[完整的ELK+filebeat+kafka笔记](https://www.cnblogs.com/lz0925/p/12061293.html)

### 之前有写过elasticsearch集群和elk集群的博客， 都是基于docker的，使用docker-compose进行编排(K8S暂未掌握)

* 三台服务器搭建es集群：[https://www.cnblogs.com/lz0925/p/12011026.html](https://www.cnblogs.com/lz0925/p/12011026.html" \t "_blank)
* 单机搭建elk集群：[https://www.cnblogs.com/lz0925/p/12018209.html](https://www.cnblogs.com/lz0925/p/12018209.html" \t "_blank)

#### 本文较长，上述两个博文中的内容本文都会重新讲解，建议收藏后进行阅读。本文包含了ELK + kafka(zookeeper) + filebeat整套方案的架构与实施

### ELK介绍

#### ELK = Elasticsearch, Logstash, Kibana 是一套实时数据收集，存储，索引，检索，统计分析及可视化的解决方案。最新版本已经改名为Elastic Stack，并新增了Beats项目。

通常当系统或者应用发生故障时，运维或者开发需要登录到各个服务器上，使用 grep / sed / awk 等 Linux 脚本工具或vim打开文件的方式去日志里查找故障原因。

在没有日志系统的情况下，首先需要定位处理请求的服务器，如果这台服务器部署了多个实例，则需要去每个应用实例的日志目录下去找日志文件。

每个应用实例还会设置日志滚动策略（如：每天生成一个文件），还有日志压缩归档策略等。

这样一系列流程下来，对于我们排查故障以及及时找到故障原因，造成了比较大的麻烦，并且太靠人手工操作。

因此，如果我们能把这些日志集中管理，并提供集中检索功能，不仅可以提高诊断的效率，同时对系统情况有个全面的理解，还可以随时进行扩容，避免事后救火的被动。

总的来说有以下三点用途：

数据查找：通过检索日志信息，定位相应的 bug ，找出解决方案。

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

数据分析：可以做进一步的应用数据分析，当然需要更深入的数据分析，以便产生经济价值。

（摘自：http://www.mamicode.com/info-detail-2741933.html）

当然elasticsearch除了用于ELK，还有许多其它用途， 我们用的只是它的一小部分功能：

分布式的搜索引擎和数据分析引擎

搜索：网站的站内搜索，IT系统的检索

数据分析：电商网站，统计销售排名前10的商家

全文检索，结构化检索，数据分析

全文检索：我想搜索商品名称包含某个关键字的商品

结构化检索：我想搜索商品分类为日化用品的商品都有哪些

数据分析：我们分析每一个商品分类下有多少个商品

对海量数据进行近实时的处理

分布式：ES自动可以将海量数据分散到多台服务器上去存储和检索

海联数据的处理：分布式以后，就可以采用大量的服务器去存储和检索数据，自然而然就可以实现海量数据的处理了

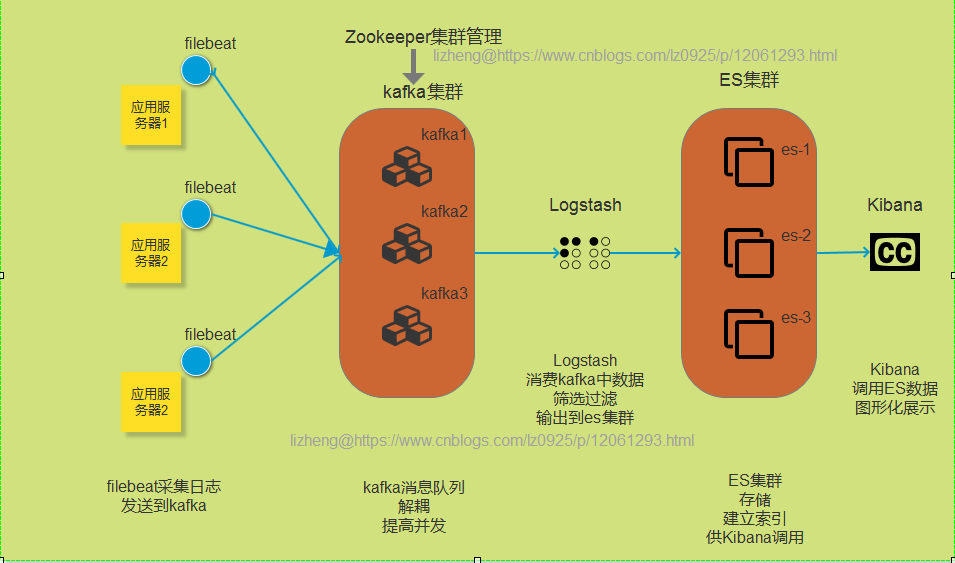
近实时：检索数据要花费1小时（这就不要近实时，离线批处理，batch-processing）；在秒级别对数据进行搜索和分析

（摘自：https://blog.csdn.net/paicmis/article/details/82535018）

#### 想必在平时工作学习中你也听到过elk或es(elasticsearch)，本文为什么在ELK的基础上又使用了kafka(zookeeper)和filebeat呢？

* 首先logstash具有日志采集、过滤、筛选等功能，功能完善但同时体量也会比较大，消耗系统资源自然也多。filebeat作为一个轻量级日志采集工具，虽然没有过滤筛选功能，但是仅仅部署在应用服务器作为我们采集日志的工具可以是说最好的选择了。但我们有些时候可能又需要logstash的过滤筛选功能，所以我们在采集日志时用filebeat，然后交给logstash过滤筛选。
* 其次，logstash的吞吐量是有限的，一旦短时间内filebeat传过来的日志过多会产生堆积和堵塞，对日志的采集也会受到影响，所以在filebeat与logstash中间又加了一层kafka消息队列来缓存或者说解耦，当然redis也是可以的。这样当众多filebeat节点采集大量日志直接放到kafka中，logstash慢慢的进行消费，两边互不干扰。
* 至于zookeeper，想必java的同学都知道，分布式服务管理神器，监控管理kafka的节点注册，topic管理等，同时弥补了kafka集群节点对外界无法感知的问题，kafka实际已经自带了zookeeper，但本教程将会使用独立的zookeeper进行管理，方便后期zookeeper集群的扩展。

### ELK + kafka(zookeeper) + filebeat 的设计模型



### ELK + kafka(zookeeper) + filebeat 的构建

#### 首先说下准备工作

* 1、服务器: 三台，内存最少4G，其中选择一台作为主节点，建议主节点8G内存。
  + PS：单台大内存服务器也可以，本教程为了更接近实际使用所以建议三台。
  + 如果资源有限，可以单独尝试文章开头的两篇博文，也可以尝试本教程，但不保证能所有的步骤都成功，可能会内存不足，内存低于4G不建议尝试）
* 2、系统环境：linux内核不低于3.10.0
* 3、安装docker环境以及docker-compose，详情见我的另一篇博文：[https://www.cnblogs.com/lz0925/p/10985700.html](https://www.cnblogs.com/lz0925/p/10985700.html" \t "_blank)
* 4、首先在三台服务器上搭建es集群，然后选择在一个es节点上搭建ELK，然后在三个节点上安装kafka-zookeeper集群，最后filebeat，各个环节会进行各功能间的可用性测试。
* 5、elk三个软件版本均为7.1.1，版本太低的话搭建完也该升级了，版本太高文档比较少，坑多，目前最新版本7.5.0，现阶段生产环境使用6.x.x的应该比较多，故选择7.1.1。

### es集群的搭建

- 基于单机搭建elasticsearch集群见官网 https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html

#### 本文旨在三台不同的服务器，搭建elasticsearch集群，版本为7.1.1

* 1、服务器列表即配置

172.168.50.40（8G)， 172.168.50.41(16G)， 172.168.50.240(8G)

服务器内存尽量不要低于4G

选用172.168.50.41作为master节点，

* 2、建立存放yml文件的目录

建立文件夹，/root/elasticsearch（随意即可)，用于存放启动elasticsearch容器的yml文件以及es的配置文件

mkdir /root/elasticsearch

* 3、创建docker-compose.yml文件

cd /root/elasticsearch

touch docker-compose.yml

* 4、docker-compose.yml 的文件内容如下

version: '3'

services:

elasticsearch: # 服务名称

image: elasticsearch:7.1.1 # 使用的镜像

container\_name: elasticsearch # 容器名称

restart: always # 失败自动重启策略

environment:

- node.name=node-41 # 节点名称，集群模式下每个节点名称唯一

- network.publish\_host=172.168.50.41 # 用于集群内各机器间通信,对外使用，其他机器访问本机器的es服务，一般为本机宿主机IP

- network.host=0.0.0.0 # 设置绑定的ip地址，可以是ipv4或ipv6的，默认为0.0.0.0，即本机

- discovery.seed\_hosts=172.168.50.40,172.168.50.240,172.168.50.41 # es7.0之后新增的写法，写入候选主节点的设备地址，在开启服务后，如果master挂了，哪些可以被投票选为主节点

- cluster.initial\_master\_nodes=172.168.50.40,172.168.50.240,172.168.50.41 # es7.0之后新增的配置，初始化一个新的集群时需要此配置来选举master

- cluster.name=es-cluster # 集群名称，相同名称为一个集群， 三个es节点须一致

# - http.cors.enabled=true # 是否支持跨域，是：true // 这里设置不起作用，但是可以将此文件映射到宿主机进行修改，然后重启，解决跨域

# - http.cors.allow-origin="\*" # 表示支持所有域名 // 这里设置不起作用，但是可以将此文件映射到宿主机进行修改，然后重启，解决跨域

- bootstrap.memory\_lock=true # 内存交换的选项，官网建议为true

- "ES\_JAVA\_OPTS=-Xms512m -Xmx512m" # 设置内存，如内存不足，可以尝试调低点

ulimits: # 栈内存的上限

memlock:

soft: -1 # 不限制

hard: -1 # 不限制

volumes:

- /root/elasticsearch/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml # 将容器中es的配置文件映射到本地，设置跨域， 否则head插件无法连接该节点

- esdata:/usr/share/elasticsearch/data # 存放数据的文件， 注意：这里的esdata为 顶级volumes下的一项。

ports:

- 9200:9200 # http端口，可以直接浏览器访问

- 9300:9300 # es集群之间相互访问的端口，jar之间就是通过此端口进行tcp协议通信，遵循tcp协议。

volumes:

esdata:

driver: local # 会生成一个对应的目录和文件，如何查看，下面有说明。

* 另外两台服务器也照着这个配置进行配置，但 publish\_host，以及 node.name 需要改一下即可。
* 内存设置，三台服务器都需要进行设置

两种设置内存的方式：

1、设置简单，但机器重启后需再次设置

sysctl -w vm.max\_map\_count=262144

2、直接修改配置文件， 进入sysctl.conf文件添加一行（解决容器内存权限过小问题）

vi /etc/sysctl.conf

sysctl vm.max\_map\_count=262144 # 添加此行

退出文件后，执行命令： sysctl -p 立即生效

* 然后三台服务器依次执行 docker-compose up -d
* 然后访问 <http://172.168.50.41:9200/_cluster/health?pretty> 查看是否集群正常运行, 正常运行会返回如下信息

{

"cluster\_name" : "es-cluster",

"status" : "green",

"timed\_out" : false,

"number\_of\_nodes" : 3,

"number\_of\_data\_nodes" : 3,

"active\_primary\_shards" : 0,

"active\_shards" : 0,

"relocating\_shards" : 0,

"initializing\_shards" : 0,

"unassigned\_shards" : 0,

"delayed\_unassigned\_shards" : 0,

"number\_of\_pending\_tasks" : 0,

"number\_of\_in\_flight\_fetch" : 0,

"task\_max\_waiting\_in\_queue\_millis" : 0,

"active\_shards\_percent\_as\_number" : 100.0

}

* elasticsearch.yml 文件内容如下：

network.host: 0.0.0.0

http.cors.enabled: true # 是否支持跨域

http.cors.allow-origin: "\*" # 表示支持所有域名

* 上面说到的顶级volumes， 如何查看挂载卷在宿主机的位置呢？

# docker volume create elk\_data // 创建一个自定义容器卷，本教程内不需要执行，我们的docker-compose.yml会帮我们自动执行改命令。

# docker volume ls // 查看所有容器卷，

# docker volume inspect elk\_data // 查看指定容器卷详情信息, 包括真实目录

注意： 如果你要删除一个挂载卷，或者重新生成，请执行删除卷操作

# docker volume rm elk\_data // 直接执行这个命令，同时会删除文件，但是先删除文件的话，必须再次执行此命令，否则可能导致该节点无法加入集群。

* 以上就是es集群的搭建，如果出现权限不足等简单问题，可以百度自行解决。
* 接下来我们将在172.168.50.41服务器上搭建elk服务，用elk中的es替换掉50.41服务器上之前的es。

### 搭建ELK服务

// 下载elasticsearch，logstash，kibana的镜像，建议提前下载好。 自es5开始，一般三个软件的版本都保持一致了。如果下载网络有问题，尝试重新配置一下docker的镜像源地址。

docker pull docker.elastic.co/elasticsearch/elasticsearch:7.1.1 && docker pull docker.elastic.co/logstash/logstash:7.1.1 && docker pull docker.elastic.co/kibana/kibana:7.1.1

### 配置

* 我们建立一个目录，用来存放yml文件（docker-compose启动一组编排过的容器时使用）

mkdir -p /root/elk（位置和名称都随意） # 与刚才的elasticsearch目录进行区别

cd /root/elk

vim docker-compose.yml

* docker-compose.yml文件内容如下

version: '3'

services: # 服务（s代表复数）

elasticsearch: # 服务名称：elasticsearch

image: docker.elastic.co/elasticsearch/elasticsearch:7.1.1 # 使用的镜像以及版本号

container\_name: elasticsearch7.1.1 # 容器名称

environment: # 环境变量

- node.name=node-41 # 节点名称

- network.publish\_host=172.168.50.41 # 用于集群内各机器间通信,对外使用，其他机器访问本机器的es服务，一般为本机宿主机IP

- network.host=0.0.0.0 # 设置绑定的ip地址，可以是ipv4或ipv6的，默认为0.0.0.0，即本机

- discovery.seed\_hosts=172.168.50.40,172.168.50.240,172.168.50.41 # es7.0之后新增的写法，写入候选主节点的设备地址，在开启服务后，如果master挂了，哪些可以被投票选为主节点

- cluster.initial\_master\_nodes=172.168.50.40,172.168.50.240,172.168.50.41 # es7.0之后新增的配置，初始化一个新的集群时需要此配置来选举master

- cluster.name=es-cluster # 集群名称，相同名称为一个集群， 三个es节点须一致

- bootstrap.memory\_lock=true # 内存交换的选项，官网建议为true

- "ES\_JAVA\_OPTS=-Xms512m -Xmx512m" # 设置内存，如内存不足，可以尝试调低点

#- discovery.type=single-node # 单es节点模式，这里我们就不启用，启用的话，除了两个内存外其他的环境配置都不需要。

volumes:

- esdata:/usr/share/elasticsearch/data # 设置es数据存放的目录

- /root/elk/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml # 映射es容器中的配置文件到宿主机

hostname: elasticsearch # 服务hostname

ulimits: # 栈内存的上限

memlock:

soft: -1 # 不限制

hard: -1 # 不限制

restart: always # 重启策略，失败时总是重启

ports:

- 9200:9200 # http端口，可以直接浏览器访问

- 9300:9300 # es集群之间相互访问的端口，jar之间就是通过此端口进行tcp协议通信，遵循tcp协议。

kibana: # 服务名称：kibana

image: docker.elastic.co/kibana/kibana:7.1.1 # 使用的镜像及版本号

container\_name: kibana7.1.1 # 容器名称

environment: # 环境配置

- elasticsearch.hosts=http://elasticsearch:9200 # 设置连接的es节点，此处的elasticsearch即上面的elasticsearch服务中的hostname:elasticsearch

# - TZ=Asia/Shanghai # 设置容器内时区

hostname: kibana # 服务hostname

depends\_on:

- elasticsearch # 依赖es服务，必须先启动es才会启动kibana

restart: always

ports:

- 5601:5601 # 访问的端口号

logstash: # 服务名称：logstash

image: docker.elastic.co/logstash/logstash:7.1.1

container\_name: logstash7.1.1

hostname: logstash

restart: always

# volumes: # 挂载卷，先注释，等ELK启动起来之后，从容器中复制出来一份，下面会说

# - /root/elk/logstash/config/:/usr/share/logstash/config/ # logstash 配置文件目录

# - /root/elk/logstash/pipeline/:/usr/share/logstash/pipeline/ # logstash 的采集与输入的配置文件目录

depends\_on:

- elasticsearch

# ports:

# - 9600:9600 # 端口号，后来没用到

# - 5044:5044 # filebeat直接传递数据给logstash时默认使用的端口号，测试了一下，没用

volumes:

esdata:

driver: local

* 上面用到了/root/elk/logstash/ 下面的config目录与pipeline目录，如何得到这两个目录呢？

// 先停止之前该服务器上的es节点  
docker stop es节点ID  
docker rm es节点ID

* + 1、在/root/elk目录下，docker-compose up -d 启动elk
  + 2、mkdir logstash
  + 3、从容器中复制两个目录到logstash下,这样我们就有了初始配置文件

docker cp 容器ID:/usr/share/logstash/config/ logstash // 容器ID为Logstash的ID

docker cp 容器ID:/usr/share/logstash/pipeline/ logstash

* 修改配置文件 config目录中的logstash.yml文件，内容如下

http.host: "0.0.0.0"

xpack.monitoring.enabled: true

xpack.monitoring.elasticsearch.hosts: [ "172.168.50.41:9200","172.168.50.40:9200","172.168.50.240:9200" ]

// elasticsearch.hosts 为你的es服务器列表以及端口号，用于将数据存储到es

* 修改pipeline目录中的logstash.conf文件（此文件是通过config目录下的 pipelines.yml 来指定的），内容如下

input{ # 输入组件

kafka{ # 从kafka消费数据

bootstrap\_servers => ["172.168.50.41:9092,172.168.50.41:9097,172.168.50.41:9098"] # kafka节点的IP+端口号

#topics => "%{[@metadata][topic]}" # 使用kafka传过来的topic

topics\_pattern => "elk-.\*" # 使用正则匹配topic

codec => "json" # 数据格式

consumer\_threads => 3 # 消费线程数量

decorate\_events => true # 可向事件添加Kafka元数据，比如主题、消息大小的选项，这将向logstash事件中添加一个名为kafka的字段，

auto\_offset\_reset => "latest" # 自动重置偏移量到最新的偏移量

group\_id => "logstash-node-41" # 消费组ID，多个有相同group\_id的logstash实例为一个消费组

}

}

filter {

# 当非业务字段时，无traceId则移除

if ([message] =~ "traceId=null") { # 过滤组件，这里只是展示，无实际意义，根据自己的业务需求进行过滤

drop {}

}

}

output { # 输出组件

elasticsearch {

# Logstash输出到es

hosts => ["172.168.50.41:9200", "172.168.50.40:9200", "172.168.50.240:9200"]

# index => "%{[fields][source]}" #直接在日志中匹配，索引会去掉elk

index => "%{[@metadata][topic]}-%{+YYYY-MM-dd}" # 以日期建索引

#flush\_size => 20000

#idle\_flush\_time => 10

#sniffing => true

#template\_overwrite => false

}

# stdout {

# codec => rubydebug

# }

}

# 可以写多个input，但是output中的index即索引要处理好，根据过滤组件或者kafka传过来的不同话题，应有不同的索引

# 也可以写多个output，向其他程序（非es）传送数据

* 有了这两个目录后，将docker-compose.yml文件中的三行注释打开

volumes: # 挂载卷，先注释，等ELK启动起来之后，从容器中复制出来一份，下面会说

- /root/elk/logstash/config/:/usr/share/logstash/config/ # logstash 配置文件目录

- /root/elk/logstash/pipeline/:/usr/share/logstash/pipeline/ # logstash 的采集与输入的配置文件目录

* 在/root/elk目录，重新编译和启动elk

// 启动ELK

docker-compose up -d --force --build

* 出现done表示成功，docker-compose logs 查看日志（分别输出elk三个服务的日志）执行docker-compose ps 可以看到三个服务的运行状态
* 在浏览器输入http://IP:5601/ 访问kibana。
* 可以使用head监控es

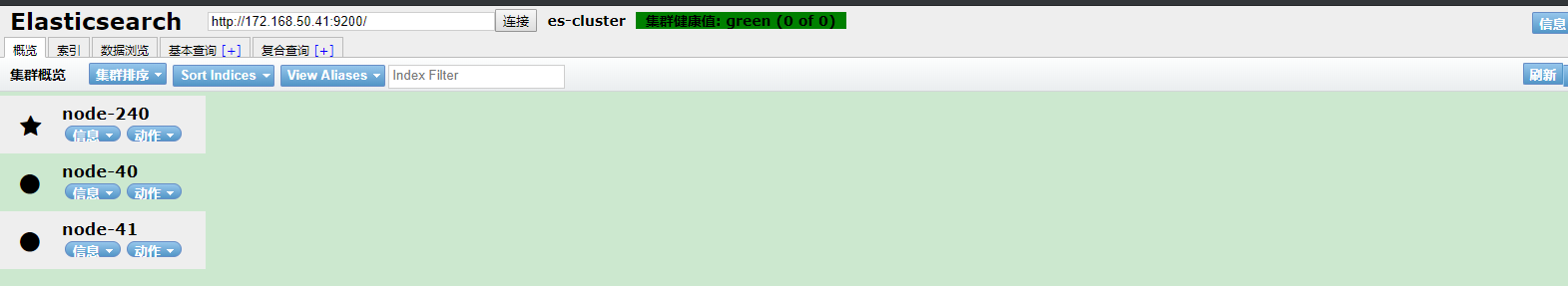
// 拉取镜像

docker pull mobz/elasticsearch-head:5

// 启动

docker run -d --name es\_admin -p 9100:9100 mobz/elasticsearch-head:5

* 在浏览器输入http://IP:9100



* 注意，IP地址要使用你es所在服务器的地址，然后点击连接，出现类似的显示说明，没问题，本教程是三个节点，当关掉一个es会变成两个节点。
* 如果点击连接没有反应，F12查看network是403，或者200，但是console中提示跨域，那么应该是跨域未设置好，重新回去检查一遍，看看容器内配置文件是否加入了那两行跨域。
* 至此，我们的Elasticsearch（集群）+ Logstash + Kibana 搭建完成， 你可以在kibana中的dev-tools菜单中操作es，甚至向es中添加数据，或者通过其他方式向es中添加数据进行测试。
* Logstash可以采集数据， 过滤数据； Elasticsearch 可以存储和索引数据； Kibana提供可视化展示和操作
* 这个时候就可以配置Logstash采集数据然后想es输入了，但是我们并不满足于此，我想在需要采集日志的应用服务器上安装 filebeat，然后经filebeat将数据转发给Logstash，Logstash只进行过滤即可（也可以filebeat直接将数据传给es，不要Logstash，看个人需求）。
* 但我没有保留filebeat直接传给Logstash的配置文件，所以我们直接在此时将kafka也接入进来，不再演示filebeat与Logstash直接交互的情景。
* 不用担心，我们尽量降低复杂度，比如，现在我们不考虑ELK，也不考虑filebeat，抛开这些东西，仅仅搭建一个kafka集群。
* kafka依赖zookeeper，我们前面说过，本文使用独立的zookeeper集群，所以接下来介绍如何搭建zookeeper

### zookeeper集群的搭建

* 依然是三节点集群，但是为了减少复杂性，我们直接在172.168.50.41（内存最大的服务器）上通过docker容器搭建单机zookeeper集群。
* 建立目录，并创建yml文件

mkdir -p /root/zookeeper

cd /root/zookeeper

vim docker-compose.yml

* docker-compose.yml文件内容如下：

version: '2'

services:

zoo1:

image: zookeeper:3.4.14 # 尽量使用3.4版本，3.4.14的镜像就可以。 3.5坑太多

restart: always

hostname: zoo1

container\_name: zoo1

ports:

- 2181:2181 # 端口号

volumes:

- /usr/local/docker-compose/zk/zk1/data:/data # 数据文件存放目录

- /usr/local/docker-compose/zk/zk1/datalog:/datalog # 日志文件存放目录

- /root/zookeeper/zoo.cfg:/conf/zoo.cfg # zookeeper的配置文件，下面会给出该文件内容

environment:

ZOO\_MY\_ID: 1 # 表示 ZK服务的 id, 它是1-255 之间的整数, 必须在集群中唯一

ZOO\_SERVERS: server.1=0.0.0.0:2888:3888;2181 server.2=zoo2:2888:3888;2181 server.3=zoo3:2888:3888;2181 # ZK集群的主机列表即端口号

# ZOOKEEPER\_CLIENT\_PORT: 2181

networks:

default:

ipv4\_address: 172.23.0.11 # 集群使用的网络

zoo2:

image: zookeeper:3.4.14

restart: always

hostname: zoo2

container\_name: zoo2

ports:

- 2180:2181

volumes:

- /usr/local/docker-compose/zk/zk2/data:/data

- /usr/local/docker-compose/zk/zk2/datalog:/datalog

- /root/zookeeper/zoo.cfg:/conf/zoo.cfg

environment:

ZOO\_MY\_ID: 2

#ZOOKEEPER\_CLIENT\_PORT: 2181

ZOO\_SERVERS: server.1=zoo1:2888:3888;2181 server.2=0.0.0.0:2888:3888;2181 server.3=zoo3:2888:3888;2181

networks:

default:

ipv4\_address: 172.23.0.12

zoo3:

image: zookeeper:3.4.14

restart: always

hostname: zoo3

container\_name: zoo3

ports:

- 2179:2181

volumes:

- /usr/local/docker-compose/zk/zk3/data:/data

- /usr/local/docker-compose/zk/zk3/datalog:/datalog

- /root/zookeeper/zoo.cfg:/conf/zoo.cfg

environment:

ZOO\_MY\_ID: 3

#ZOOKEEPER\_CLIENT\_PORT: 2181

ZOO\_SERVERS: server.1=zoo1:2888:3888;2181 server.2=zoo2:2888:3888;2181 server.3=0.0.0.0:2888:3888;2181

networks:

default:

ipv4\_address: 172.23.0.13

networks:

default:

external:

name: zookeeper\_network # 网络名称，此网络需要进行创建

* zoo.cfg文件内容如下：

tickTime=2000 # 心跳2秒，服务器之间或客户端与服务器之间维持心跳的时间间隔

initLimit=10 # 集群中的follower服务器(F)与leader服务器(L)之间 初始连接 时能容忍的最多心跳数（tickTime的数量）。

syncLimit=5 # 集群中的follower服务器(F)与leader服务器(L)之间 请求和应答 之间能容忍的最多心跳数（tickTime的数量）。

dataDir=/data # 数据目录

dataLogDir=/datalog # 日志文件目录

clientPort=2181 # 客户端连接端口

autopurge.snapRetainCount=3 # 至少需要保留3个快照数据文件和对应的事务日志文件

autopurge.purgeInterval=1 # 进行历史文件自动清理的频率，单位小时

server.1= zoo1:2888:3888 # 定义个服务通信端口，以及投票端口

server.2= zoo2:2888:3888

server.3= zoo3:2888:3888

* 启动前先创建网络

docker network create --driver bridge --subnet 172.23.0.0/25 --gateway 172.23.0.1 zookeeper\_network

* 启动zk集群

当前目录下执行 docker-compose up -d

* 连接zk，并打开客户端

docker run -it --rm --link zoo1:zk1 --link zoo2:zk2 --link zoo3:zk3 --net zookeeper\_network zookeeper:3.4.14 zkCli.sh -server zk1:2181,zk2:2181,zk3:2181

// 执行完上面的命令后，会进去zk的客户端，我们可以进行一些操作，比如

执行 ls / 可以查看根节点信息，然后根据现实内容，查看下一级数据节点的信息

建议等下搭建完kafka集群并创建topic后通过 ls /brokers/topics 来查看topic列表（kafka在zookeeper注册后，创建的话题会被zookeeper记录下来）

zk常用命令

create /test "test"

get /test

set /test "hello test"

delete /test

quit

* 至此zookeeper集群搭建完毕，比较简单。

### kafka集群的搭建

* 老规矩先建立目录，用于存放yml文件以及kafka的配置文件

mkdir -p /root/kafka

cd /root/kafka

vim docker-compose.yml

* docker-compose.yml 文件内容如下：

version: '2'

services:

kafka1:

image: wurstmeister/kafka:2.12-2.3.0

restart: always

hostname: kafka1

container\_name: kafka1

ports:

- "9092:9092"

environment:

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://172.168.50.41:9092 # 宿主机的IP地址而非容器的IP，及暴露出来的端口

KAFKA\_ADVERTISED\_HOST\_NAME: 172.168.50.41 # 外网访问地址

KAFKA\_ADVERTISED\_PORT: 9092 # 端口

KAFKA\_ZOOKEEPER\_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181 # 连接的zookeeper服务及端口

JMX\_PORT: 9966 # kafka需要监控broker和topic的数据的时候,是需要开启jmx\_port的

volumes:

- /usr/local/docker-compose/kafka/kafka1:/kafka # kafka数据文件存储目录

- /root/kafka/kafka-run-class.sh:/opt/kafka\_2.12-2.3.0/bin/kafka-run-class.sh # kafka启动时的配置文件，我们修改了一部分代码，用来解决java.rmi.server.ExportException: Port already in use这个问题。过程详见https://github.com/apache/kafka/pull/1983/commits/2c5d40e946bcc149b1a9b2c01eced4ae47a734c5。

external\_links: # 连接本compose文件以外的容器

- zoo1

- zoo2

- zoo3

networks:

default:

ipv4\_address: 172.23.0.14 # 使用zookeeper创建的网络中一个IP

kafka2:

image: wurstmeister/kafka:2.12-2.3.0

restart: always

hostname: kafka2

container\_name: kafka2

ports:

- "9097:9092"

environment:

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://172.168.50.41:9097

KAFKA\_ADVERTISED\_HOST\_NAME: 172.168.50.41

KAFKA\_ADVERTISED\_PORT: 9097

KAFKA\_ZOOKEEPER\_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181

JMX\_PORT: 9988

volumes:

- /usr/local/docker-compose/kafka/kafka2:/kafka

- /root/kafka/kafka-run-class.sh:/opt/kafka\_2.12-2.3.0/bin/kafka-run-class.sh

external\_links: # 连接compose文件以外的container

- zoo1

- zoo2

- zoo3

networks:

default:

ipv4\_address: 172.23.0.15

kafka3:

image: wurstmeister/kafka:2.12-2.3.0

restart: always

hostname: kafka3

container\_name: kafka3

ports:

- "9098:9092"

environment:

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://172.168.50.41:9098

KAFKA\_ADVERTISED\_HOST\_NAME: 172.168.50.41

KAFKA\_ADVERTISED\_PORT: 9098

KAFKA\_ZOOKEEPER\_CONNECT: zoo1:2181,zoo2:2181,zoo3:2181

JMX\_PORT: 9977

volumes:

- /usr/local/docker-compose/kafka/kafka3

- /root/kafka/kafka-run-class.sh:/opt/kafka\_2.12-2.3.0/bin/kafka-run-class.sh

external\_links: # 连接compose文件以外的container

- zoo1

- zoo2

- zoo3

networks:

default:

ipv4\_address: 172.23.0.16

kafka-manager: # kafka管理和监控的一个工具

image: sheepkiller/kafka-manager:latest # 此时我使用的最新版本为5.0

restart: always

container\_name: kafa-manager

hostname: kafka-manager

ports:

- "9099:9000" # 浏览器访问的端口

links: # 连接本compose文件创建的container

- kafka1

- kafka2

- kafka3

external\_links: # 连接compose文件以外的container

- zoo1

- zoo2

- zoo3

environment:

ZK\_HOSTS: zoo1:2181,zoo2:2181,zoo3:2181 # zk服务列表

KAFKA\_BROKERS: kafka1:9092,kafka2:9097,kafka3:9098 # kafka节点列表及端口号

APPLICATION\_SECRET: letmein # 应用秘钥，默认letmein

KM\_ARGS: -Djava.net.preferIPv4Stack=true # 在ipv4的机器上想获取到完整的机器名， 应该是IPV4网络相关的东西

networks:

default:

ipv4\_address: 172.23.0.10 # IP地址，zookeeper创建的网络中的一个IP

networks:

default:

external:

name: zookeeper\_network # 使用zookeeper创建的网络

* kafka-run-class.sh 文件内容如下，可能会出现有权限问题, 使用下面我修改后的文件
* 或者 参考该连接 <https://github.com/apache/kafka/pull/1983/commits/2c5d40e946bcc149b1a9b2c01eced4ae47a734c5>

#!/bin/bash

# Licensed to the Apache Software Foundation (ASF) under one or more

# contributor license agreements. See the NOTICE file distributed with

# this work for additional information regarding copyright ownership.

# The ASF licenses this file to You under the Apache License, Version 2.0

# (the "License"); you may not use this file except in compliance with

# the License. You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

if [ $# -lt 1 ];

then

echo "USAGE: $0 [-daemon] [-name servicename] [-loggc] classname [opts]"

exit 1

fi

# CYGWIN == 1 if Cygwin is detected, else 0.

if [[ $(uname -a) =~ "CYGWIN" ]]; then

CYGWIN=1

else

CYGWIN=0

fi

if [ -z "$INCLUDE\_TEST\_JARS" ]; then

INCLUDE\_TEST\_JARS=false

fi

# Exclude jars not necessary for running commands.

regex="(-(test|test-sources|src|scaladoc|javadoc)\.jar|jar.asc)$"

should\_include\_file() {

if [ "$INCLUDE\_TEST\_JARS" = true ]; then

return 0

fi

file=$1

if [ -z "$(echo "$file" | egrep "$regex")" ] ; then

return 0

else

return 1

fi

}

################ lizheng #################

# need to check if called to start server or client

# in order to correctly decide about JMX\_PORT

ISKAFKASERVER="false"

if [[ "$\*" =~ "kafka.Kafka" ]]; then

ISKAFKASERVER="true"

fi

################ lizheng #################

base\_dir=$(dirname $0)/..

if [ -z "$SCALA\_VERSION" ]; then

SCALA\_VERSION=2.12.8

fi

if [ -z "$SCALA\_BINARY\_VERSION" ]; then

SCALA\_BINARY\_VERSION=$(echo $SCALA\_VERSION | cut -f 1-2 -d '.')

fi

# run ./gradlew copyDependantLibs to get all dependant jars in a local dir

shopt -s nullglob

for dir in "$base\_dir"/core/build/dependant-libs-${SCALA\_VERSION}\*;

do

CLASSPATH="$CLASSPATH:$dir/\*"

done

for file in "$base\_dir"/examples/build/libs/kafka-examples\*.jar;

do

if should\_include\_file "$file"; then

CLASSPATH="$CLASSPATH":"$file"

fi

done

if [ -z "$UPGRADE\_KAFKA\_STREAMS\_TEST\_VERSION" ]; then

clients\_lib\_dir=$(dirname $0)/../clients/build/libs

streams\_lib\_dir=$(dirname $0)/../streams/build/libs

rocksdb\_lib\_dir=$(dirname $0)/../streams/build/dependant-libs-${SCALA\_VERSION}

else

clients\_lib\_dir=/opt/kafka-$UPGRADE\_KAFKA\_STREAMS\_TEST\_VERSION/libs

streams\_lib\_dir=$clients\_lib\_dir

rocksdb\_lib\_dir=$streams\_lib\_dir

fi

for file in "$clients\_lib\_dir"/kafka-clients\*.jar;

do

if should\_include\_file "$file"; then

CLASSPATH="$CLASSPATH":"$file"

fi

done

for file in "$streams\_lib\_dir"/kafka-streams\*.jar;

do

if should\_include\_file "$file"; then

CLASSPATH="$CLASSPATH":"$file"

fi

done

if [ -z "$UPGRADE\_KAFKA\_STREAMS\_TEST\_VERSION" ]; then

for file in "$base\_dir"/streams/examples/build/libs/kafka-streams-examples\*.jar;

do

if should\_include\_file "$file"; then

CLASSPATH="$CLASSPATH":"$file"

fi

done

else

VERSION\_NO\_DOTS=`echo $UPGRADE\_KAFKA\_STREAMS\_TEST\_VERSION | sed 's/\.//g'`

SHORT\_VERSION\_NO\_DOTS=${VERSION\_NO\_DOTS:0:((${#VERSION\_NO\_DOTS} - 1))} # remove last char, ie, bug-fix number

for file in "$base\_dir"/streams/upgrade-system-tests-$SHORT\_VERSION\_NO\_DOTS/build/libs/kafka-streams-upgrade-system-tests\*.jar;

do

if should\_include\_file "$file"; then

CLASSPATH="$file":"$CLASSPATH"

fi

done

fi

for file in "$rocksdb\_lib\_dir"/rocksdb\*.jar;

do

CLASSPATH="$CLASSPATH":"$file"

done

for file in "$base\_dir"/tools/build/libs/kafka-tools\*.jar;

do

if should\_include\_file "$file"; then

CLASSPATH="$CLASSPATH":"$file"

fi

done

for dir in "$base\_dir"/tools/build/dependant-libs-${SCALA\_VERSION}\*;

do

CLASSPATH="$CLASSPATH:$dir/\*"

done

for cc\_pkg in "api" "transforms" "runtime" "file" "json" "tools" "basic-auth-extension"

do

for file in "$base\_dir"/connect/${cc\_pkg}/build/libs/connect-${cc\_pkg}\*.jar;

do

if should\_include\_file "$file"; then

CLASSPATH="$CLASSPATH":"$file"

fi

done

if [ -d "$base\_dir/connect/${cc\_pkg}/build/dependant-libs" ] ; then

CLASSPATH="$CLASSPATH:$base\_dir/connect/${cc\_pkg}/build/dependant-libs/\*"

fi

done

# classpath addition for release

for file in "$base\_dir"/libs/\*;

do

if should\_include\_file "$file"; then

CLASSPATH="$CLASSPATH":"$file"

fi

done

for file in "$base\_dir"/core/build/libs/kafka\_${SCALA\_BINARY\_VERSION}\*.jar;

do

if should\_include\_file "$file"; then

CLASSPATH="$CLASSPATH":"$file"

fi

done

shopt -u nullglob

if [ -z "$CLASSPATH" ] ; then

echo "Classpath is empty. Please build the project first e.g. by running './gradlew jar -PscalaVersion=$SCALA\_VERSION'"

exit 1

fi

# JMX settings

if [ -z "$KAFKA\_JMX\_OPTS" ]; then

KAFKA\_JMX\_OPTS="-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false "

fi

# JMX port to use

#if [ $JMX\_PORT ]; then

if [ $JMX\_PORT ] && [ -z "$ISKAFKASERVER" ]; then

KAFKA\_JMX\_OPTS="$KAFKA\_JMX\_OPTS -Dcom.sun.management.jmxremote.port=$JMX\_PORT "

fi

# Log directory to use

if [ "x$LOG\_DIR" = "x" ]; then

LOG\_DIR="$base\_dir/logs"

fi

# Log4j settings

if [ -z "$KAFKA\_LOG4J\_OPTS" ]; then

# Log to console. This is a tool.

LOG4J\_DIR="$base\_dir/config/tools-log4j.properties"

# If Cygwin is detected, LOG4J\_DIR is converted to Windows format.

(( CYGWIN )) && LOG4J\_DIR=$(cygpath --path --mixed "${LOG4J\_DIR}")

KAFKA\_LOG4J\_OPTS="-Dlog4j.configuration=file:${LOG4J\_DIR}"

else

# create logs directory

if [ ! -d "$LOG\_DIR" ]; then

mkdir -p "$LOG\_DIR"

fi

fi

# If Cygwin is detected, LOG\_DIR is converted to Windows format.

(( CYGWIN )) && LOG\_DIR=$(cygpath --path --mixed "${LOG\_DIR}")

KAFKA\_LOG4J\_OPTS="-Dkafka.logs.dir=$LOG\_DIR $KAFKA\_LOG4J\_OPTS"

# Generic jvm settings you want to add

if [ -z "$KAFKA\_OPTS" ]; then

KAFKA\_OPTS=""

fi

# Set Debug options if enabled

if [ "x$KAFKA\_DEBUG" != "x" ]; then

# Use default ports

DEFAULT\_JAVA\_DEBUG\_PORT="5005"

if [ -z "$JAVA\_DEBUG\_PORT" ]; then

JAVA\_DEBUG\_PORT="$DEFAULT\_JAVA\_DEBUG\_PORT"

fi

# Use the defaults if JAVA\_DEBUG\_OPTS was not set

DEFAULT\_JAVA\_DEBUG\_OPTS="-agentlib:jdwp=transport=dt\_socket,server=y,suspend=${DEBUG\_SUSPEND\_FLAG:-n},address=$JAVA\_DEBUG\_PORT"

if [ -z "$JAVA\_DEBUG\_OPTS" ]; then

JAVA\_DEBUG\_OPTS="$DEFAULT\_JAVA\_DEBUG\_OPTS"

fi

echo "Enabling Java debug options: $JAVA\_DEBUG\_OPTS"

KAFKA\_OPTS="$JAVA\_DEBUG\_OPTS $KAFKA\_OPTS"

fi

# Which java to use

if [ -z "$JAVA\_HOME" ]; then

JAVA="java"

else

JAVA="$JAVA\_HOME/bin/java"

fi

# Memory options

if [ -z "$KAFKA\_HEAP\_OPTS" ]; then

KAFKA\_HEAP\_OPTS="-Xmx256M"

fi

# JVM performance options

if [ -z "$KAFKA\_JVM\_PERFORMANCE\_OPTS" ]; then

KAFKA\_JVM\_PERFORMANCE\_OPTS="-server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent -Djava.awt.headless=true"

fi

while [ $# -gt 0 ]; do

COMMAND=$1

case $COMMAND in

-name)

DAEMON\_NAME=$2

CONSOLE\_OUTPUT\_FILE=$LOG\_DIR/$DAEMON\_NAME.out

shift 2

;;

-loggc)

if [ -z "$KAFKA\_GC\_LOG\_OPTS" ]; then

GC\_LOG\_ENABLED="true"

fi

shift

;;

-daemon)

DAEMON\_MODE="true"

shift

;;

\*)

break

;;

esac

done

# GC options

GC\_FILE\_SUFFIX='-gc.log'

GC\_LOG\_FILE\_NAME=''

if [ "x$GC\_LOG\_ENABLED" = "xtrue" ]; then

GC\_LOG\_FILE\_NAME=$DAEMON\_NAME$GC\_FILE\_SUFFIX

# The first segment of the version number, which is '1' for releases before Java 9

# it then becomes '9', '10', ...

# Some examples of the first line of `java --version`:

# 8 -> java version "1.8.0\_152"

# 9.0.4 -> java version "9.0.4"

# 10 -> java version "10" 2018-03-20

# 10.0.1 -> java version "10.0.1" 2018-04-17

# We need to match to the end of the line to prevent sed from printing the characters that do not match

JAVA\_MAJOR\_VERSION=$($JAVA -version 2>&1 | sed -E -n 's/.\* version "([0-9]\*).\*$/\1/p')

if [[ "$JAVA\_MAJOR\_VERSION" -ge "9" ]] ; then

KAFKA\_GC\_LOG\_OPTS="-Xlog:gc\*:file=$LOG\_DIR/$GC\_LOG\_FILE\_NAME:time,tags:filecount=10,filesize=102400"

else

KAFKA\_GC\_LOG\_OPTS="-Xloggc:$LOG\_DIR/$GC\_LOG\_FILE\_NAME -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=100M"

fi

fi

# Remove a possible colon prefix from the classpath (happens at lines like `CLASSPATH="$CLASSPATH:$file"` when CLASSPATH is blank)

# Syntax used on the right side is native Bash string manipulation; for more details see

# http://tldp.org/LDP/abs/html/string-manipulation.html, specifically the section titled "Substring Removal"

CLASSPATH=${CLASSPATH#:}

# If Cygwin is detected, classpath is converted to Windows format.

(( CYGWIN )) && CLASSPATH=$(cygpath --path --mixed "${CLASSPATH}")

# Launch mode

if [ "x$DAEMON\_MODE" = "xtrue" ]; then

nohup $JAVA $KAFKA\_HEAP\_OPTS $KAFKA\_JVM\_PERFORMANCE\_OPTS $KAFKA\_GC\_LOG\_OPTS $KAFKA\_JMX\_OPTS $KAFKA\_LOG4J\_OPTS -cp $CLASSPATH $KAFKA\_OPTS "$@" > "$CONSOLE\_OUTPUT\_FILE" 2>&1 < /dev/null &

else

exec $JAVA $KAFKA\_HEAP\_OPTS $KAFKA\_JVM\_PERFORMANCE\_OPTS $KAFKA\_GC\_LOG\_OPTS $KAFKA\_JMX\_OPTS $KAFKA\_LOG4J\_OPTS -cp $CLASSPATH $KAFKA\_OPTS "$@"

fi

* 启动kafka，在当前目录下

docker-compose up -d

##### 注意点：

1、单机搭建zk集群需要创建网络

2、zk的版本要使用3.4的，3.5坑太多，另外需要挂载配置文件，并在配置文件中设置客户端端口

3、使kafka与zk在同一网络下，

4、映射kafka-run-class.sh文件，主要修改配置解决JMX\_PORT相关的报错

* 通过 [http://你的IP:9099](http://xn--ip-0p3cm89l:9099/) 访问kafka-manager，然后点击 Cluster 按钮， 在下拉框中选择 Add Cluster， 添加你的kafka集群接口，其中kafka的版本不用管他
* 然后查看该集群，能看到相关的zookeeper、topic等信息。
* 然后我们可以创建话题并查看

1、 通过docker exec -it 容器ID /bin/bash 进入容器

2、 cd /opt/kafka\_2.12-2.3.0/

3、 bin/kafka-topics.sh --create --zookeeper zoo1:2181 --replication-factor 3 --partitions 3 --topic test （测试数据记得删除）

4、 bin/kafka-topics.sh --list --zookeeper zoo1:2181 # 查看话题列表

* 话题创建成功后，就可以退出容器了，我们在宿主机上运行一个生产者和一个消费者

运行一个消息生产者

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

运行一个消息消费者

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning

此时我们在生产者中输入任何字符，都会及时的显示在消费者的页面中，并且kafka在zookeeper注册后，创建的话题会被zookeeper记录下来。

* 另外我们可以在zookeeper中查看话题是否创建成功

连接zk，并打开客户端

docker run -it --rm --link zoo1:zk1 --link zoo2:zk2 --link zoo3:zk3 --net zookeeper\_network zookeeper:3.4.14 zkCli.sh -server zk1:2181,zk2:2181,zk3:2181

// 执行完上面的命令后，会进去zk的客户端，我们可以进行一些操作，比如

执行 ls / 可以查看根节点信息，然后根据现实内容，查看下一级数据节点的信息

建议等下搭建完kafka集群并创建topic后通过 ls /brokers/topics 来查看topic列表（kafka在zookeeper注册后，创建的话题会被zookeeper记录下来）

* 至此，kafka + zookeeper 集群搭建完毕，也进行了topic的创建以及监听，记得删除测试数据。

### 下面讲解filebeat的部署在应用服务器采集应用或系统日志，此处我们以Nginx日志为例

#### 我们选择172.168.50.240 作为应用服务器，你可以将filebeat部署到任何可以与kafka集群通信的机器

* 老规矩，建立文件夹，存放yml以及filebeat的配置文件

mkdir -p /root/filebeat // 可以是任意目录

cd /root/filebeat

vim docker-compose.yml

* docker-compose.yml的内容如下：

version: '3'

services:

filebeat-240:

image: docker.elastic.co/beats/filebeat:7.1.1 # 镜像依然选择与elk的版本一致

container\_name: filebeat-240

restart: always

volumes:

- /home/wwwlogs:/home/wwwlogs # 此处是宿主机的Nginx日志目录（修改为你自己的目录）

# - /root/dockerfiles/nginx/logs:/logs/nginx/logs # 宿主机部署了nginx+php的 docker环境，此处为容器中Nginx映射到宿主机的日志目录

- /root/filebeat/filebeat.yml:/usr/share/filebeat/filebeat.yml # filebeat的配置文件

* 上面提到的filebeat.yml 配置文件如下:

filebeat.inputs: # inputs为复数，表名type可以有多个

- type: log # 输入类型

access:

enabled: true # 启用这个type配置

paths:

- /logs/nginx/logs/access.log # 监控nginx1 的access日志，建议采集日志时先采集单一nginx的access日志。

- /home/wwwlogs/access.log # 监控nginx2 的access日志（docker产生的日志）

- include\_lines: ['ERROR', 'WARN'] # 这样 FileBeat 就会收集包含 pro.ERROR、pro.WARN 的日志行。 PS：与 multiline 联合使用的时候，会针对合并后的记录再过滤。

close\_rename: true # 重命名后，文件将不会继续被采集信息

tail\_files: true # 配置为true时，filebeat将从新文件的最后位置开始读取，如果配合日志轮循使用，新文件的第一行将被跳过

multiline.pattern: '^\[.+\]' # 这三行表示，如果是多行日志，合并为一行

multiline.negate: true

multiline.match: "after"

fields: # 额外的字段

source: nginx-access-240 # 自定义source字段，用于es建议索引（字段名小写，我记得大写好像不行）

- type: log

access:

enabled: true

paths:

- /logs/nginx/logs/error.log # 监控nginx1 的error日志

- /home/wwwlogs/error.log # 监控nginx2 的error日志（docker产生的日志）

fields:

source: nginx-error-240

# 多行合并一行配置

multiline.pattern: '^\[.+\]'

multiline.negate: true

multiline.match: "after"

tail\_files: true

filebeat.config: # 这里是filebeat的各模块配置，我理解modules为inputs的一种新写法。

modules:

path: ${path.config}/modules.d/\*.yml # 进入容器查看这里有很多模块配置文件，Nginx,redis,apache什么的

reload.enabled: false # 设置为true来启用配置重载

reload.period: 10s # 检查路径下的文件更改的期间（多久检查一次）

# 在7.4版本中，自定义es的索引需要把ilm设置为false， 这里未验证，抄来的

setup.ilm.enabled: false

output.kafka: # 输出到kafka

enabled: true # 该output配置是否启用

hosts: ["172.168.50.41:9092", "172.168.50.41:9097", "172.168.50.41:9098"] # kafka节点列表

topic: "elk-%{[fields.source]}" # kafka会创建该topic，然后logstash(可以过滤修改)会传给es作为索引名称

partition.hash:

reachable\_only: true # 是否只发往可达分区

compression: gzip # 压缩

max\_message\_bytes: 1000000 # Event最大字节数。默认1000000。应小于等于kafka broker message.max.bytes值

required\_acks: 1 # kafka ack等级

worker: 3 # kafka output的最大并发数

# version: 0.10.1 # kafka版本

bulk\_max\_size: 2048 # 单次发往kafka的最大事件数

logging.to\_files: true # 输出所有日志到file，默认true， 达到日志文件大小限制时，日志文件会自动限制替换，详细配置：https://www.cnblogs.com/qinwengang/p/10982424.html

close\_older: 30m # 如果一个文件在某个时间段内没有发生过更新，则关闭监控的文件handle。默认1h

force\_close\_files: false # 这个选项关闭一个文件,当文件名称的变化。只在window建议为true

#没有新日志采集后多长时间关闭文件句柄，默认5分钟，设置成1分钟，加快文件句柄关闭；

#close\_inactive: 1m

#

##传输了3h后荏没有传输完成的话就强行关闭文件句柄，这个配置项是解决以上案例问题的key point；

#close\_timeout: 3h

#

###这个配置项也应该配置上，默认值是0表示不清理，不清理的意思是采集过的文件描述在registry文件里永不清理，在运行一段时间后，registry会变大，可能会带来问题。

#clean\_inactive: 72h

#

##设置了clean\_inactive后就需要设置ignore\_older，且要保证ignore\_older < clean\_inactive

#ignore\_older: 70h

#

## 限制 CPU和内存资源

#max\_procs: 1

#queue.mem.events: 256

#queue.mem.flush.min\_events: 128

* 配置完成后，docker-compose up -d 启动filebeat服务
* 访问你的nginx，看看access.log文件是否正常产生日志
* 通过kafka-manage查看是否接受到消息 <http://172.168.50.41:9099/> （部署kafka-manage服务的机器）
* kafka正常收到消息后，通过elasticsearch-head查看是否正常建立索引 <http://172.168.50.41:9201/> （部署elasticsearch-head服务的机器）
* 如kafka未正常收到消息，检查file的配置，如es未正常建立索引，检查logstash配置并查看日志
* 查看单个容器日志

docker logs --since 30m 容器ID

#### 以上就是日志采集与存储的过程，以Nginx的access.log日志（应用日志，接口日志也一样）为例，filebeat从服务器的日志文件采集到日志，发送给kafka，然后由logstash进行消费，再经由logstash过滤后输出到elasticsearch进行存储并建立索引。

知止而后有定，定而后能静，静而后能安，安而后能虑，虑而后能得。 所谓诚其意者，毋自欺也。