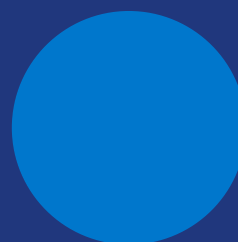
ES在derbysoft的优化实践

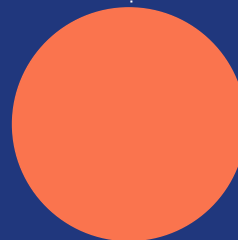
黄绍平，数据平台负责人
德比软件 derbysoft，2023/04/08

分享嘉宾

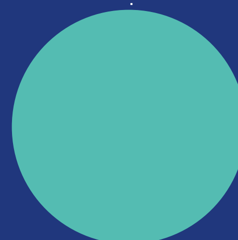
从2015年开始接触大数据相关技术，对Kafka，Hadoop，Elasticsearch 相关技术有多年经验，目前主要专注于基于AWS云的企业数据湖数据仓库平台建设



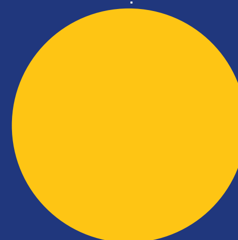
日志系统简介



写入ES (Kafka Connect)

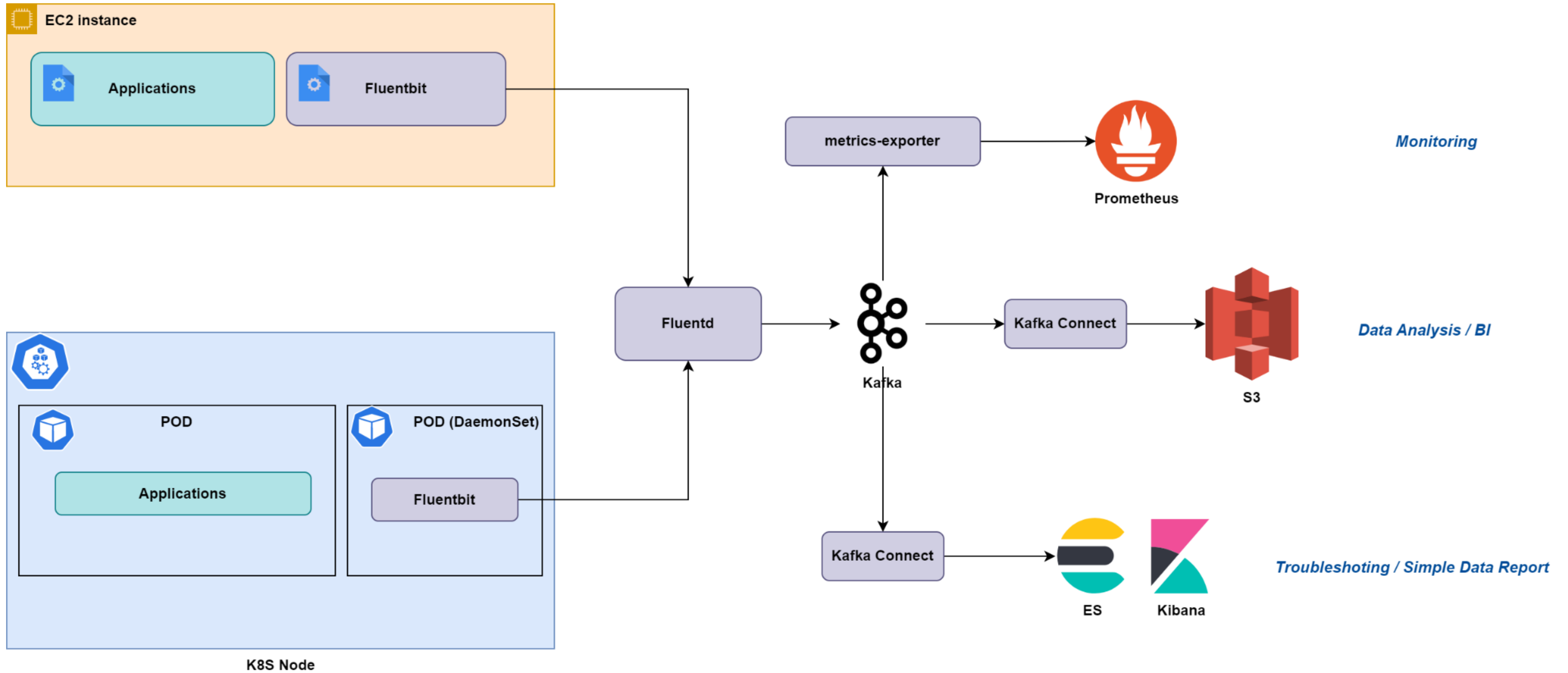


成本优化



其他优化

日志系统架构简介



日志系统简介

每日size：5TB

每日document数：120亿

Indexing rate: 12w/s

*按照写入ES后的单复本统计

ES版本为 6.8，以我个人的理解本分享的经验也完全适用7.x版本。

日志格式：

```
{
  "timestamp": "2023-03-29T11:39:58.943",
  "app_name": "abc-app",
  "host": "10.0.0.1",
  "region": "cn-north-1",
  "version": "v2",
  "process": "Book",
  "process_result": "Success",
  "process_duration": 32,
  ... // 基于不同事件会有其他字段
}
```

写入ES(Kafka Connect)

elastic

日志写入ES方案

早期我们完全自己实现了一套写入ES的组件，存在如下问题：

- 依赖MySQL实现任务分布式分发控制，过于复杂，并且不稳定
- 数据转换逻辑和整个工程耦合

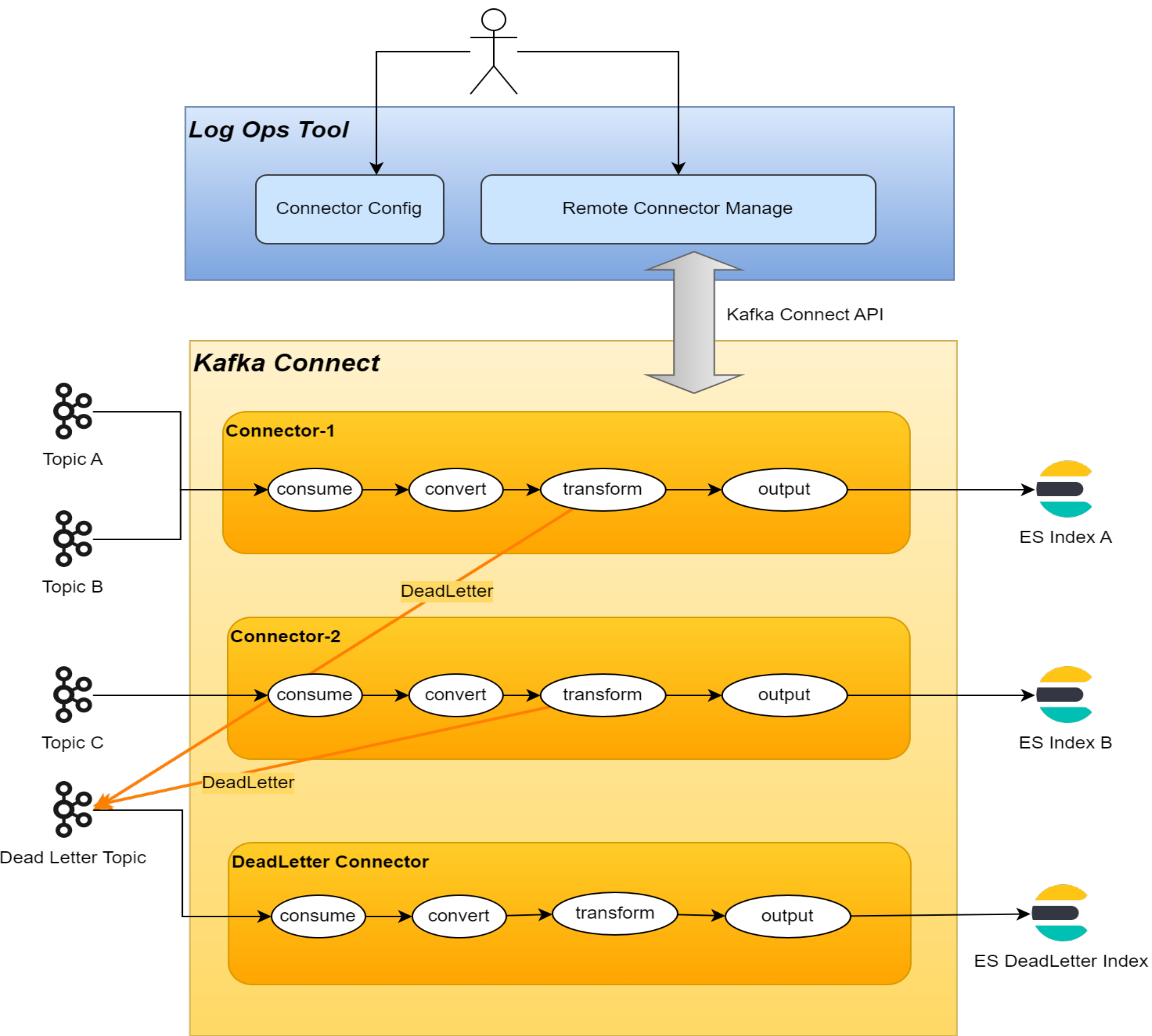
新方案改用**Kafka Connect**：

- 分布式任务控制交给Kafka Connect实现
- 数据转换逻辑解耦，仅需开发数据转换逻辑代码，以插件的形式部署
- 通过升级Kafka Connect，以及Connect ES Sink插件配合上下游系统升级

新方案需要满足的需求

- 实现kafka topic和ES index 多对多的映射
- 通过配置实现基于日期时间分索引，如按年，按月，按日
- 可实现自定义的数据转换逻辑
- 可实现数据字段格式类型校验，确保写入ES数据类型正确
- 异常消息进死信Topic，并写入ES，便于排查问题
- 数据写入任务的管理（创建，配置，启动，停止）

基于Kafka Connect 写入ES的架构



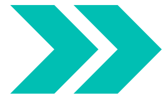
- Connector 本质上是一个Consumer Group, 来消费Topic数据，每个Connector对应写一个ES Index
- Transform即执行自定义的数据转换逻辑代码
- Kafka Connect 提供了Connector创建, 更新，暂停，删除，状态获取等HTTP API
- 死信Index，用于排查问题
- Log Ops Tool, 自研的一个配置和任务管理工具

实现 200+ Kafka Topic通过100+ Connector 将数据写入对应 100+ 索引.

Connector配置

```
{
  "topics": "topicA,topicB", # 指定消费数据的topic列表

  # 指定是一个ES的sink connector，以及ES对应的连接配置信息
  "connector.class": "io.confluent.connect.elasticsearch.ElasticsearchSinkConnector",
  "connection.url": "https://10.0.0.1:9200",

  # 指定定制的Transformation，功能包括：a) 完成定制数据转换逻辑; b) 执行数据类型的校验
  "transforms": "eventlog",
  "transforms.eventlog.type": "com.derbysoft.kafka.connect.transforms.EventLog",

  # 通过API获取日志schema的定义，作为数据类型校验的依据
  "transforms.eventlog.fields.whitelist.url": "http://10.0.0.2:8080/api/field/schema",

  # 自定义index名称，并基于dateFormat实现按日期时间划分索引
  "transforms.eventlog.index.pattern": "'myindex-'yyyyMMdd",

  # 开启死信队列，当Transformation过程中有格式异常或类型异常的消息进入死信队列kafka Topic
  "errors.deadletterqueue.topic.name": "dlq_kafka-connect-es",
  "errors.deadletterqueue.context.headers.enable": "true",
}
```

** 上述省略了部分必要配置*



死信消息入ES

	Field	Value
t	errors_class_name	com.derbysoft.kafka.connect.transforms.KFCPerfLog
t	errors_connector_name	es
t	errors_exception_class_name	org.apache.kafka.connect.errors.DataException
t	errors_exception_stacktrace	see below
#	errors_offset	3,510,502,410
t	errors_partition	2
t	errors_stage	TRANSFORMATION
t	errors_task_id	2
t	errors_topic	
t	reason	java.lang.NumberFormatException: For input string: "null"
t	record	see below

errors_exception_stacktrace:

```
org.apache.kafka.connect.errors.DataException: java.lang.NumberFormatException: For input string: "null"
        at com.derbysoft.kafka.connect.transforms.KFCPerfLog.valueProcessPerfV2(KFCPerfLog.java:166)
        at com.derbysoft.kafka.connect.transforms.KFCPerfLog.valueProcess(KFCPerfLog.java:173)
        at com.derbysoft.kafka.connect.transforms.KFCCommonLog.applyRecord(KFCCommonLog.java:93)
        at com.derbysoft.kafka.connect.transforms.KFCCommonLog.apply(KFCCommonLog.java:73)
        at org.apache.kafka.connect.runtime.TransformationChain.lambda$apply$0(TransformationChain.java:50)
        at org.apache.kafka.connect.runtime.errors.RetryWithToleranceOperator.execAndRetry(RetryWithToleranceOperatc
        at org.apache.kafka.connect.runtime.errors.RetryWithToleranceOperator.execAndHandleError(RetryWithToleranceC
        at org.apache.kafka.connect.runtime.errors.RetryWithToleranceOperator.execute(RetryWithToleranceOperator.jav
        at org.apache.kafka.connect.runtime.TransformationChain.apply(TransformationChain.java:50)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.convertAndTransformRecord(WorkerSinkTask.java:507)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.convertMessages(WorkerSinkTask.java:465)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.poll(WorkerSinkTask.java:321)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.iteration(WorkerSinkTask.java:224)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.execute(WorkerSinkTask.java:192)
        at org.apache.kafka.connect.runtime.WorkerTask.doRun(WorkerTask.java:177)
        at org.apache.kafka.connect.runtime.WorkerTask.run(WorkerTask.java:227)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:750)
Caused by: java.lang.NumberFormatException: For input string: "null"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.valueOf(Integer.java:766)
        at com.derbysoft.kafka.connect.perflog.PerfLogField$Type.convert(PerfLogField.java:40)
        at com.derbysoft.kafka.connect.transforms.KFCPerfLog.valueProcessPerfV2(KFCPerfLog.java:159)
        ... 20 more
```

record:

{"pod_name":" ","pod_ip":" "app_name":"
raw","perf_version":"v2","timestamp":"2023-04-03T15:56:53.980","channel": ,"echo_token":
6","supplier": ,"check_in":"2023-07-27","check_out":"2023-08-03","children_cnt":"null","children_ages":"null"
ity_result":"Avail","process":"GetAvailability","process_result":"Success","request_type": "process_dura

自研运维工具

Kafka Manager | Connect URL | Connector Common Config | Connector | Connector Remote | Connector Remote Log

batch create 0 | batch delete 0

storage | group search | status search

es | es-perf | running
es | es-perf | running
es | es-perf | running

es-

type: sink | status: running

create | update | pause | resume | restart | delete

Tasks | Config | Topics | Rest Log

090
0 2 6

9090
1 3 7

9090
4 8

9090
5 9

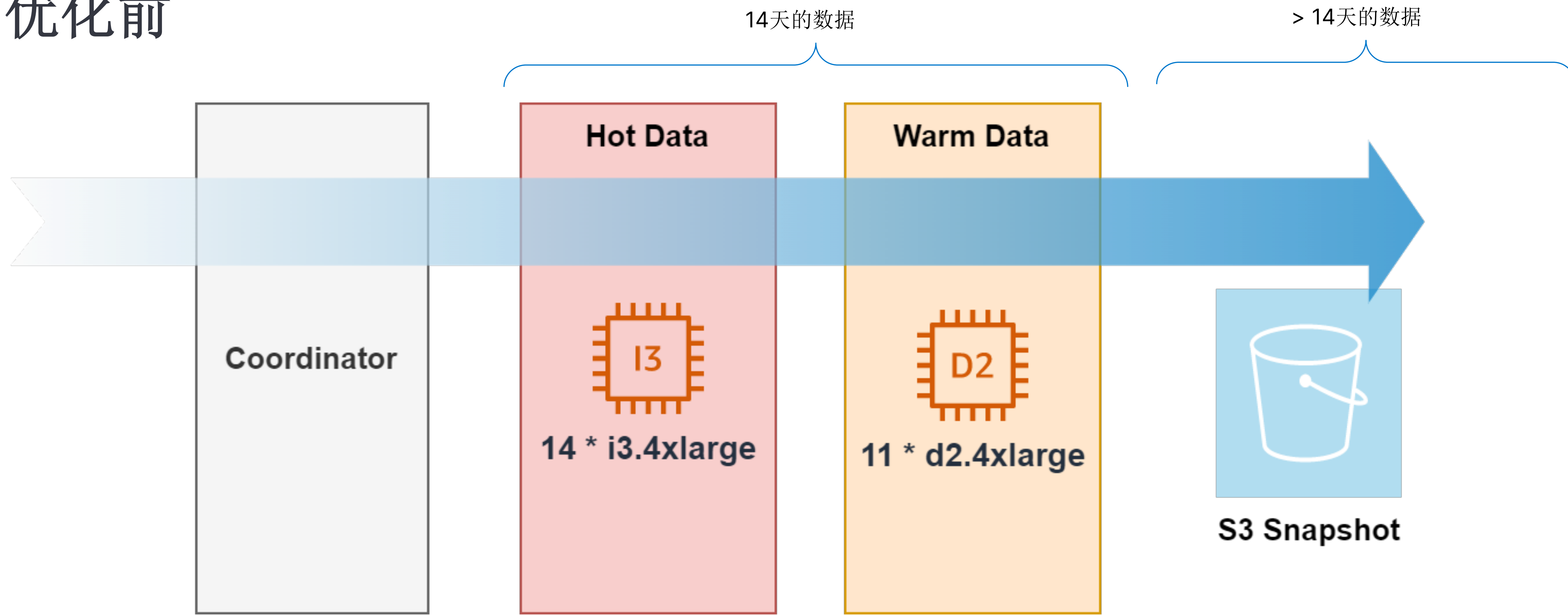
优势

- 数据的消费和写入ES交由kafka connect和ES connector插件完成，仅需开发Transformation插件，减少开发工作量
- 结合自己开发的运维工具大大简化运维配置工作
- 通过将死信写入ES，提高排查问题效率

成本优化

elastic

优化前



型号	vCPU	内存 (GiB)	实例存储 (GB)	按需每小时费率	数量	每小时成本
i3.4xlarge	16	122	**2 个 1900 NVMe SSD**	1.248 USD	14	17.472
d2.4xlarge	16	122	**12 个 2000 HDD**	2.76 USD	11	30.36

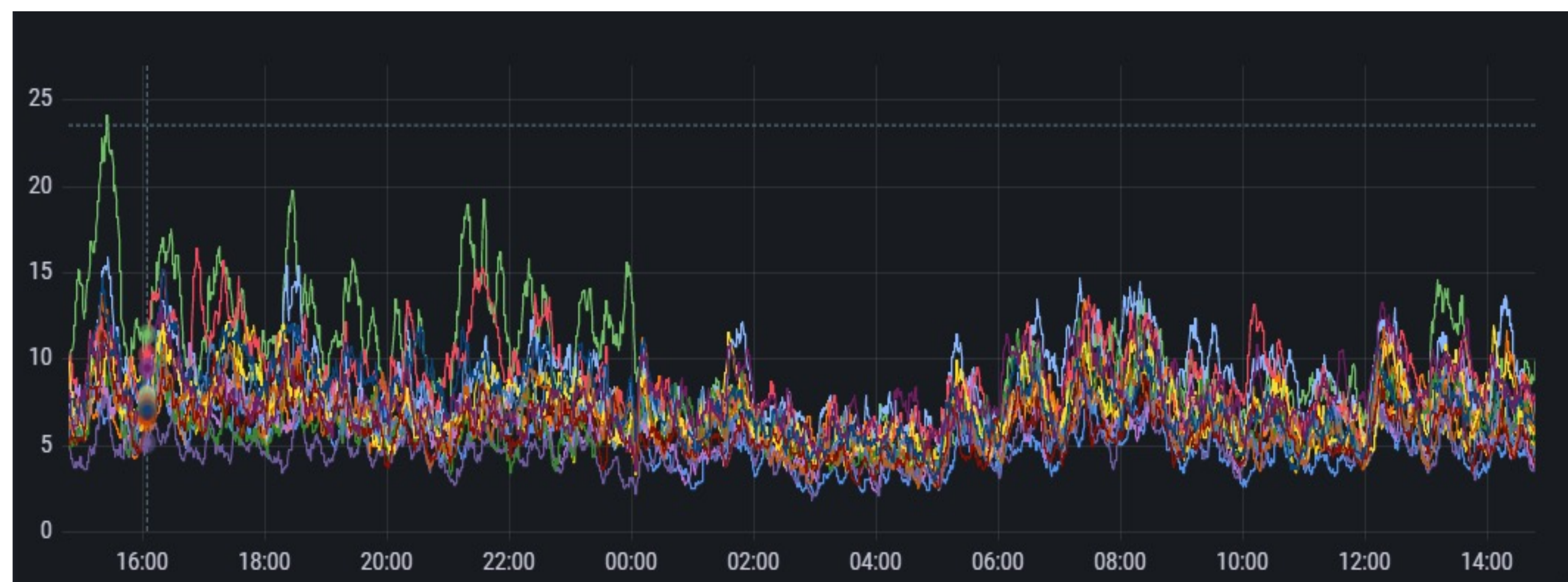
基于AWS EC2 安装部署

优化前的问题

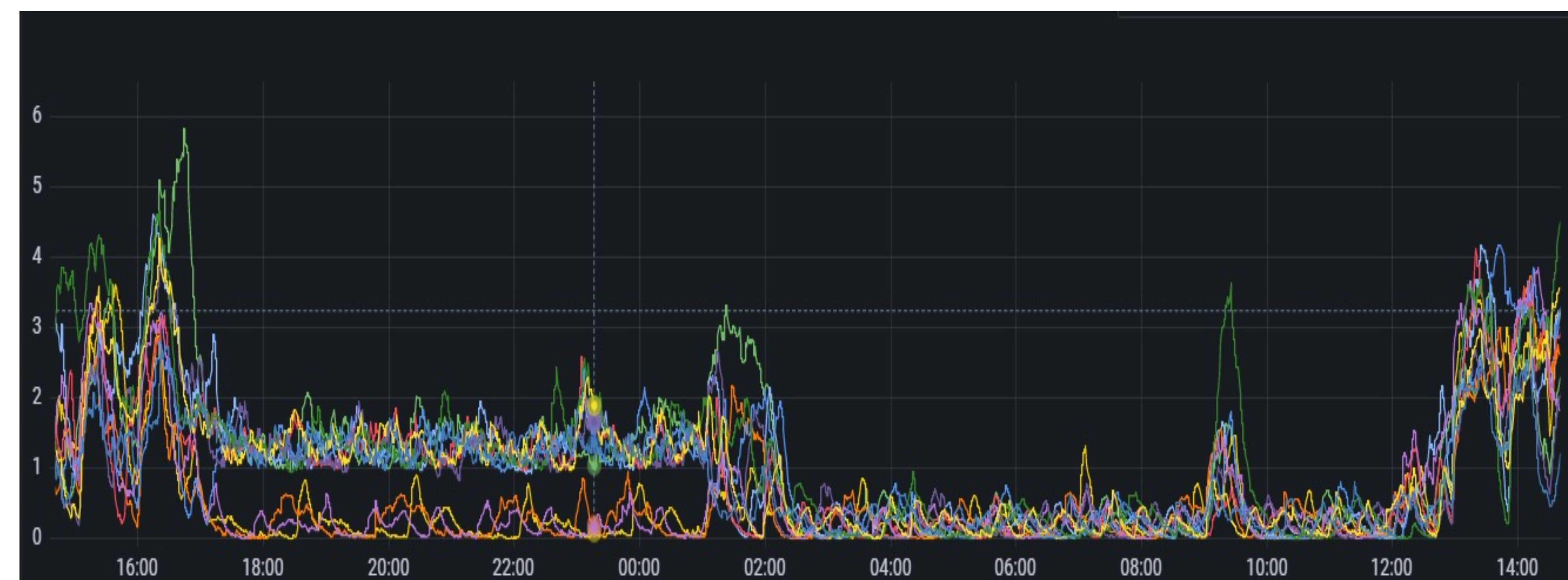
1. Warm节点资源利用率非常低
2. Hot to Warm 每天都会导致几个TB数据的移动，耗费大量资源
3. Hot节点负载高，负载不均衡

实时的写入负载在Hot节点上，近期数据被读的更频繁，所以大部分的读负载也在Hot节点上。这是导致Warm节点资源利用率低，以及Hot节点负载高的主要原因。

Hot 节点CPU Load (一天)



Warm 节点CPU Load (一天)



Warm节点资源利用率低问题解决

思路一，给Warm节点降级，选用更少CPU的机型，一方面我们使用了EC2的**实例存储**，降级就意味着磁盘容量的减少，容量减少不可接受。当然也可以选择不用实例存储，而是采用通用机型加EBS，但是成本会高出不少，而且没有实例存储性能好。另外为了支持Hot to Warm, 仍然需要预置足够的资源，当非Hot to Warm的时段，资源利用率还是会很低。

思路二，去掉Hot to Warm，因为Warm的读取请求本来就比较少，为了这些少量的读请求去移动大量数据是不经济的。这样只有一个分层，所有写入负载和读取负载都落到这一层。**一方面减少了移动数据带来的资源损耗，另一方面消除了资源利用率低的问题。**

结论：采用去掉Hot to Warm方案

机型的选择

		Total CPU cores	Total Mem GB	Total Disk (TB)	Cost saving
现状	i3.4xlarge (hot) d2.4xlarge (warm)	hot: 224 warm: 176 total: 400	hot: 1,792 warm: 1,342 total: 3,134	hot(SSD): 52 warm(HDD): 253 total: 305	
方案A	is4gen.4xlarge	256	1,536	(SSD) 240	-22%
方案B	im4gn.4xlarge	400	1600	(SSD) 187.5	-23%
方案C	d3.4xlarge	256	2,048	(HDD) 384	-33%

从存储空间上来看方案B刚好够用，但是无法应对未来数据的增长，并且CPU资源过剩。从CPU资源来看，方案A, C也都是够用的，所以综合来看方案C是最经济，同时又能得到更充足资源的。唯一的问题是磁盘的类型是HDD。

HDD如何玩

d3.4xlarge   12 x 2TB HDD

虽然是HDD，但其实是12块盘组成的，所以理论上可以通过RAID0 使聚合磁盘吞吐达到2+GB/s.

机型	BS	R/W	IOPS	吞吐量 (MB/s)
d3.4xlarge (HDD)	16k	随机读	4323	69
		随机写	4689	75
		顺序读	3525	450
		顺序写	128897	2014.2
i3.4xlarge (SSD)	16k	随机读	157509	2461.9
		随机写	76951	1202.4
		顺序读	194000	3031.3
		顺序写	91074	1423.4

d3.4xlarge (HDD) 加入生产集群做验证

和i3.4xlarge (SSD) 做对比

发现如下问题：

- CPU Load非常高
- Processes blocked指标比较高
- 系统调用fdatasync 时间占比比较高

% time	seconds	usecs/call	calls	errors	syscall
55.25	670.980206	128960	5203		epoll_wait
31.65	384.390424	10657	36068	5498	futex
5.76	69.998450	29560	2368		fdatasync
5.23	63.463618	961569	66	27	restart_syscall
1.14	13.852722	133	104096		write
0.39	4.731353	876	5397		read
0.24	2.855333	3587	796		writev

index.translog.durability参数

`index.translog.durability`

Whether or not to `fsync` and commit the translog after every index, delete, update, or bulk request. This setting accepts the following parameters:

`request`

(default) `fsync` and commit after every request. In the event of hardware failure, all acknowledged writes will already have been committed to disk.

`async`

`fsync` and commit in the background every `sync_interval`. In the event of a failure, all acknowledged writes since the last automatic commit will be discarded.

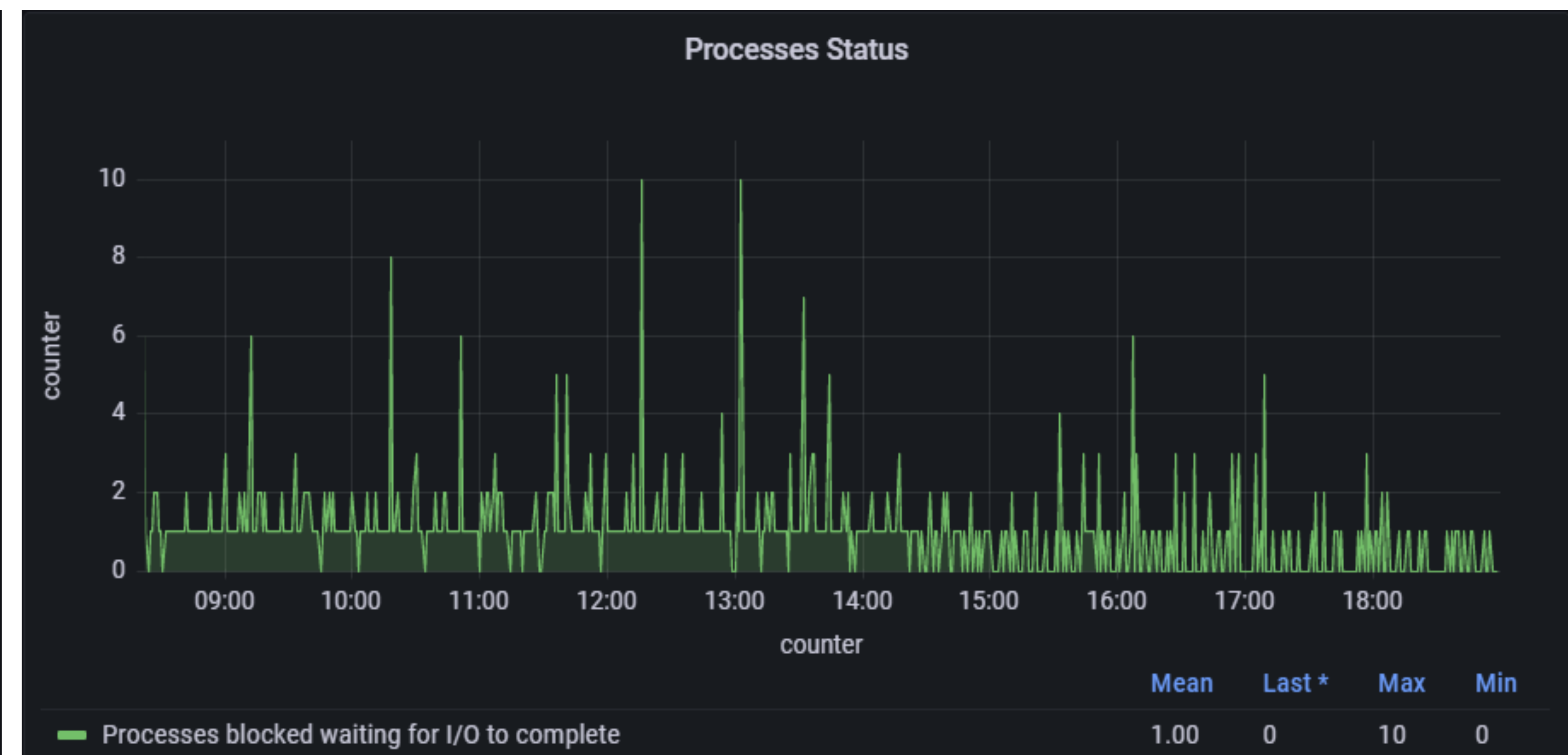
`index.translog.sync_interval`

How often the translog is `fsync`ed to disk and committed, regardless of write operations. Defaults to `5s`. Values less than `100ms` are not allowed.

调整index.translog.durability 为async

- CPU Load大幅下降
- Processed Blocked明显改善
- fdatasync系统调用时间占比非常小，0.x%

虽然数据可靠性有略微降低，但是对于日志系统来说，完全可以接受。



总结

执行操作：

- 去掉Hot to Warm，只有一层数据节点
- 将原有的节点类型(14台i3.4xlarge + 11台 d2.4xlarge) 替换为18台d3.4xlarge
- 对12块HDD盘做RAID0
- 将所有日志索引参数index.translog.durability调整为async

收益：

- 硬件成本下降20+%
- 节点数减少7台，减少相应的License成本
- 架构更简单，运维更简单
- 整体磁盘容量提升40%
- 更多的节点来承担写入负载（因为写入负载远大于读负载）

其他优化

- 写负载不均衡
- 大量小索引

elastic

写负载不均衡

虽然已经完成节点类型的替换，并且承载写负载的机器数量增加到18台，但是各机器上的负载非常不均衡，这其实也会造成资源的浪费。

虽然集群级别的balance会确保shard数在所有node中尽可能的均衡，但是因为同时存在大索引和小索引，可能会出现大索引的shard集中都分配到某几个节点，导致某几个节点过热。

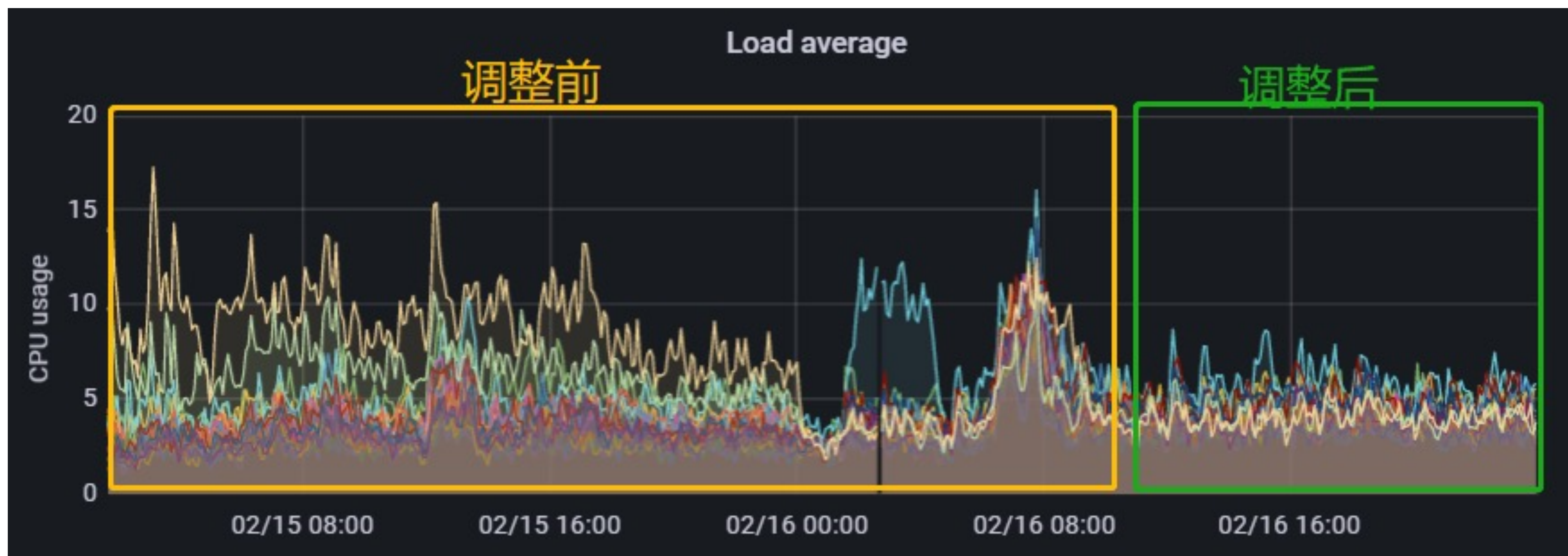
调整参数

通过设置index.routing.allocation.total_shards_per_node参数为2，确保大索引shard分散开，不会集中到个别node上。

官方文档解释：

`index.routing.allocation.total_shards_per_node`

The maximum number of shards (replicas and primaries) that will be allocated to a single node. Defaults to unbounded.



小索引问题

小索引即数据吞吐比较小的索引，因为我们是基于业务主题来分索引，某些业务索引每天数据量可能都不到1GB. 一个shard底层为一个lucene索引，会消耗一定文件句柄，内存，cpu等。

来自官方的提示：

- *分片过小会导致段过小，进而致使开销增加。您要尽量将分片的平均大小控制在至少几 GB 到几十 GB 之间。对时序型数据用例而言，分片大小通常介于 20GB 至 40GB 之间。*
- *每个节点上可以存储的分片数量与可用的堆内存大小成正比关系，但是 Elasticsearch 并未强制规定固定限值。这里有一个很好的经验法则：确保对于节点上已配置的每个 GB，将分片数量保持在 20 以下。如果某个节点拥有 30GB 的堆内存，那其最多可有 600 个分片，但是在此限值范围内，您设置的分片数量越少，效果就越好*

通过rollover机制解决小索引问题

按天分索引的方式改为基于rollover的机制按大小和时间自动分索引。

比如, 原来每天1GB数据，1个索引，1个shard，14天总共14个shard，每个shard 1GB。

改为rollover机制后，设定max size为40GB，max age为14天。则总共只会产生1个索引1个shard。大大减少了shard的数量。

最终优化总结

elastic

1. 通过Kafka Connect实现数据写入ES, 结合自研的运维工具，提升了开发运维排错的工作效率
2. 通过去掉Hot-Warm机制，以及更换节点类型实现成本的降低和资源利用率的上升，
（涉及参数调整：index.translog.durability）
1. 通过参数的调整解决负载不均衡问题
（涉及参数调整：index.routing.allocation.total_shards_per_node）
1. 通过Rollover机制解决小索引shard过多问题

感谢观看

专业、垂直、纯粹的 Elastic 开源技术交流社区

https://elasticsearch.cn/