ELK操作手册：

ELK+kafka安装

# 相关背景介绍

## ELK

背景

通常，日志被分散的储存不同的设备上。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：ELK中的logstash，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，这里我们使用ELK中的elasticsearch和kibana。

一、ELK是什么？

ELK实际上是三个工具的集合，Elasticsearch + Logstash + Kibana，这三个工具组合形成了一套实用、易用的监控架构，很多公司利用它来搭建可视化的海量日志分析平台。大家熟知的Sina、饿了么、携程、华为、美团、新浪微博、魅族、IBM...... 这些公司都在使用ELK。

Elasticsearch是实时全文搜索和分析引擎，提供搜集、分析、存储数据三大功能；是一套开放REST和JAVA API等结构提供高效搜索功能，可扩展的分布式系统。它构建于Apache Lucene搜索引擎库之上。

Logstash是一个用来搜集、分析、过滤日志的工具。它支持几乎任何类型的日志，包括系统日志、错误日志和自定义应用程序日志。它可以从许多来源接收日志，这些来源包括 syslog、消息传递（例如 RabbitMQ、kafka）和JMX，它能够以多种方式输出数据，包括电子邮件、websockets和Elasticsearch等。

Kibana是一个基于Web的图形界面，用于搜索、分析和可视化存储在 Elasticsearch指标中的日志数据。它利用Elasticsearch的REST接口来检索数据，不仅允许用户创建他们自己的数据的定制仪表板视图，还允许他们以特殊的方式查询和过滤数据。

### ELK有何优势？

1. 强大的搜索功能，elasticsearch可以以分布式搜索的方式快速检索，而且支持DSL的语法来进行搜索，简单的说，就是通过类似配置的语言，快速筛选数据。

2. 完美的展示功能，可以展示非常详细的图表信息，而且可以定制展示内容，将数据可视化发挥的淋漓尽致。

3. 分布式功能，能够解决大型集群运维工作很多问题，包括监控、预警、日志收集解析等。

## KAFKA

kafka详解：https://blog.csdn.net/ychenfeng/article/details/74980531

背景

通过查看网络上一些ELK的性能测试报告、博客和官方文档，发现logstash有性能瓶颈而且不支持缓存，目前业内典型替代方案是将 Redis 或 Kafka 作为中心缓冲池，这里我们选择kafka作为我们的日志收集缓冲池。

### KAFKA是什么？

Kafka是最初由Linkedin公司开发，是一个分布式、支持分区的（partition）、多副本的（replica），基于zookeeper协调的分布式消息系统，它的最大的特性就是可以实时的处理大量数据以满足各种需求场景：比如基于hadoop的批处理系统、低延迟的实时系统、storm/Spark流式处理引擎，web/nginx日志、访问日志，消息服务等等，用scala和java编写，Linkedin于2010年贡献给了Apache基金会并成为顶级开源 项目。

### KAFKA何优势？

1. 高吞吐量、低延迟：kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒。

2. 可扩展性：kafka集群支持热扩展。

3. 持久性、可靠性：消息被持久化到本地磁盘，并且支持数据备份防止数据丢失。

4. 容错性：允许集群中节点失败（若副本数量为n,则允许n-1个节点失败）。

5. 高并发：支持数千个客户端同时读写。

### KAFKA使用场景？

1. 日志收集：一个公司可以用Kafka可以收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer，例如hadoop、Hbase、Solr、logstash等。

2. 消息系统：解耦和生产者和消费者、缓存消息等。

3. 用户活动跟踪：Kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后订阅者通过订阅这些topic来做实时的监控分析，或者装载到hadoop、数据仓库中做离线分析和挖掘。

4. 运营指标：Kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告。

### KAFKA相关名词解释

Kafka中发布订阅的对象是topic。我们可以为每类数据创建一个topic，把向topic发布消息的客户端称作producer，从topic订阅消息的客户端称作consumer。Producers和consumers可以同时从多个topic读写数据。一个kafka集群由一个或多个broker服务器组成，它负责持久化和备份具体的kafka消息。

Broker：Kafka节点，一个Kafka节点就是一个broker，多个broker可以组成一个Kafka集群。

Topic：一类消息，消息存放的目录即主题，例如page view日志、click日志等都可以以topic的形式存在，Kafka集群能够同时负责多个topic的分发。

Partition：topic物理上的分组，一个topic可以分为多个partition，每个partition是一个有序的队列

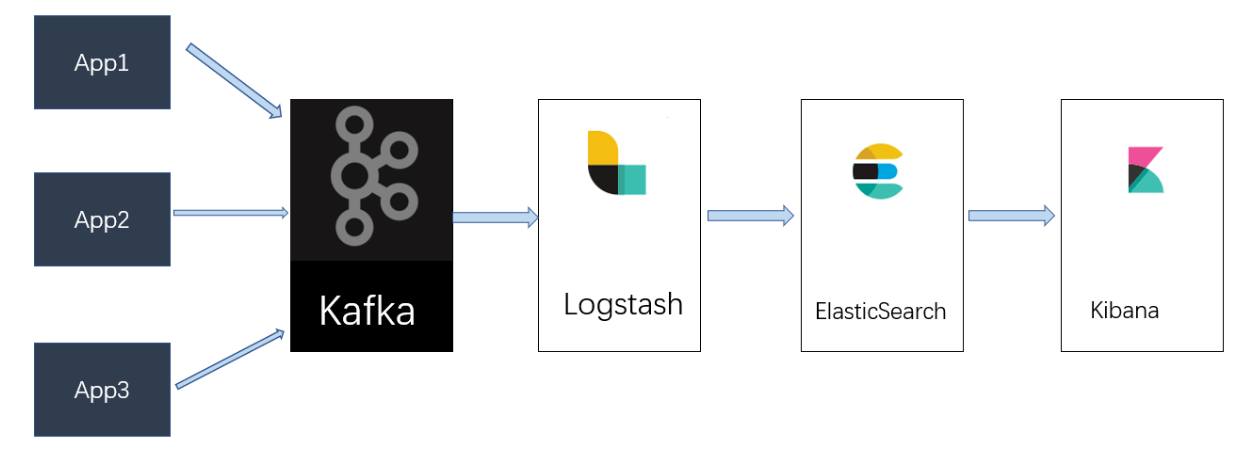
Segment：partition物理上由多个segment组成，每个Segment存着message信息

Producer: 生产message发送到topic

Consumer: 订阅topic消费message, consumer作为一个线程来消费

Consumer Group：一个Consumer Group包含多个consumer, 这个是预先在配置文件中配置好的。各个consumer（consumer 线程）可以组成一个组（Consumer group ），partition中的每个message只能被组（Consumer group ） 中的一个consumer（consumer 线程 ）消费，如果一个message可以被多个consumer（consumer 线程 ） 消费的话，那么这些consumer必须在不同的组。Kafka不支持一个partition中的message由两个或两个以上的consumer thread来处理，即便是来自不同的consumer group的也不行。

# 总体部署方案：(ip根据实际修改安装完成各服务后，别忘开启相应的端口)



# ELK安装部署+kafka集群

## 准备安装包

ELK下载地址：<https://www.elastic.co/downloads>

zookeeper下载地址：<https://www.apache.org/dyn/closer.cgi/zookeeper/>

kafka下载地址：<http://kafka.apache.org/downloads>

elasticsearch中文分词器下载地址： <https://pan.baidu.com/s/1i4ZAuwNueb_1-tZKBzQJ4g>

VMwareWorkstation

CentOS7

## 安装ES

  elasticsearch官方文档：<https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

### 创建用户

因为elasticsearch不能使用root用户启动，所以需要先创建一个用户，并对用户进行授权，使用root用户登录，执行以下命令

useradd es         #创建用户

passwd es          #设置用户密码，输入与用户名一样的密码

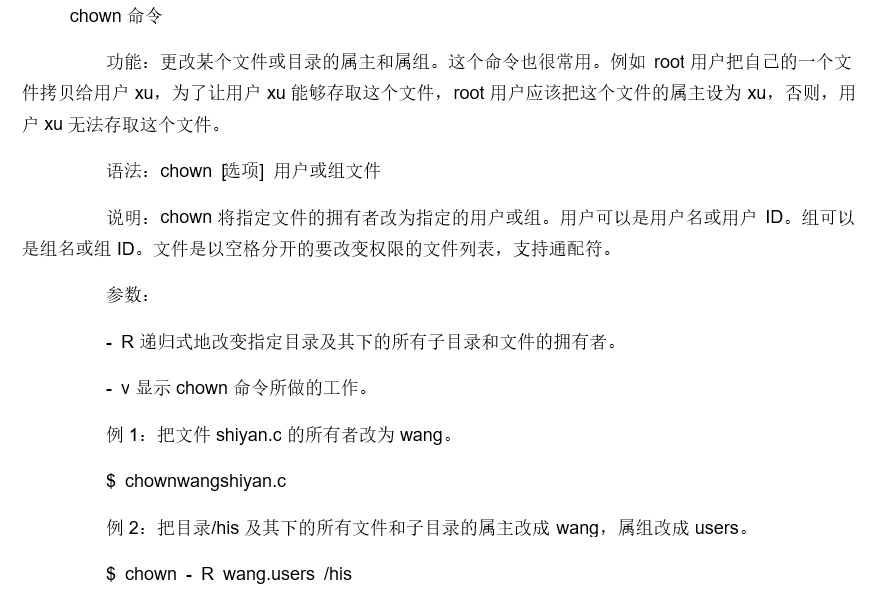
linux切换用户方法： su username

如：su es

su root

给用户授权：

chown 意思是->change own



### 修改系统参数

vi /etc/security/limits.conf

增加以下内容

\* soft nofile 65536

\* hard nofile 131072

\* soft nproc 2048

\* hard nproc 4096

vi /etc/security/limits.d/90-nproc.conf

修改如下内容：

\* soft nproc 1024

#修改为

\* soft nproc 4096

vi /etc/sysctl.conf

添加下面配置：

vm.max\_map\_count=655360

并执行命令：

sysctl -p

### 上传文件

  使用elasticsearch用户登录，将之前下载好的elasticsearch-6.2.4.tar.gz、elasticsearch-analysis-ik-6.2.4.zip上传至根目录

### 创建数据与日志目录

  在elasticsearch用户的根目录执行以下命令

mkdir {log,data}

### 解压安装包、修改配置

  解压安装包

tar -zxvf elasticsearch-6.2.4.tar.gz

进入elasticsearch-6.2.4/config目录，config目录下有三个文件：

elasticsearch.yml                            # els的配置文件

jvm.options                                  # JVM相关的配置，内存大小等等

log4j2.properties                            # 日志系统定义

 修改后的配置elasticsearch.yml如下：

cluster.name: es-cluster                                                                       #集群名称

node.name: node-110                                                                            #节点名称，仅仅是描述名称，用于在日志中区分

path.data: /home/es/data                                                                       #数据的默认存放路径

path.logs: /home/es/log                                                                        #日志默认存放路径

network.host: 192.168.68.110                                                                   #当前节点的IP地址

http.port: 9200                                                                                #对外提供服务的端口，9300为集群服务的端口

discovery.zen.ping.unicast.hosts: ["192.168.68.110", "192.168.68.111","192.168.68.112"]        #集群个节点IP地址

discovery.zen.minimum\_master\_nodes: 2                                                          #为了避免脑裂，集群节点数最少为 半数+1

bootstrap.system\_call\_filter: false                                                            #禁止系统调用过滤器

 jvm.options中默认内存为1G，开发环境可以不修改，生产时需要根据服务器内存调整大小

-Xms1g          # JVM最大、最小使用内存

-Xmx1g

### 加入中文分词器插件

日志中可能出现中文，为了方便中文检查，需要加入中文分词器插件

unzip elasticsearch-analysis-ik-6.2.4.zip -d elasticsearch-6.2.4/plugins/

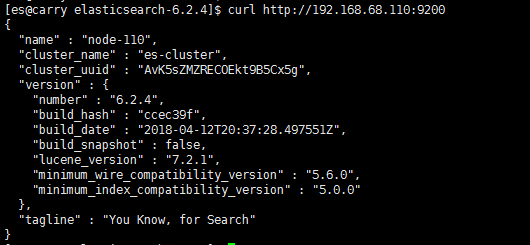
mv elasticsearch/ik                      #因为ik解压后的目录名为elasticsearch，为了避免其他插件可能冲突，修改名称为ik

### 启动easticsearch（一定不要在root下启动，会报错: can not run elasticsearch as root）

./elasticsearch -d

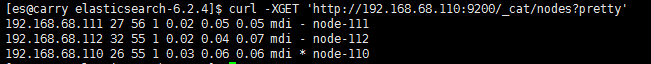
### 访问测试

使用curl访问：curl [http://192.168.68.110:9200](http://192.168.11.81:9200/)，出现如下信息，说明访问成功



按上述步骤将三台集群安装好后，检查集群状态

curl -XGET 'http://192.168.68.110:9200/\_cat/nodes?pretty'



带“\*”的为主节点

### 安装时遇到的问题

**问题一： [1]: max file descriptors [4096] for elasticsearch process is too low, increase to at least [65536])**  
**解决：**切换到root用户，进入/etc/security目录下修改配置文件   
vi /etc/security/limits.conf  
**增加以下内容**  
\* soft nofile 65536  
\* hard nofile 131072  
\* soft nproc 2048  
\* hard nproc 4096

**注：修改以上配置后需要重启服务器**

**问题二： max number of threads [2048] for user [es] is too low, increase to at least [4096]**

**解决：**切换到root用户，进入limits.d目录下修改配置文件。

vi /etc/security/limits.d/90-nproc.conf   
**修改如下内容：**  
\* soft nproc 1024  
**#修改为**  
\* soft nproc 4096  
**注：修改以上配置后需要重启服务器**

**问题三：max virtual memory areas vm.max\_map\_count [65530] likely too low, increase to at least [262144]**  
**解决：**切换到root用户修改配置sysctl.conf  
vi /etc/sysctl.conf   
**添加下面配置：**  
**vm.max\_map\_count=655360**  
**并执行命令：**  
sysctl -p

## kafka集群搭建(普通安装方法)

### 创建用户

   使用root用户登录，执行以下命令

useradd kafka                  #创建用户

passwd kafka                   #设置用户密码，输入与用户名一样的密码

### 解压安装包

  使用新创建的用户将包上传至用户根目录，并解压

tar -zxvf kafka\_2.11-1.1.0.tgz

创建目录

mkdir zookeeper

cd zookeeper

mkdir {logs,data}

### 修改zookeeper配置（kafka自带zookeeper）

进入配置目录：cd kafka\_2.11-1.1.0/config/

修改配置文件：vi zookeeper.properties

修改后的配置如下(三台机的zookeeper.properties内容一致)：

#zookeeper数据存放目录

dataDir=/home/kafka/zookeeper/data

#zookeeper日志存放目录

dataLogDir=/home/kafka/zookeeper/logs

# the port at which the clients will connect

clientPort=2181

# disable the per-ip limit on the number of connections since this is a non-production config

# 客户端连接的最大数量.

maxClientCnxns=60

# 心跳间隔时间，zookeeper中使用的基本时间单位，毫秒值。每隔2秒发送一个心跳

tickTime=2000

# leader与follower连接超时时间。表示10个心跳间隔

initLimit=10

# Leader与follower之间的超时时间，表示5个心跳间隔

syncLimit=5

server.1=192.168.68.110:2888:3888

server.2=192.168.68.111:2888:3888

server.3=192.168.68.112:2888:3888

创建myid文件，进入cd /home/kafka/zookeeper/data

192.168.68.110     echo 1 > myid

192.168.68.111     echo 2 > myid

192.168.68.112     echo 3 > myid

注：myid是zk集群用来发现彼此的标识，必须创建，且不能相同

### 修改kakfa配置

进入配置目录：cd kafka\_2.11-1.1.0/config/

修改配置文件：vi server.properties

broker.id=0                                #分别配置为192.168.68.110=0，192.168.68.111=1，192.168.68.112=2

listeners=PLAINTEXT://192.168.68.110:9092  #监听地址，分别配置不同的服务器PLAINTEXT://192.168.68.111:9092，PLAINTEXT://192.168.68.112:9092

log.dirs=/home/kafka/kafka\_2.11-1.1.0/logs

zookeeper.connect=192.168.68.110:2181,192.168.68.111:2181,192.168.68.112:2181

### 启动zookeeper

./zookeeper-server-start.sh  ../config/zookeeper.properties

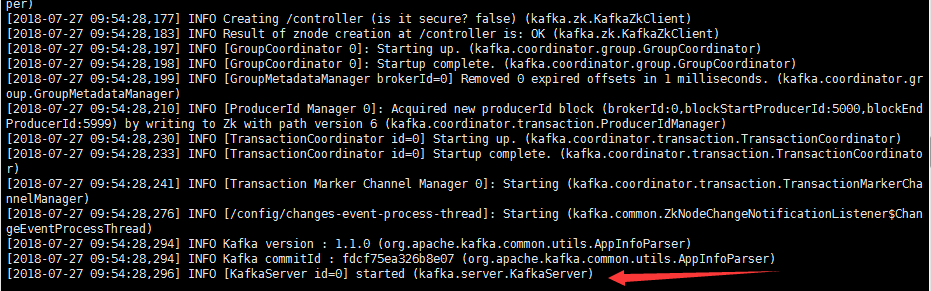
查看日志文件/home/kafka/kafka\_2.11-1.1.0/logs/server.log，如无异常信息说明zookeeper集群成功

### 启动kafka

**注：启动kafka之前需要将kafka、zookeeper所在服务器的防火墙关闭**

./kafka-server-start.sh ../config/server.properties

出现下图中日志则说明启动成功了



## kafka集群搭建(docker安装方法)

### 启动zookeeper容器

docker run -d --name zookeeper -p 2181:2181 -t wurstmeister/zookeeper

### 启动kafka容器

docker run -d --name kafka \

-p 9092:9092 \

-e KAFKA\_BROKER\_ID=0 \

-e KAFKA\_ZOOKEEPER\_CONNECT=192.168.204.128:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://192.168.204.128:9092 \

-e KAFKA\_LISTENERS=PLAINTEXT://0.0.0.0:9092 -t wurstmeister/kafka

这里面主要设置了4个参数

KAFKA\_BROKER\_ID=0

KAFKA\_ZOOKEEPER\_CONNECT=192.168.204.128:2181

KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://192.168.204.128:9092

KAFKA\_LISTENERS=PLAINTEXT://0.0.0.0:9092

中间两个参数的192.168.204.128改为宿主机器的IP地址，如果不这么设置，可能会导致在别的机器上访问不到kafka。

### 测试kafka

进入kafka容器的命令行

docker exec -ti kafka /bin/bash

* 1

进入kafka所在目录

cd opt/kafka\_2.12-1.1.0/

* 1

后面不再赘述，可参考前一篇文章[https://blog.csdn.net/lblblblblzdx/article/details/80548157](https://blog.csdn.net/lblblblblzdx/article/details/80548157" \t "_blank)

### 集群搭建

使用docker命令可快速在同一台机器搭建多个kafka，只需要改变brokerId和端口

docker run -d --name kafka1 \

-p 9093:9093 \

-e KAFKA\_BROKER\_ID=1 \

-e KAFKA\_ZOOKEEPER\_CONNECT=192.168.204.128:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://192.168.204.128:9093 \

-e KAFKA\_LISTENERS=PLAINTEXT://0.0.0.0:9093 -t wurstmeister/kafka

1

在kafka容器中的opt/kafka\_2.12-1.1.0/目录下输入

*bin/kafka*-*topics*.*sh* --*create* --*zookeeper* *192*.*168*.*204*.*128:2181* --*replication*-*factor* *2* --*partitions* *2* --*topic* *applog*

### 查看topic的状态

在kafka容器中的opt/kafka\_2.12-1.1.0/目录下输入

*bin/kafka*-*topics*.*sh* --*describe* --*zookeeper* *192*.*168*.*204*.*128:2181* --*topic* *applog*

输出结果：

Topic:partopic PartitionCount:2 ReplicationFactor:2 Configs:

Topic: partopic Partition: 0 Leader: 0 Replicas: 0,1 Isr: 0,1

Topic: partopic Partition: 1 Leader: 0 Replicas: 1,0 Isr: 0,1

显示每个分区的Leader机器为broker0，在broker0和1上具有备份，Isr代表存活的备份机器中存活的。   
当停掉kafka1后，

docker stop kafka1

1

再查看topic状态，输出结果：

Topic:partopic PartitionCount:2 ReplicationFactor:2 Configs:

Topic: partopic Partition: 0 Leader: 0 Replicas: 0,1 Isr: 0

Topic: partopic Partition: 1 Leader: 0 Replicas: 1,0 Isr: 0

## 安装logstash（192.168.68.111）

logstash官方文档：<https://www.elastic.co/guide/en/logstash/current/index.html>

### 解压安装包、修改配置

解压安装包：tar -zxvf logstash-6.2.4.tar.gz

进入到解压后的目录：cd logstash-6.2.4

进入到config文件夹：cd config

由于logstash的配置文件在启动的时候指定，编辑创建配置文件：vi kafkaInput\_esOutPut.conf

修改后的配置如下：

input {

    kafka {

        bootstrap\_servers => "192.168.68.110:9092, 192.168.68.111:9092,192.168.68.112:9092"

        topics => ["test"]

        auto\_offset\_reset => "latest"

        consumer\_threads => 5

        decorate\_events => true

     }

}

output {

    elasticsearch {

        hosts => ["http://192.168.68.110:9200"]

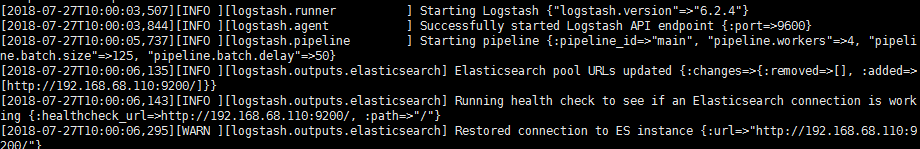
        index => "kafka-logs-%{+YYYY-MM-dd}"

    }

}

### 启动logstash

启动kafka：./bin/logstash -f config/kafkaInput\_esOutPut.conf



## 安装kibana（192.168.68.1112）

 kibana官方配置指南：<https://www.elastic.co/guide/en/kibana/current/settings.html>

### 解压安装包、修改配置

解压安装包：tar -zxvf kibana-6.2.4-linux-x86\_64.tar.gz

进入到解压后的目录：cd kibana-6.2.4-linux-x86\_64

进入到config文件夹：cd config

修改配置文件：vi kibana.yml

修改后的配置如下：

server.port: 5601

server.host: "192.168.68.112"

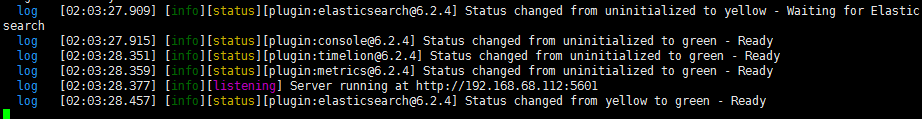
server.name: "kibana"

elasticsearch.url: "http://192.168.68.110:9200"

### 启动kibana

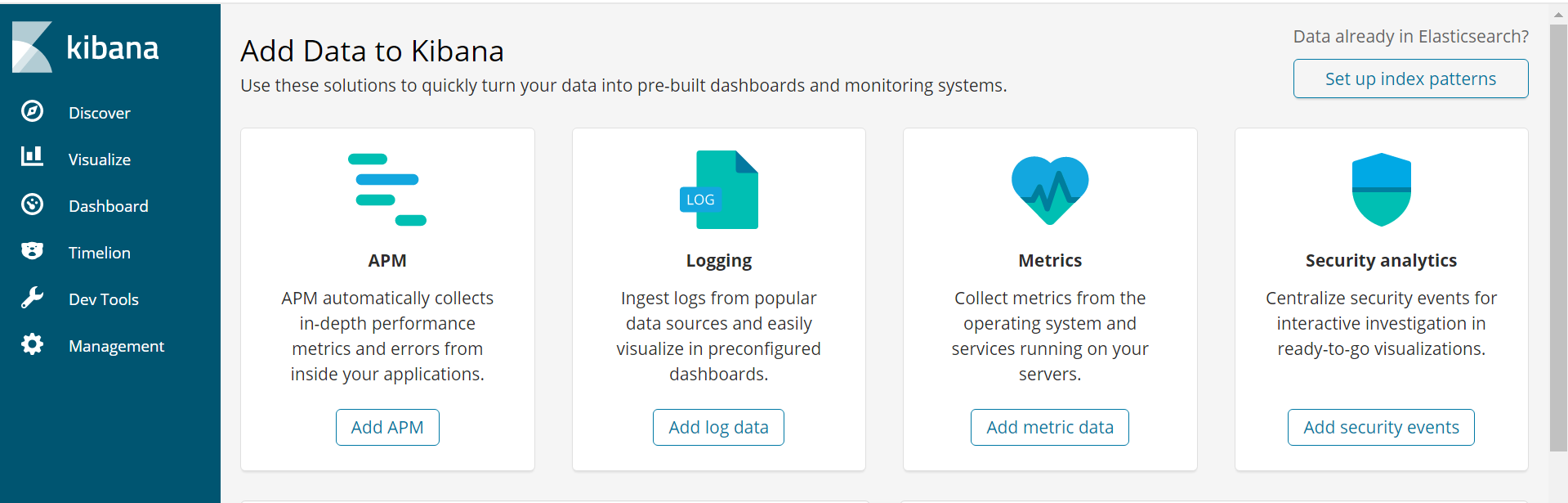
启动：./kibana

出现下图中日志则说明启动成功

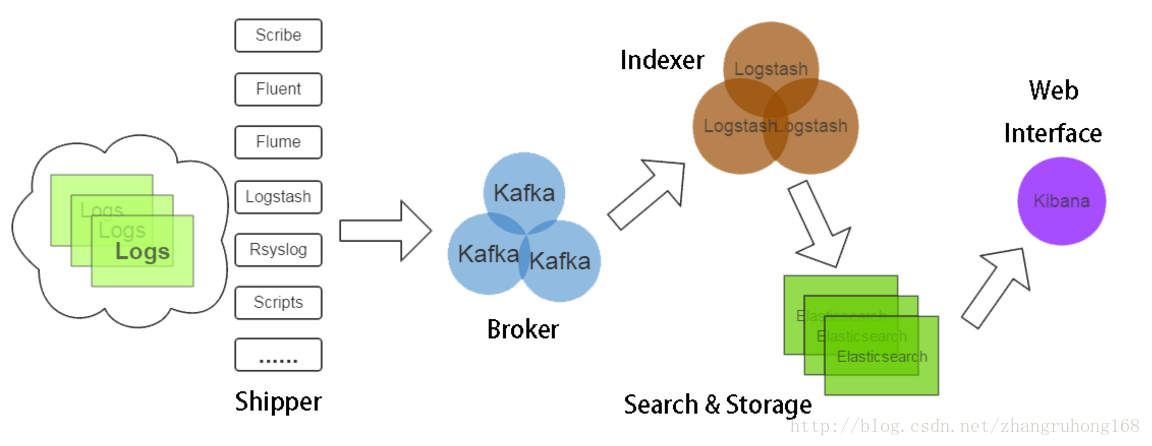


### 访问测试

在浏览器中输入：[http://192.168.68.112:5601](http://192.168.11.165:5601/)，出现如下界面，说明部署成功



# Spring Cloud Sleuth通过Kafka将链路追踪日志输出到ELK



## 工程简介

eureka-server作为其他三个项目的服务注册中心

kafka-service1调用kafka-service2， service2再调用kafka-service3

## 新建springboot项目eureka-server

### pom.xml内容如下

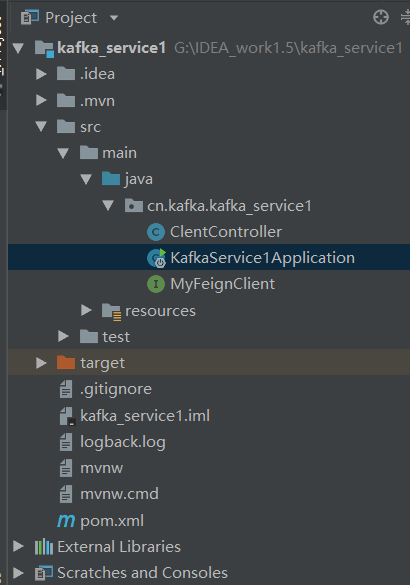
<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.1.3.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.example</groupId>  
 <artifactId>demo</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>war</packaging>  
 <name>demo</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Greenwich.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.yaml</groupId>  
 <artifactId>snakeyaml</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-tomcat</artifactId>  
 <scope>provided</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
 <repositories>  
 <repository>  
 <id>spring-milestones</id>  
 <name>Spring Milestones</name>  
 <url>https://repo.spring.io/milestone</url>  
 </repository>  
 </repositories>  
  
</project>

### application.yml内容如下

**server:  
 port:** 8080  
**spring:  
 application:  
 name:** eureka-server  
**eureka:  
 instance:  
 hostname:** localhost  
 **client:  
 register-with-eureka: false  
 fetch-registry: false  
 server:  
 enable-self-preservation: false** *#serviceUrl:  
 #defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/*

在启动类ElkEurekaServerApplication上加上注解@EnableEurekaServer

## 新建springboot项目kafka-service1



### pom.xml内容如下

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.1.3.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>cn.kafka</groupId>  
 <artifactId>kafka\_service1</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>kafka\_service1</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Greenwich.SR1</spring-cloud.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>  
 <!--服务调用-->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
 </dependency>  
 <!--链路追踪-->  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-sleuth</artifactId>  
 </dependency>  
 <!--负责收集日志-->  
 <dependency>  
 <groupId>net.logstash.logback</groupId>  
 <artifactId>logstash-logback-encoder</artifactId>  
 <version>4.11</version>  
 </dependency>  
 <!--消息转发-->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>com.github.danielwegener</groupId>  
 <artifactId>logback-kafka-appender</artifactId>  
 <version>0.1.0</version>  
 <scope>runtime</scope>  
 </dependency>  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
 <repositories>  
 <repository>  
 <id>spring-milestones</id>  
 <name>Spring Milestones</name>  
 <url>https://repo.spring.io/milestone</url>  
 </repository>  
 </repositories>  
  
</project>

### application.yml内容如下

**server:  
 port:** 9020  
**spring:  
 application:  
 name:** KAFKA-CLIENT1  
**eureka:  
 client:  
 serviceUrl:  
 defaultZone:** http://localhost:8080/eureka/  
**ribbon:  
 eureka:  
 enabled: true**

### 在resources目录下新增logback.xml文件，内容如下

<?xml version="1.0" encoding="UTF-8"?>  
<configuration debug="false" scan="true" scanPeriod="1 seconds">  
 <include resource="org/springframework/boot/logging/logback/base.xml" />  
 <!-- <jmxConfigurator/> -->  
 <contextName>logback</contextName>  
 <!-- 应该是日志存储的地址 -->  
 <property name="log.path" value="logback.log" />  
 <!-- 日志的格式 -->  
 <property name="log.pattern" value="%d{yyyy-MM-dd HH:mm:ss.SSS} -%5p ${PID} --- traceId:[%X{mdc\_trace\_id}] [%15.15t] %-40.40logger{39} : %m%n" />  
  
 <!-- 将日志记录到本地文件上 -->  
 <appender name="RollingFileAppender" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${log.path}</file>  
 <encoder>  
 <pattern>${log.pattern}</pattern>  
 <charset>UTF-8</charset>  
 </encoder>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <fileNamePattern>info-%d{yyyy-MM-dd}-%i.log </fileNamePattern>  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 <maxHistory>10</maxHistory>  
 </rollingPolicy>  
 <!-- <append>false</append>  
 <prudent>false</prudent>-->  
 </appender>  
 <!-- 指明输出到文件上日志的级别 -->  
 <root level="info">  
 <appender-ref ref="RollingFileAppender" />  
 </root>  
  
  
 <!-- 将日志发送到kafka上 -->  
 <appender name="KafkaAppender" class="com.github.danielwegener.logback.kafka.KafkaAppender">  
 <encoder class="com.github.danielwegener.logback.kafka.encoding.LayoutKafkaMessageEncoder">  
 <charset>UTF-8</charset>  
 <layout class="net.logstash.logback.layout.LogstashLayout">  
 <includeContext>true</includeContext>  
 <includeCallerData>true</includeCallerData>  
 <customFields>{"system":"kafka"}</customFields>  
 <fieldNames class="net.logstash.logback.fieldnames.ShortenedFieldNames" />  
 </layout>  
 </encoder>  
 <!--kafka topic 需要与配置文件里面的topic一致 否则kafka会沉默并鄙视你 -->  
 <topic>applog</topic>  
 <keyingStrategy class="com.github.danielwegener.logback.kafka.keying.HostNameKeyingStrategy" />  
 <deliveryStrategy class="com.github.danielwegener.logback.kafka.delivery.AsynchronousDeliveryStrategy" />  
 <producerConfig>bootstrap.servers=192.168.153.136:9092,192.168.153.136:9093,192.168.153.136:9094</producerConfig>  
 </appender>  
 <!--  
 <logger name="Application\_ERROR">  
 <appender-ref ref="KafkaAppender" />  
 </logger>  
 -->  
  
 <root level="info">  
 <appender-ref ref="KafkaAppender" />  
 </root>  
</configuration>

### 新增Feign接口CilentFeignApi，内容如下

package cn.kafka.kafka\_service1;  
  
import org.springframework.cloud.openfeign.FeignClient;  
import org.springframework.stereotype.Component;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
@Component  
@FeignClient("KAFKA-CLIENT2")  
public interface MyFeignClient {  
 @RequestMapping("/getString")  
 public String getString();  
  
}

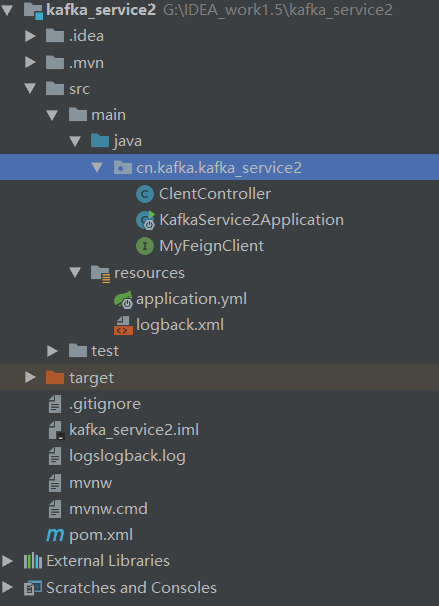
### 新增ClientController，内容如下

package cn.kafka.kafka\_service1;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class ClentController {  
  
 @Autowired  
 private MyFeignClient feignClient;  
 private Logger logger = LoggerFactory.*getLogger*(ClentController.class);  
  
 @RequestMapping("/getString")  
 public String test(){  
 logger.info("service1-->>>>>>>调用>--service2");  
 return feignClient.getString();  
 }  
  
}

### 在启动类ElkKafkaClientApplication上加上注解@EnableEurekaClient和@EnableFeignClients

package cn.kafka.kafka\_service1;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.openfeign.EnableFeignClients;  
  
@SpringBootApplication  
@EnableFeignClients  
public class KafkaService1Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(KafkaService1Application.class, args);  
 }  
  
}

## 新建springboot项目kafka-service2



此项目跟kafka-service1项目相似只需作一下修改

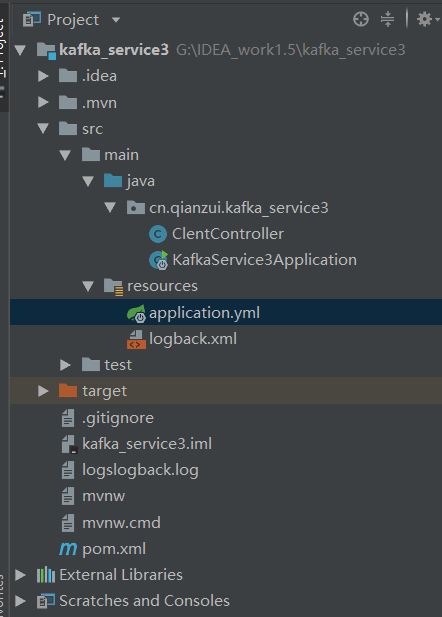
application.yml文件中修改端口号为9030，application.name为KAFKA-CLIENT2

Feign接口中注解修改为@FeignClient("KAFKA-CLIENT3")

### Controller内容修改为

package cn.kafka.kafka\_service2;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class ClentController {  
  
 @Autowired  
 private MyFeignClient feignClient;  
 private Logger logger = LoggerFactory.*getLogger*(ClentController.class);  
  
 @RequestMapping("/getString")  
 public String test(){  
 logger.info("service2-->>>>>>被-->>>>>>>>--service1调用");  
 return feignClient.getString();  
 }  
  
}

## 新建springboot项目kafka-service3



此项目不需要调用其他项目所以去掉Feign的支持

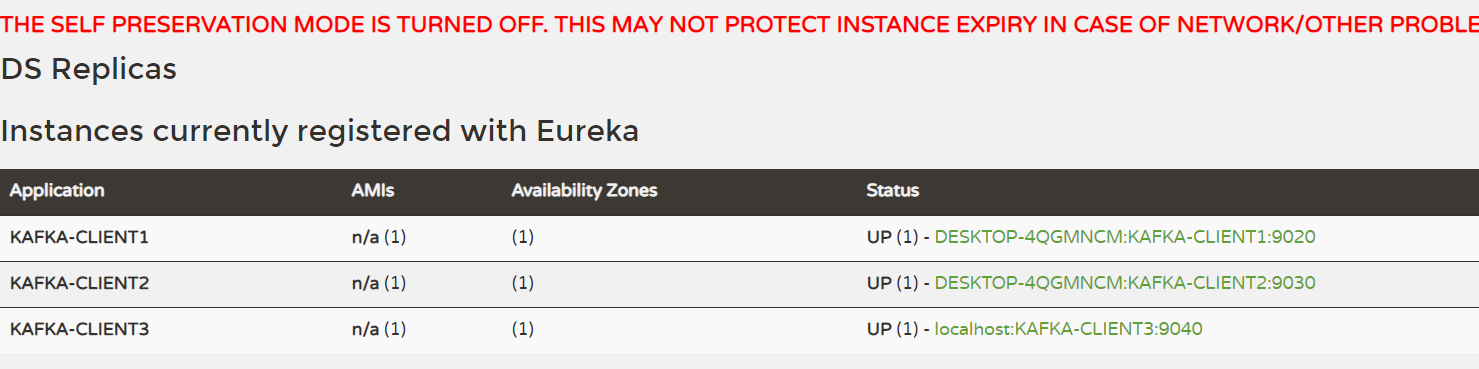
application.yml文件中修改端口号为9040，application.name为 KAFKA-CLIENT3

Controller内容修改为

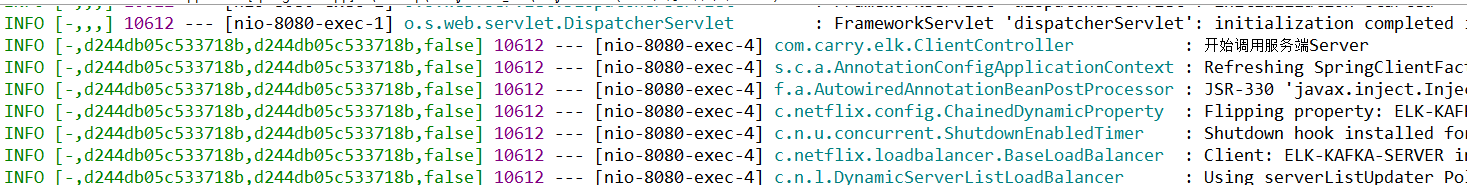
package cn.qianzui.kafka\_service3;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class ClentController {  
  
 private Logger logger = LoggerFactory.*getLogger*(ClentController.class);  
  
 @RequestMapping("/getString")  
 public String test(){  
 logger.info("service3-->>>>>>被-->>>>>>>>--service2调用");  
 return "我是service3------------》》》》》》》》》";  
 }  
  
}

## 项目已写完让我们来看看结果吧

**首先启动eureka-server，然后依次启动kafka-service3， kafka-service2， kafka- service1，浏览器中访问localhost:9020/，查看三个项目是否成功注册到eureka上去**

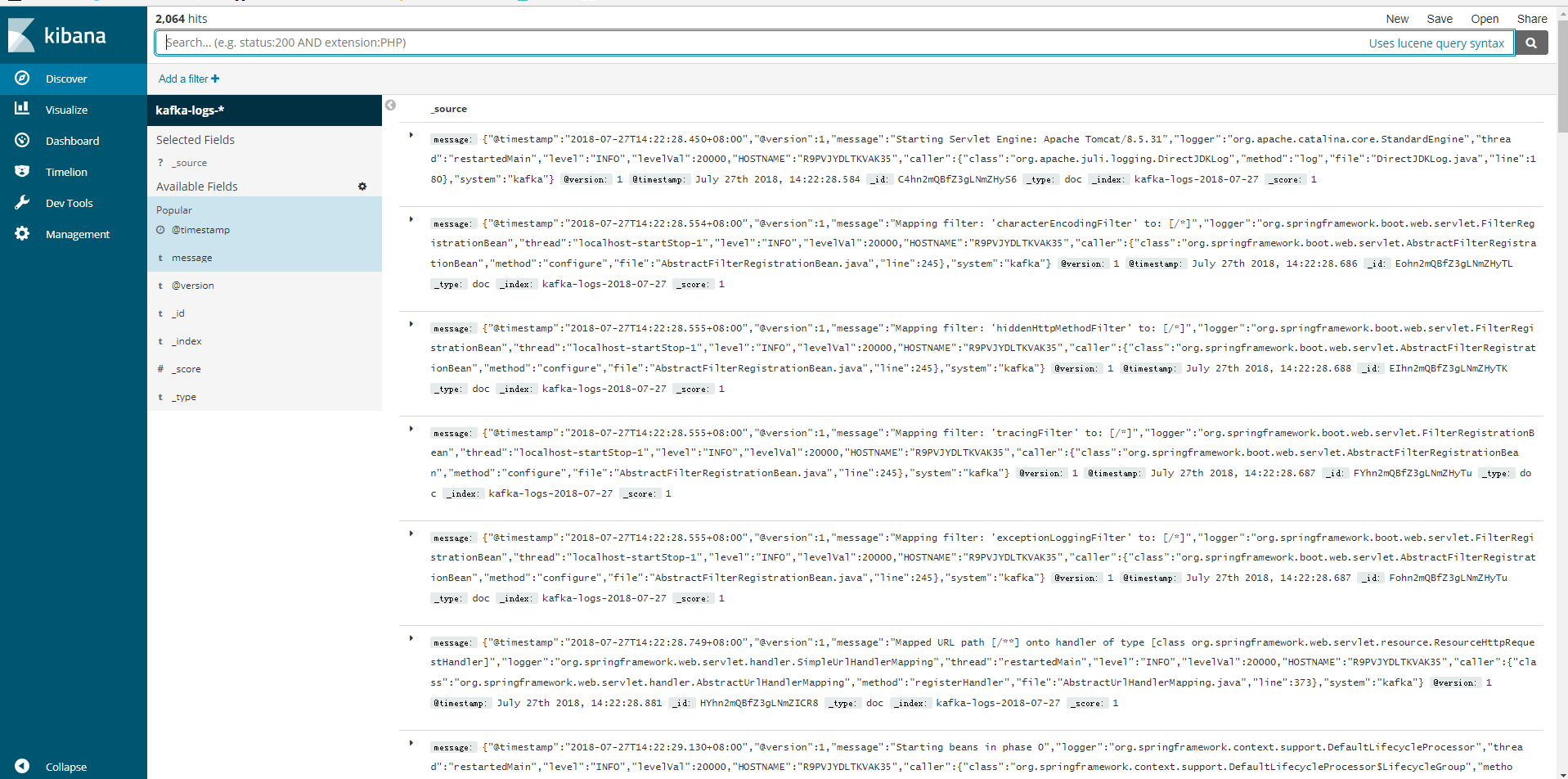


**访问localhost:8080/client，查看项目后台日志**

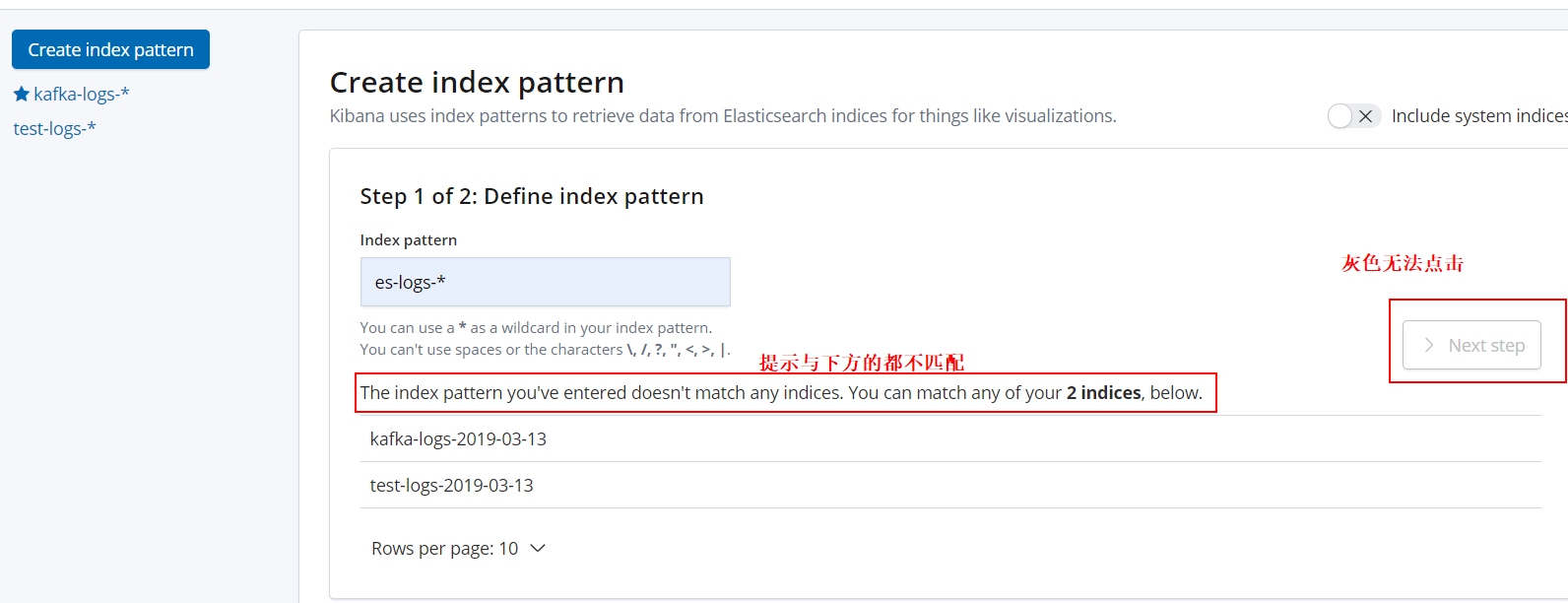
****

**这时我们可以去Kibana上查询日志了，访问<http://192.168.153.165:5601/>，**

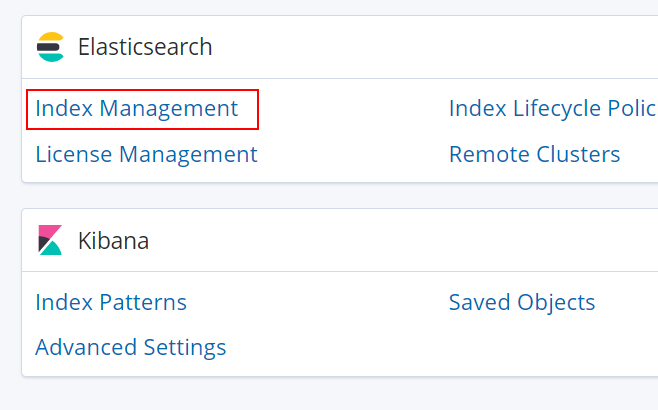
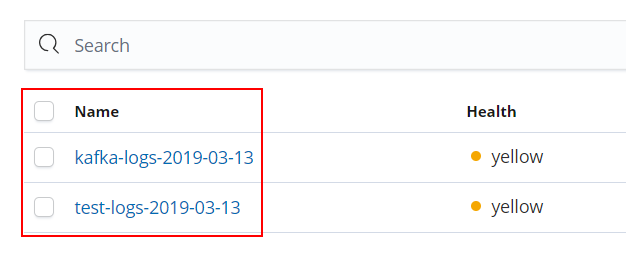
**进入Management创建index pattern kafka-logs-\*，返回Discover页面**

****

有时创建index pattern 会无法点击下一步，并报：不匹配



**进入ElasticSearch的index 管理中看看：**

发现列表中只有两个，还是创建时下边的那两个，这意味着只能创建与这两个相匹配的了。

可是在logstash中，我已经设置里index为 :

 index => "test2-logs-%{+YYYY-MM-dd}"

为什么没有呢？

原来是因为我使用logstash连接了数据库，但是数据库却是空的没有数据，所以赶紧加了几条数据，果然logstash搜索到数据后，就给ElasticSearch打报告了，然后，这个索引模式就出现了。

此时在去创建就可以了。