
```

2.8 将数据输出到ES中

1. 启动Elasticsearch
2. 启动Kibana，连接Elasticsearch
3. 修改Filebeat配置文件：

```
vim nginxlogconfig.yml

# 修改为以下内容
filebeat.config.modules:
 path: ${path.config}/modules.d/*.yml
 reload.enabled: true
output.elasticsearch:
 hosts: ["127.0.0.1:9200"]
```

4. 启动filebeat，如果filebeat还在启动，关闭已启动的filebeat

```
./filebeat -e -c nginxlogconfig.yml
```

5. 进入Kibana查看数据

3 Metricbeat

3.1 介绍

Metricbeat是一款轻量型指标采集器，用于收集操作系统及应用服务的指标数据。

3.2 安装

1. 使用rz工具将Metricbeat压缩文件上传到Linux虚拟机
2. 解压：

```
tar -zxvf metricbeat-7.12.1-linux-x86_64.tar.gz  -C /usr/local/
```

3.3 采集系统指标

1. 修改Metricbeat默认配置文件

```
cd /usr/local/metricbeat-7.12.1-linux-x86_64/
vim metricbeat.yml
```

配置内容如下：

```
# 模板文件的位置
metricbeat.config.modules:
  path: ${path.config}/modules.d/*.yml
# 采集到的数据输出到ES的路径
output.elasticsearch:
  hosts: ["localhost:9200","localhsot:9201"]
```

2. 查看系统指标采集模板

```
# 进入模板文件目录
cd /usr/local/metricbeat-7.12.1-linux-x86_64/modules.d/

# 查看系统指标采集模板
cat system.yml
```

3. 开启Metricbeat，开始采集系统指标

```
cd /usr/local/metricbeat-7.12.1-linux-x86_64
./metricbeat -e
```

> -e 代表启动时使用默认配置文件

3.4 采集Nginx指标

1. nginx必须开启状态查询，才能查询到指标数据。

```
# 重新安装nginx
cd /usr/local/nginx-1.21.1/
./configure --prefix=/usr/local/nginx --with-http_stub_status_module
make
make install

# 配置nginx
cd /usr/local/nginx/conf
vim nginx.conf
# 在server内加入以下内容
location /status {
    stub_status on;
    access_log off;
}

# 重启nginx
cd /usr/local/nginx/sbin
./nginx -s stop
./nginx
```

2. 访问http://虚拟机IP/status，查看nginx指标数据：

```
Active connections: 1
server accepts handled requests
 1      1          3
Reading: 0 Writing: 1 Waiting: 0
```

- Active connections：目前活跃的连接数
- server：总共处理的连接数
- accepts ：成功创建的握手数

> server - accepts = 0，证明所有的连接均成功握手，没有失败连接。

- handled requests：总共处理的请求数
- Reading：Nginx读取到客户端的Header信息数
- Writing：Nginx返回给客户端Header信息数
- Waiting：Nginx已经等待请求的驻留链接

3. Metricbeat采集nginx指标

   配置nginx指标采集模板：

```
#启用ngnix模板
./metricbeat modules enable nginx

#修改ngnix模板配置
cd /usr/local/metricbeat-7.12.1-linux-x86_64/
vim modules.d/nginx.yml
```

加入如下配置：

```
- module: nginx
  period: 10s

  hosts: ["http://127.0.0.1"]
  server_status_path: "status"
```

重启Metricbeat：

```
./metricbeat -e
```

4. kibana查看采集到的nginx指标

```
GET /metricbeat索引/_search
{
  "query": {
    "term": {
      "service.type": "nginx"
    }
  }
}
```

三、Kibana

1 介绍

Kibana是一款开源的数据分析和可视化平台，它是Elastic Stack成员之一，设计用于和Elasticsearch协作。您可以使用Kibana对Elasticsearch索引中的数据进行搜索、查看、交互操作。您可以很方便的利用图表、表格及地图对数据进行多元化的分析和呈现。

2 安装

1. 使用rz工具将Kibana压缩文件上传到Linux虚拟机
2. 解压：

```
tar -zxvf kibana-7.12.1-linux-x86_64.tar.gz  -C /usr/local/
```

3. 修改配置

```
# 进入Kibana解压路径
cd /usr/local/kibana-7.12.1-linux-x86_64/config

# 修改配置文件
vim kibana.yml

# 加入以下内容
# kibana主机IP
server.host: "虚拟机IP"
# Elasticsearch路径
elasticsearch.hosts: ["http://127.0.0.1:9200","http://127.0.0.1:9201"]
```

4. 启动：

   kibana不能以root用户运行，我们给es用户设置kibana目录的权限，并使用es用户运行kibana

```
# 给es用户设置kibana目录权限
chown -R es:es /usr/local/kibana-7.12.1-linux-x86_64/

# 切换为es用户
su es

# 启动kibana
cd /usr/local/kibana-7.12.1-linux-x86_64/bin/
./kibana
```

3 Metricbeat仪表盘

kibana可以将Metricbeat收集的指标数据展示出来，并生成友好的图形化界面。接下来安装Metricbeat仪表盘。

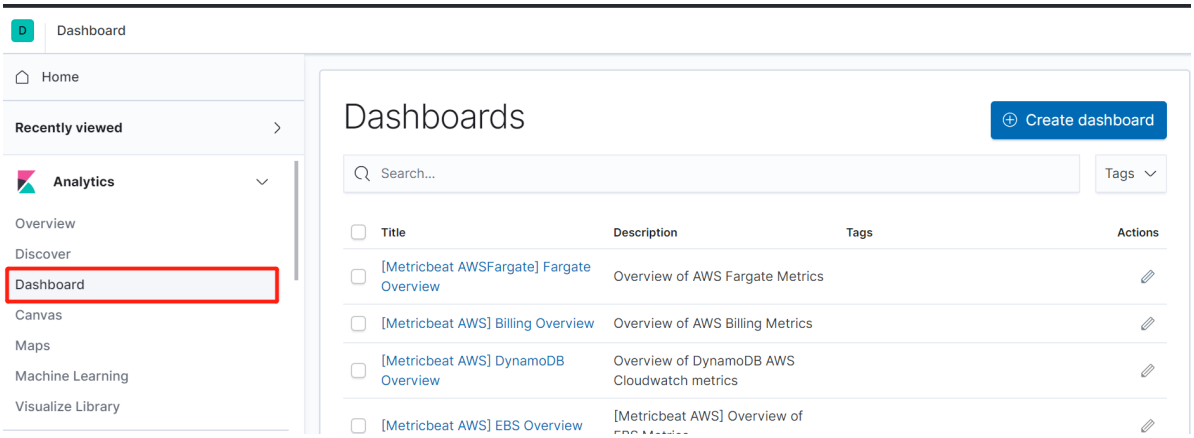
> 注：Metricbeat仪表盘是在Metricbeat中进行的

```
# 修改Metricbeat配置文件
cd /usr/local/metricbeat-7.12.1-linux-x86_64/
vim metricbeat.yml

# 加入如下内容
# 设置kibana路径
setup.kibana:
  host: "192.168.1.58:5601"
```

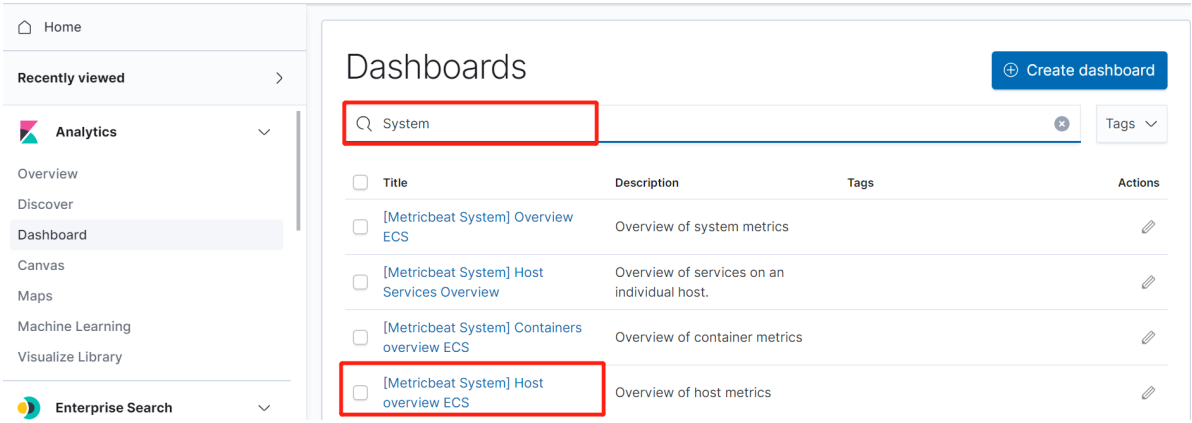
```
# 安装仪表盘（此时kibana必须处于运行状态）
./metricbeat setup --dashboards

# 启动Metricbeat
./metricbeat -e
```

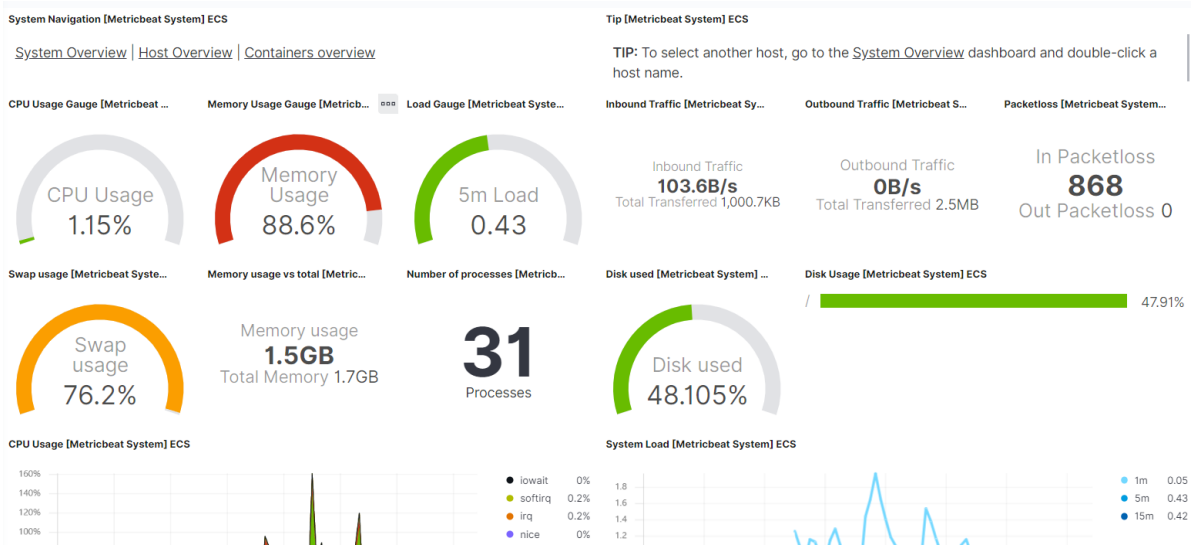
打开kibana，查看Metricbeat收集数据的仪表盘：

在菜单栏点击Dashboard

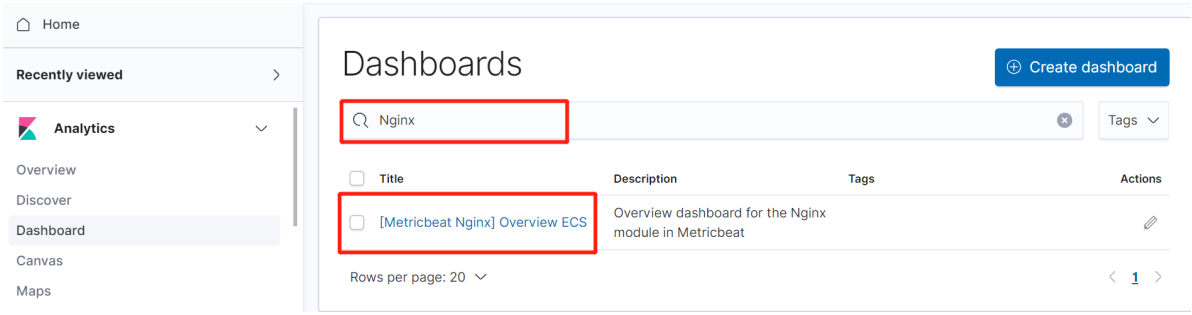


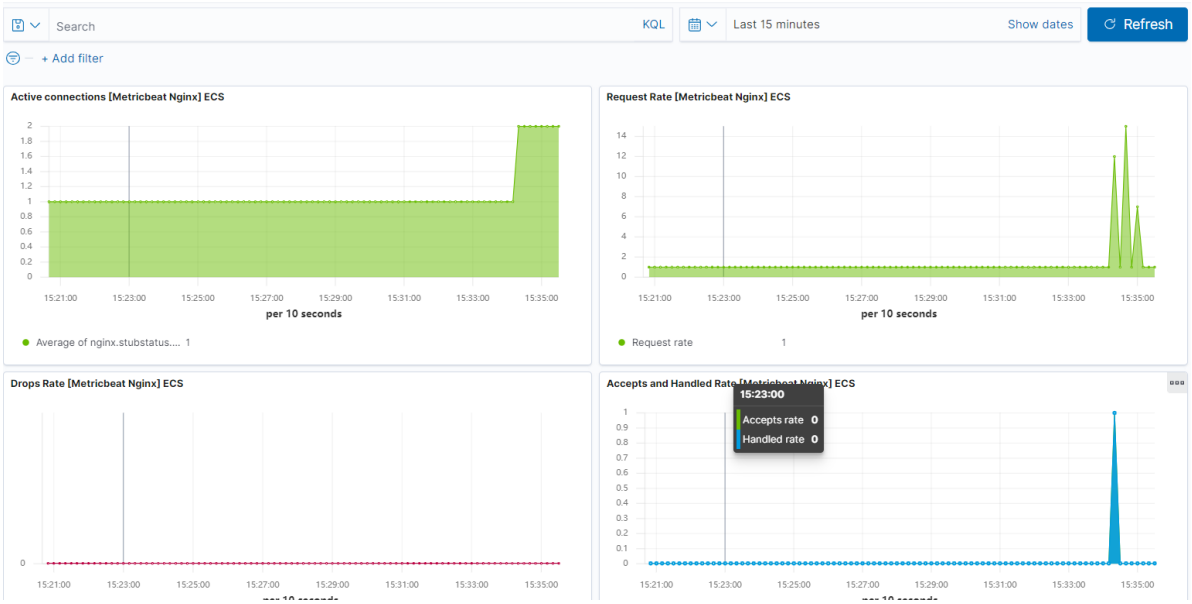
Metricbeat已经安装了很多仪表盘，搜索System，点击Host overview ESC，查看系统指标仪表盘。





搜索Nginx，点击Overview ECS，查看Ngnix指标仪表盘。



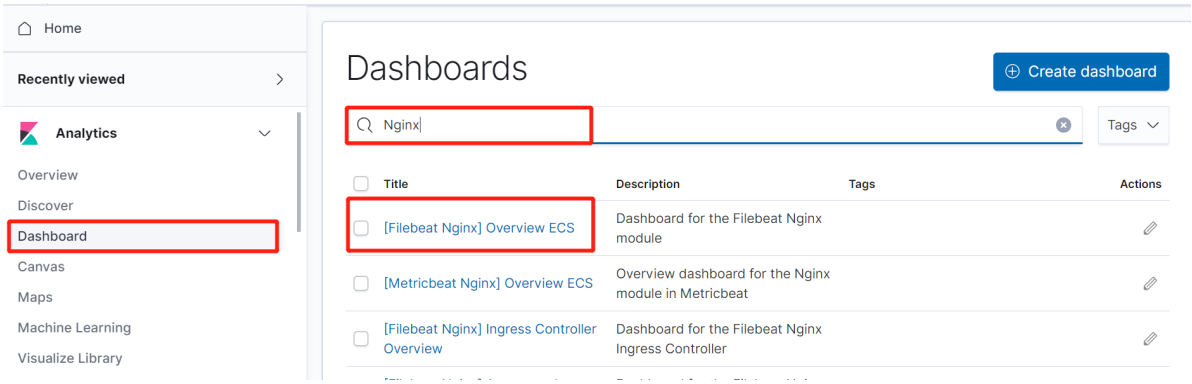


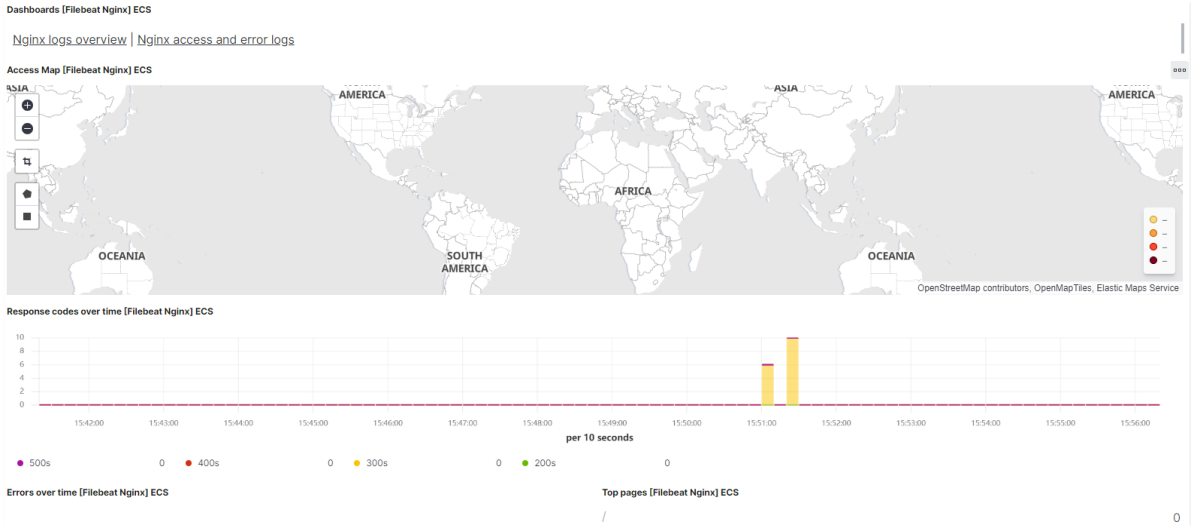
4 Filebeat仪表盘

kibana也可以将Filebeat收集的日志数据在仪表盘中展示出来，接下来我们在Filebeat中安装仪表盘。

```
# 修改Filebeat配置文件
cd /usr/local/filebeat-7.12.1-linux-x86_64/
vim nginxlogconfig.yml

# 加入如下内容
# 设置kibana路径
setup.kibana:
  host: "192.168.1.58:5601"

# 安装仪表盘
./filebeat -c nginxlogconfig.yml setup

# 启动Filebeat
./filebeat -e -c nginxlogconfig.yml
```

进入Kibana，点击Dashboard，搜索Nginx，点击[Filebeat Nginx] Overview ECS，查看Nginx日志仪表盘

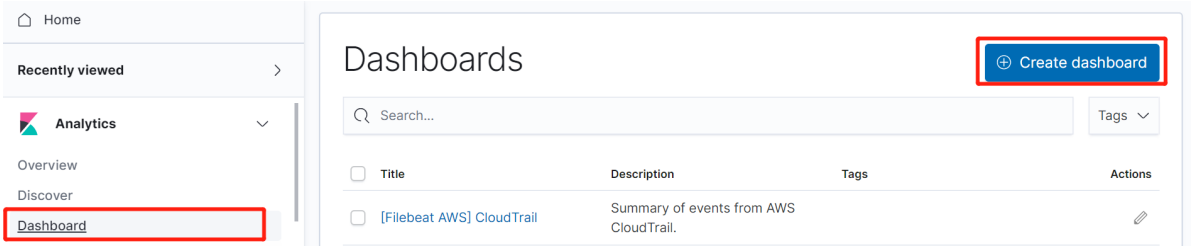




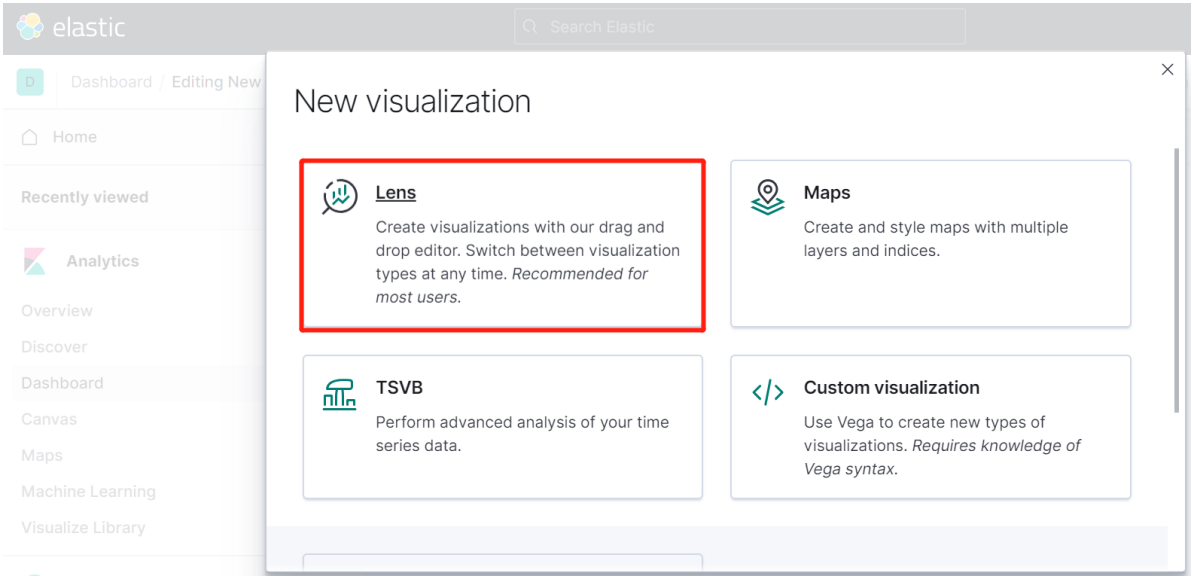
5 自定义仪表盘

在Kibana中，我们可以自定义仪表盘展示我们需要的数据，比如我们可以制作Ngnix每分钟访问量的仪表盘。

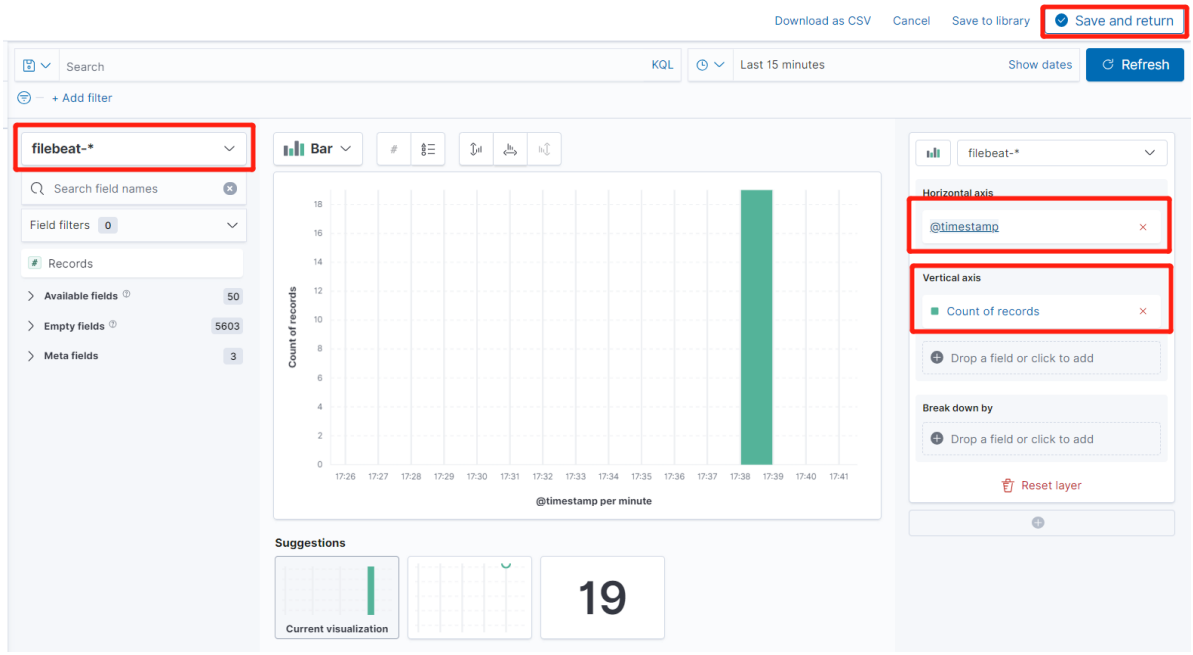
点击Dashborad --> Create dashboard



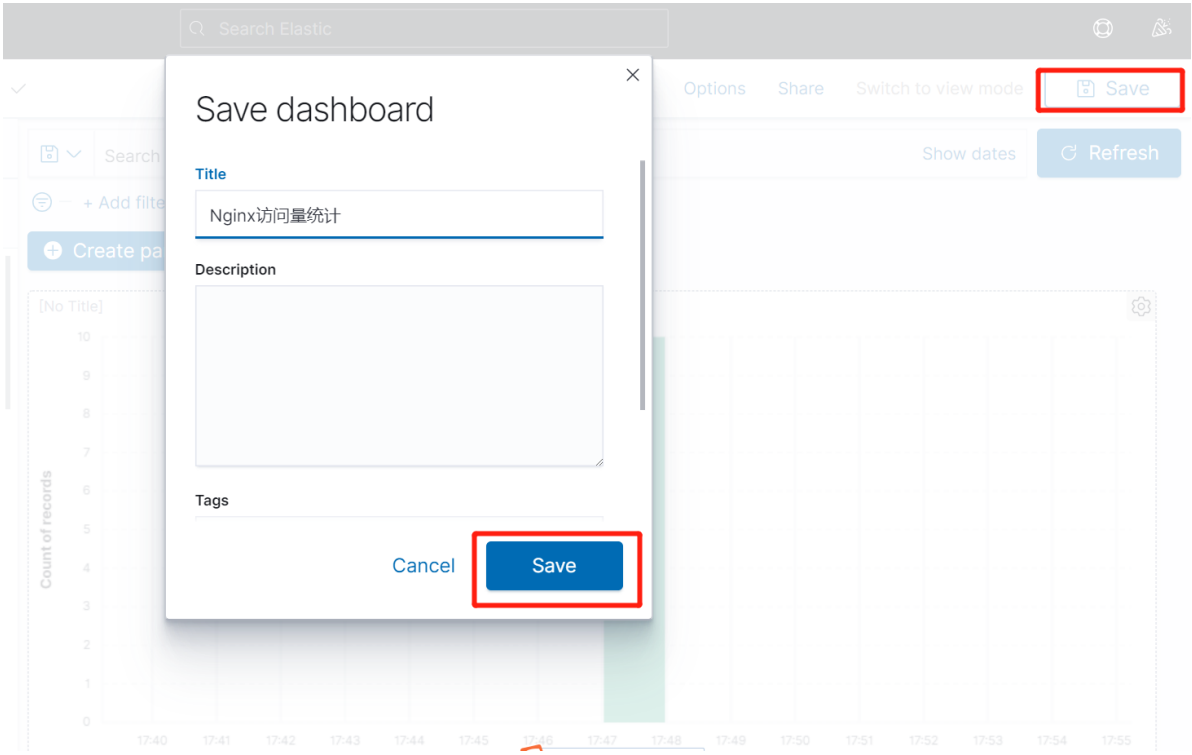
点击create panel --> Lens



选择对应的索引，横轴为@timestamp时间戳，纵轴为count次数，设置好点击Save and return



点击save，保存自定义Dashboard

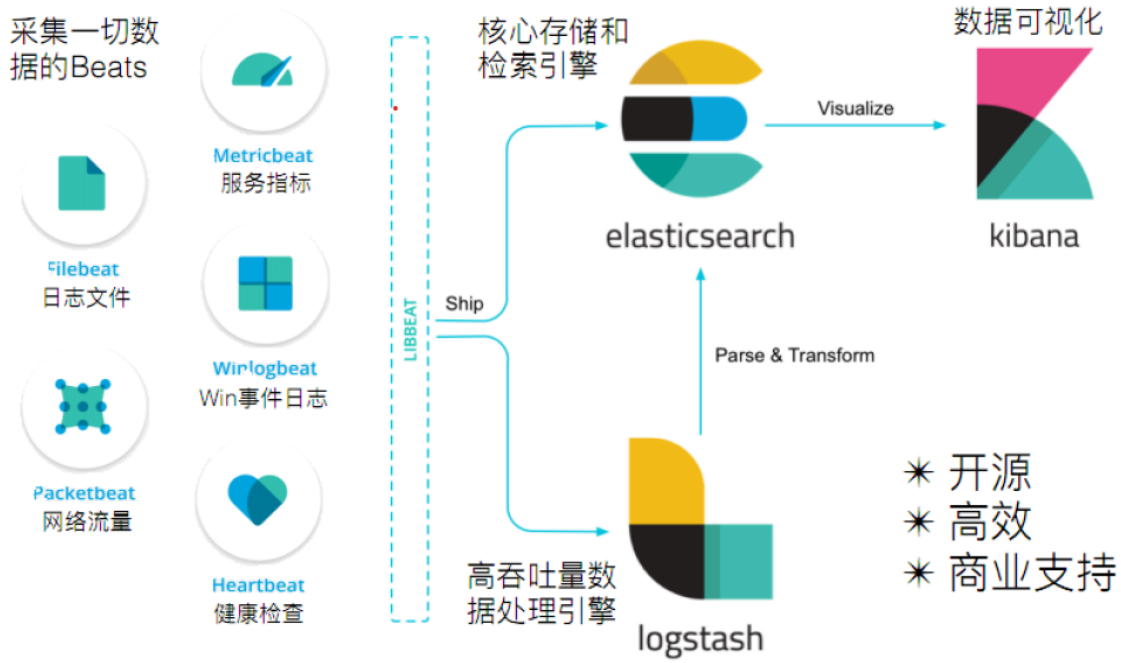


四、Logstash

1 介绍

Logstash是免费且开放的服务器端数据处理管道，能够从多个来源采集数据，转换数据，然后将数据发送到您最喜欢的"存储库"中。

早期的Logstash用于收集和处理数据。但由于Logstash在JVM中运行，运行时资源消耗比较大，现在更多使用Beats收集数据，Logstash处理Beats收集的数据。



2 安装

1. 使用rz工具将Logstash压缩文件上传到Linux虚拟机
2. 解压：

```
tar -zxvf logstash-7.12.1-linux-x86_64.tar.gz  -C /usr/local/
```

3 入门案例

接下来我们使用Logstash收集并处理一个日志文件

1. 创建日志文件

```
# 创建日志文件
cd /
vim mylog.log

# 加入如下日志
2021-08-17 16:21:21|ERROR|1001|查询产品异常|FindProduct
2021-08-17 16:22:21|OK|200|新增产品成功|AddProduct
2021-08-17 16:23:21|OK|200|新增产品成功|AddProduct
2021-08-17 16:24:21|OK|200|新增产品成功|AddProduct
2021-08-17 16:25:21|OK|200|新增产品成功|AddProduct
```

每条日志的内容由|分割，收集数据后也需要对数据做分割处理。

2. 编写配置文件

   Logstash配置文件有以下三部分构成：

```
# 输入源
input {}
# 处理方案
filter {}
# 输出目标
output {}
```

根据该结构编写配置文件：

```
# 编写配置文件
cd /usr/local/logstash-7.12.1/config/
vim mylog.conf

# 添加以下内容
input {
    file {
        path => "/mylog.log"
        start_position => "beginning"
    }
}
filter {
    mutate {
        split => {"message"=>"|"}
    }
}
output {
    elasticsearch {
        hosts => ["127.0.0.1:9200","127.0.0.1:9201"]
    }
}
```

3. 启动Logstash

```
# 启动Logstash
cd /usr/local/logstash-7.12.1/bin/
./logstash -f ../config/mylog.conf

# 追加数据
echo "2021-08-17 17:21:21|ERROR|1001|查询产品异常|FindProduct" >> /mylog.log
```

4 处理Beats收集的数据

由于JVM的启动，Logstash的收集速度比较慢，所以后面使用Beats来代替了Logstash进行收集，而Logstash负责处理Beats收集的数据。

1. 配置Logstash

```
cd /usr/local/logstash-7.12.1/config/
vim mylog.conf

# 设置为以下内容
input {
```

```
    beats {
        # 暴露给beats的端口
        port => "5044"
    }
}
filter {
    mutate {
        split => {"message"=>"|"}
    }
}
output {
    elasticsearch {
        hosts => ["127.0.0.1:9200"]
    }
}

# 启动Logstash
cd /usr/local/logstash-7.12.1/bin/
./logstash -f ../config/mylog.conf
```

2. 配置Filebeat

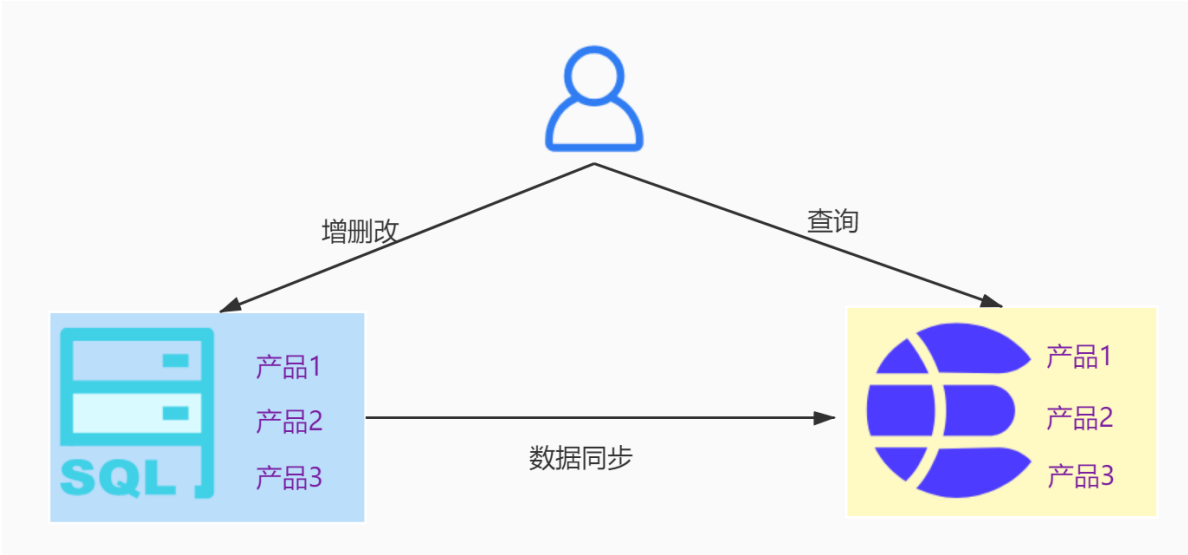
```
# 创建Filebeat配置文件
cd /usr/local/filebeat-7.12.1-linux-x86_64/
vim mylog.yml

# 添加如下配置
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /mylog.log
output.logstash:
  hosts: ["127.0.0.1:5044"]

# 启动Filebeat
./filebeat -e -c mylog.yml
```

3. 追加数据

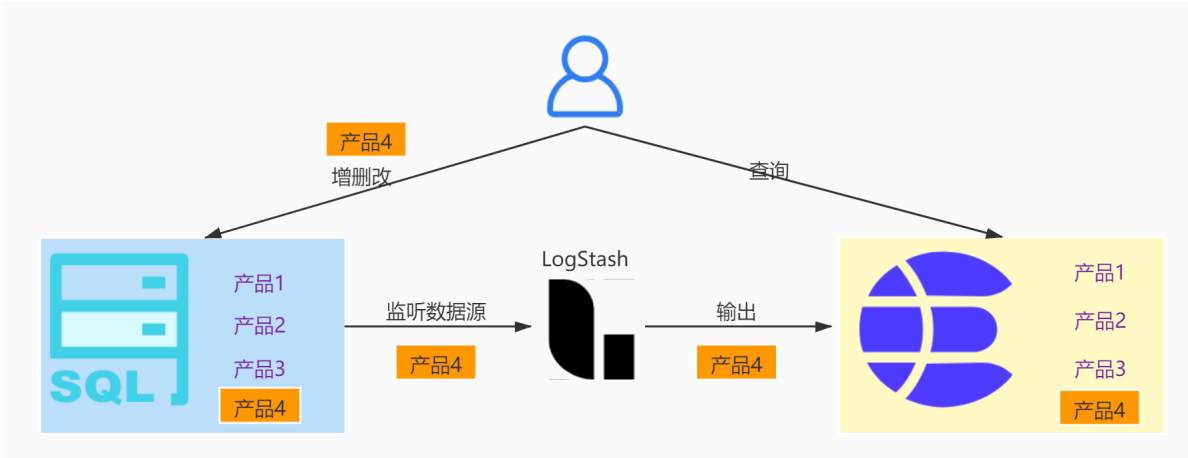
```
echo "2021-08-17 18:21:21|ERROR|1001|查询产品异常|FindProduct" >> /mylog.log
```

5 实现双写一致

在实际开发过程中，一些数据需要实现数据库与ES的双写一致。例如在电商网站中，增删改产品需要在数据库操作，而查询产品则需要在ES中操作，此时就需要在操作数据库后将数据同步到ES中。



Logstash可以将输入源设为数据库，输出源设为ES，定时监听数据库的数据变化，从而实现数据库和ES的双写一致。



1. 准备数据库

id	productName	price	updatetime	isDelete
1	iphone13	4999	2021-08-17 18:09:00	0
2	小米mix4	5399	2021-08-17 18:09:19	0
3	华为mate40	6999	2021-08-17 18:09:27	0
(Auto)	*(NULL)*	*(NULL)*	CURRENT_TIMESTAMP	*(NULL)*

> Logstash不支持删除同步。如果想实现删除同步，可以在设计数据库表时设置数据软删除，即添加一个字段表示该数据是否删除。删除时不进行物理删除，而是修改该字段的值。

Logstash实现增量导入需要有一个定位字段，通过该字段判断这个数据是否更新过。

案例中使用updateTime（修改时间）作为定位字段，logstash读取数据时会记录所有数据中最大的updateTime。下次读取数据会和上次最大的updateTime对比，如果大于上次最大的updateTime，证明该数据更新过，需要更新到ES中。

2. 在Elasticsearch中创建索引

```
PUT /product
{
    "mappings": {
        "properties": {
            "id": {
                "type": "integer",
```

```
                "store": true,
                "index": true
            },
            "productName": {
                "type": "text",
                "store": true,
                "index": true
            },
            "price": {
                "type": "double",
                "store": true,
                "index": true
            },
            "updatetime": {
                "type": "date",
                "store": true,
                "index": true
            },
            "isDelete": {
                "type": "integer",
                "store": true,
                "index": true
            }
        }
    }
}
```

3. 配置Logstash

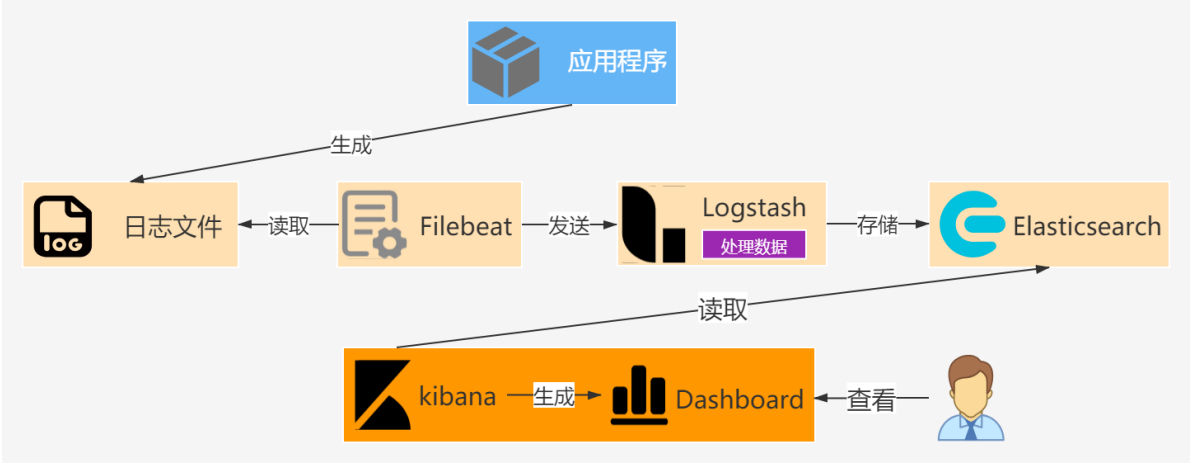
```
cd /usr/local/logstash-7.12.1/config/
vim mysql_product.conf

# 设置为以下内容
input {
    jdbc {
        jdbc_connection_string => "jdbc:mysql://192.168.1.12:3306/shopping"
        jdbc_user => "root"
        jdbc_password => "root"
        jdbc_driver_library => "/usr/local/logstash-7.12.1/lib/mysql-
connector-java-5.1.37-bin.jar"
        jdbc_driver_class => "com.mysql.jdbc.Driver"
        # 时区
        jdbc_default_timezone => "Asia/Shanghai"
        # SQL语句
        statement => "select * from product where updatetime >=
:sql_last_value;"
        # 执行SQL的周期，  [秒] 分钟 小时 天 月 年
        schedule => "* * * * *"

        # 是否使用字段的值作为比较策略
        use_column_value => true
        # 比较的字段名称
        tracking_column => "updatetime"
        # 比较的字段类型，numberic为数字，timestamp为日期
        tracking_column_type => "timestamp"
```

```
        # 记录比较字段值的文件，相对寻址路径是logstash的安装路径
        last_run_metadata_path => "./product-last-value"
    }
}
filter {
    # 解决采集数据的时差问题
    ruby {
        code => "event.set('updatetime',
event.get('updatetime').time.localtime + 8*60*60)"
    }
}
output {
    elasticsearch {
        hosts => ["127.0.0.1:9200","127.0.0.1:9201"]
        index => "product"
        document_id => "%{id}"
    }
}

# 启动Logstash
cd /usr/local/logstash-7.12.1/bin/
./logstash -f ../config/mysql_product.conf
```

五、Elastic Stack案例

1 案例介绍

接下来我们搭建一个应用程序的日志平台。对于应用程序来说，日志能够监控项目状态，排查异常，分析用户行为。以下是日志平台的流程介绍：

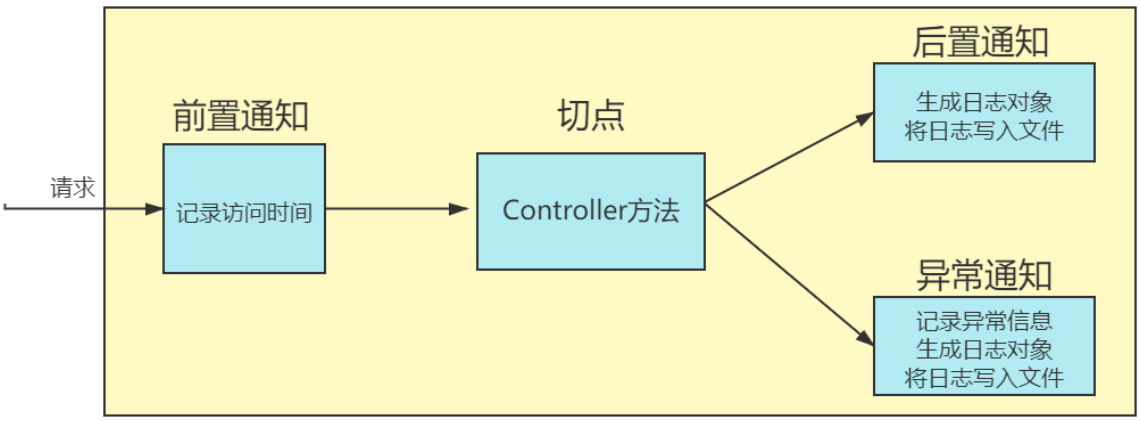


2 生成日志文件

我们在之前新闻搜索案例的基础上搭建项目：

日志生成的时机为：每次访问Controller接口都生成一条日志。

利用Spring的面向切面编程可以增强Controller的方法，使每次访问时都能创建一条日志对象写入文件。



1. 添加AOP起步依赖

```
<!-- aop起步依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

2. 准备日志实体类

```
public class Log {
    private Integer id; // id
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss",timezone = "GMT+8")
    private Date visitTime; // 访问时间
    private int executionTime; // 访问时长
    private String ip; // 访问者ip
    private String url; // 访问路径
    private String method; // 访问方法
    private int level; // 日志等级 0:正常 1:警告 2:异常
    private String message; // 异常信息
    private String stackTrace;//异常栈信息

    // 省略getter&setter
}
```

3. 准备日志AOP类:

```
@Component
@Aspect
public class LogAop {
    @Autowired
    private HttpServletRequest request;

    // 切点为Controller层所有方法
    @Pointcut("execution(* com.baizhan.estask.*.*Controller.*(..))")
    public void pointCut() {
    }

    @Before("pointCut()") // 前置通知
```

```
    public void before() {
        // 记录访问时间
        request.setAttribute("visitTime", new Date());
    }

    @AfterReturning("pointCut()") // 后置通知
    public void AfterReturning(JoinPoint joinPoint) {
        Date visitTime = (Date) request.getAttribute("visitTime"); // 访问时间
        long executionTime = new Date().getTime() - visitTime.getTime(); //
访问时长
        String ip = request.getRemoteAddr(); // 访问ip
        String url = request.getRequestURI();// 访问路径
        String method = joinPoint.getSignature().getName();//访问方法名（切点方
法）
        int level = 0; // 日志等级

        Log log = new Log();
        log.setId(UUID.randomUUID().toString());
        log.setMethod(method);
        log.setExecutionTime(executionTime);
        log.setUrl(url);
        log.setIp(ip);
        log.setLevel(level);
        log.setVisitTime(visitTime);
        printLog(log);

    }

    @AfterThrowing(pointcut = "pointCut()", throwing = "ex") // 异常通知
    public void afterThrowing(JoinPoint joinPoint, Throwable ex) {
        Date visitTime = (Date) request.getAttribute("visitTime"); // 访问时间
        Date now = new Date();
        int executionTime = (int) (now.getTime() - visitTime.getTime()); //
访问时长
        String ip = request.getRemoteAddr(); // 访问ip
        String url = request.getRequestURI();// 访问路径
        String method = joinPoint.getSignature().getName();//访问方法名（切点方
法）
        int level = 1;

        String message = ex.getMessage(); // 异常信息
        StringWriter sw = new StringWriter();
        PrintWriter printWriter = new PrintWriter(sw);
        ex.printStackTrace(printWriter);
        String stackTrace = sw.toString(); // 异常栈信息

        Log log = new Log();
        log.setId(UUID.randomUUID().toString());
        log.setMethod(method);
        log.setExecutionTime(executionTime);
        log.setUrl(url);
        log.setIp(ip);
        log.setLevel(level);
        log.setVisitTime(visitTime);
        log.setMessage(message);
        log.setStackTrace(stackTrace);
        printLog(log);
    }
```

```
    @Value("${logPath}")
    private String logPath; // 日志文件路径
    private ObjectMapper objectMapper = new ObjectMapper();
    // 写日志到文件
    public void printLog(Log log) {
        try {
            //改变输出方向,不输出到控制台,输出到指定文件
            System.setOut(new PrintStream(new
FileOutputStream(logPath,true)));
            //将日志对象转为JSON格式，写入文件
            System.out.println(objectMapper.writeValueAsString(log));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }
}
```

配置文件配置log文件路径：

```
spring:
 elasticsearch:
  rest:
   uris: http://192.168.1.24:9200
logPath: /usr/local/myAppLog.log
```

4. 使用maven将项目打成jar包
5. 使用rz命令将jar包上传至Linux虚拟机
6. 运行项目

```
java -jar estask-0.0.1-SNAPSHOT.jar
```

7. 测试项目，看是否将日志写入文件

3 创建Elasticsearch索引

创建es索引，用于存放日志数据。

```
PUT /myapplog
{
  "mappings": {
    "properties": {
      "id": {
        "type": "keyword",
        "store": true,
        "index": true
      },
      "visitTime": {
        "type": "keyword",
```

```
        "store": true,
        "index": true
      },
      "ip": {
        "type": "keyword",
        "store": true,
        "index": true
      },
      "url": {
        "type": "keyword",
        "store": true,
        "index": true
      },
      "executionTime": {
        "type": "long",
        "store": true,
        "index": true
      },
      "method": {
        "type": "keyword",
        "store": true,
        "index": true
      },
      "level": {
        "type": "integer",
        "store": true,
        "index": true
      },
      "message": {
        "type": "text",
        "store": true,
        "index": true
      },
      "stackTrace": {
        "type": "text",
        "store": true,
        "index": true
      }
    }
  }
}
```

4 配置Logstash

配置Logstash，用于处理Filebeat发送来的数据

```
cd /usr/local/logstash-7.12.1/config/
vim myapplog.conf

# 写入以下内容
input {
    beats {
        port => "5044"
    }
}
filter {
    # 分析json数据，存入对应的域中
```

```
    json {
        source => "message"
    }
    # 将host的json对象变为字符串，否则类型不匹配
    mutate {
        rename => { "[host][name]" => "host" }
    }
}
output {
    elasticsearch {
        hosts => ["127.0.0.1:9200","127.0.0.1:9201"]
        index => "myapplog"
        document_id => "%{id}"
    }
}

# 启动Logstash
cd /usr/local/logstash-7.12.1/bin/
./logstash -f ../config/myapplog.conf
```

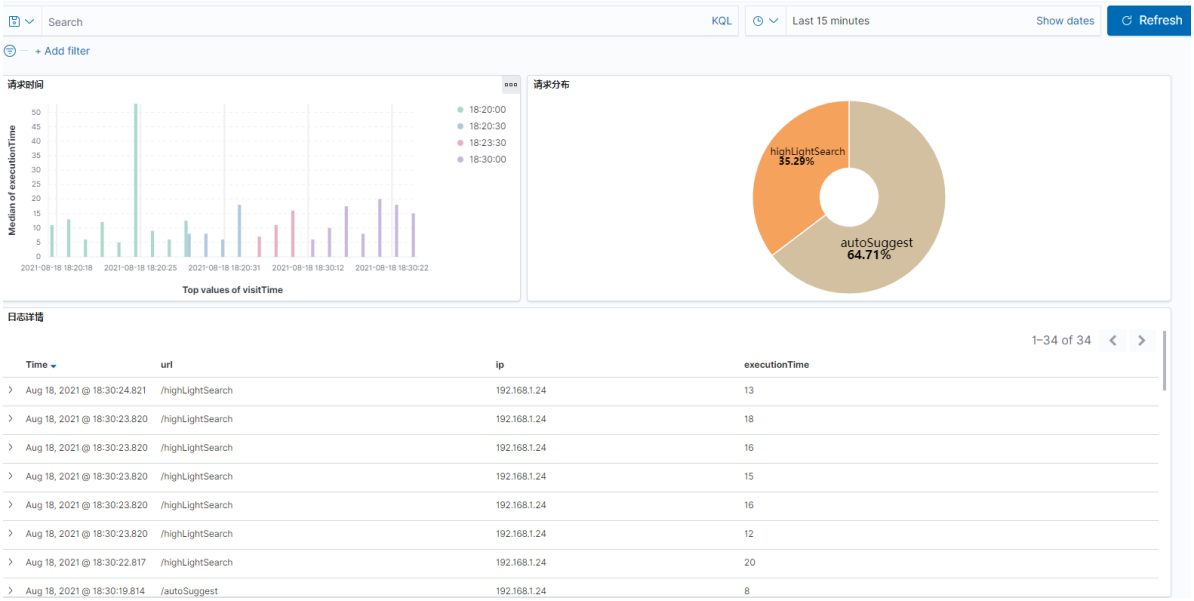
5 配置Filebeat

配置Filebeat，读取日志文件，发送给Logstash

```
# 创建Filebeat配置文件
cd /usr/local/filebeat-7.12.1-linux-x86_64/
vim myapplog.yml

# 添加如下配置
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /usr/local/news.log
output.logstash:
  hosts: ["127.0.0.1:5044"]

# 启动Filebeat
./filebeat -e -c myapplog.yml
```

6 制作仪表盘

接下来我们制作自定义仪表盘监控日志数据



日志详情

1–34 of 34

Time	url	ip	executionTime
Aug 18, 2021 @ 18:30:24.821	/highLightSearch	192.168.1.24	13
Aug 18, 2021 @ 18:30:23.820	/highLightSearch	192.168.1.24	18
Aug 18, 2021 @ 18:30:23.820	/highLightSearch	192.168.1.24	16
Aug 18, 2021 @ 18:30:23.820	/highLightSearch	192.168.1.24	15
Aug 18, 2021 @ 18:30:23.820	/highLightSearch	192.168.1.24	16
Aug 18, 2021 @ 18:30:23.820	/highLightSearch	192.168.1.24	12
Aug 18, 2021 @ 18:30:22.817	/highLightSearch	192.168.1.24	20
Aug 18, 2021 @ 18:30:19.814	/autoSuggest	192.168.1.24	8