

Elasticsearch 八大经典应用

搜索&推荐技术应用系列



- 对垒8大主流数据产品，孰优孰劣？
- 人脸识别、地理位置分析如何轻松实现？
- PB级数据如何保证毫秒检索、秒级分析？





钉钉扫码加入
“Elasticsearch中文技术社区”
交流群



钉钉扫码加入
“推荐与搜索技术交流群”



微信扫码关注
“搜索与推荐技术”
公众号



阿里云开发者“藏经阁”
海量免费电子书下载

目录

Elasticsearch对垒8大产品技术，孰优孰劣？	04
ES既是搜索引擎又是数据库？真的有那么全能吗？	15
DB 与 Elasticsearch 混合之应用系统场景分析探讨	26
初次使用 Elasticsearch 遇多种分词难题？那是你没掌握这些原理	34
Transforms数据透视让Elasticsearch数据更易分析	45
Observability：使用 Elastic Stack 分析地理空间数据	59
阿里云 Elasticsearch 向量检索，轻松玩转29个业务场景	79
阿里云 Elasticsearch 索引数据生命周期管理	86
PB级数据量背后阿里云 Elasticsearch 的内核优化实践	96
一次有趣的Elasticsearch+矩阵变换聚合实践	108

Elasticsearch对垒8大产品技术，孰优孰劣？

简介：简要用Elasticsearch与其它8种数据产品做了个对比，基于很多业务场景对比，代表了笔者对于Elasticsearch优胜劣汰的看法

作者介绍：

李猛(ynuosoft)，Elastic-stack产品深度用户，ES认证工程师，2012年接触Elasticsearch，对Elastic-Stack开发、架构、运维等方面有深入体验，实践过多种Elasticsearch项目，最暴力的大数据分析应用，最复杂的业务系统应用；业余为企业提供Elastic-stack咨询培训以及调优实施。

序言

355 systems in ranking, April 2020

Rank			DBMS	Database Model	Score		
Apr 2020	Mar 2020	Apr 2019			Apr 2020	Mar 2020	Apr 2019
1.	1.	1.	Oracle	Relational, Multi-model	1345.42	+4.78	+65.48
2.	2.	2.	MySQL	Relational, Multi-model	1268.35	+8.62	+53.21
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1083.43	-14.43	+23.47
4.	4.	4.	PostgreSQL	Relational, Multi-model	509.86	-4.06	+31.14
5.	5.	5.	MongoDB	Document, Multi-model	438.43	+0.82	+36.45
6.	6.	6.	IBM Db2	Relational, Multi-model	165.63	+3.07	-10.42
7.	7.	8.	Elasticsearch	Search engine, Multi-model	148.91	-0.26	+2.91
8.	8.	7.	Redis	Key-value, Multi-model	144.81	-2.77	-1.57
9.	10.	10.	SQLite	Relational	122.19	+0.24	-2.02
10.	9.	9.	Microsoft Access	Relational	121.92	-3.22	-22.73

Elasticsearch当前热度排名很高

青出于蓝，而胜于蓝。

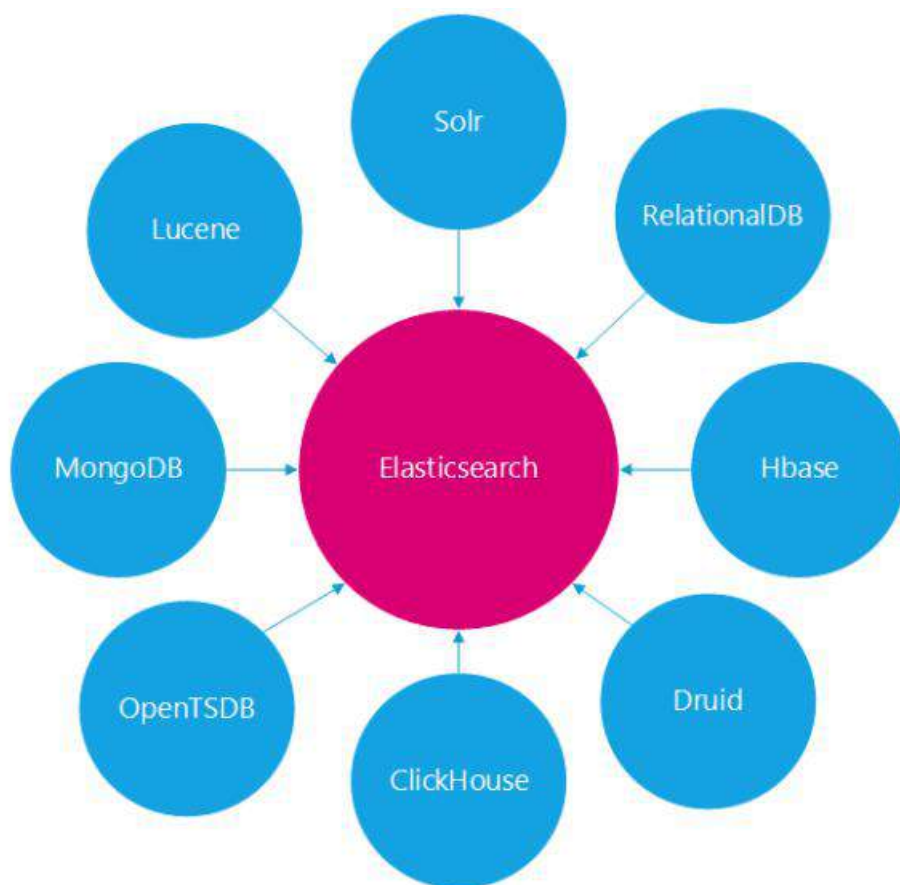
入行 Elastic-Stack 技术栈很久很久，为了免于知识匮乏眼光局限，有必要到外面的世界看看，丰富自己的世界观。本篇内容从 Elastic 的竞争产品角度分析探讨。

- 哪些应用场景下使用 Elasticsearch 最佳？
- 哪些应用场景下不使用 Elasticsearch 最好？

本文仅代表个人的观点，不代表社区技术阵营观点，无意口水之争，限于本人的经验知识有限，可能与读者观点认知不一致。

竞争产品

Elasticsearch 从做搜索引擎开始，到现在主攻大数据分析领域，逐步进化成了一个全能型的数据产品，在 Elasticsearch 诸多优秀的功能中，与很多数据产品有越来越多的交叉竞争，有的功能很有特色，有的功能只是附带，了解这些产品特点有助于更好的应用于业务需求。



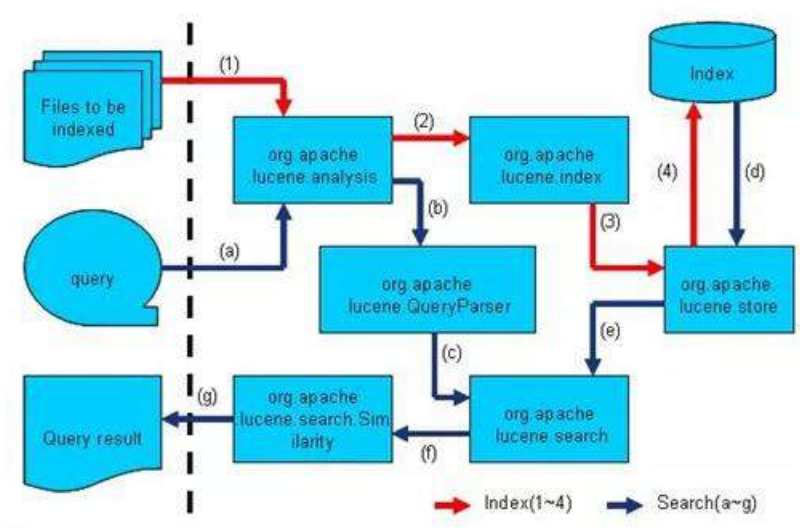
图片：Elasticsearch竞争图谱示意图

1、Lucene

Lucene 是一个搜索的核心库，Elastic 也是在 Lucene 基础之上构建，它们之间的竞争关系是由 Lucene 本身决定的。

在互联网 2.0 时代，考验各互联网公司最简单的技术要求，就是看他们的搜索做的怎么样，那时大家的做法几乎一样，都基于 Lucene 核心库构建一套搜索引擎，剩下的就看各公司的开发者们的水平。笔者有幸在 2012 年之前，基于 Lucene 做过垂直行业的搜索引擎，遇到很多问题有必要说一下：

- 项目基于 Lucene 包装，业务代码与核心库一起构建发布，代码耦合度很高，每次有数据字段变更，都需要重新编译打包发布，这个过程非常的繁琐，且相当危险。
- 程序重新发布，需要关闭原有的程序，涉及到进程切换问题。
- 索引数据定期全量重新生成，也涉及到新旧索引切换，索引实时刷新等问题，都需要设计一套复杂的程序机制保障。
- 每个独立业务线需求，都需要单独构建一个 Lucene 索引进程，业务线多了之后，管理是个麻烦的事情。
- 当单个 Lucene 索引数据超过单实例限制之后，需要做分布式，这个原有 Lucene 是没有办法的，所以常规的做法也是按照某特定分类，拆分成多个索引进程，客户端查询时带上特定分类，后端根据特定分类路由到具体的索引。
- Lucene 库本身的掌控难度，对于功力尚浅的开发工程师，需要考虑的因素实在太多了，稍微不慎，就会出现很大的程序问题。



图示：Lucene内部索引构建与查询过程

Elasticsearch 与 Lucene 核心库竞争的优势在于：

- 完美封装了 Lucene 核心库，设计了友好的 Restful-API，开发者无需过多关注底层机制，直接开箱即用。
- 分片与副本机制，直接解决了集群下性能与高可用问题。

Elastic 近年的快速发展，市面上已经很少发现基于 Lucene 构建搜索引擎的项目，几乎清一色选择 Elasticsearch 作为基础数据库服务，由于其开源特性，广大云厂商也在此基础上定制开发，与自己的云平台深度集成，但也没有独自发展一个分支。

本次的竞争中，**Elasticsearch 完胜。**

2、Solr

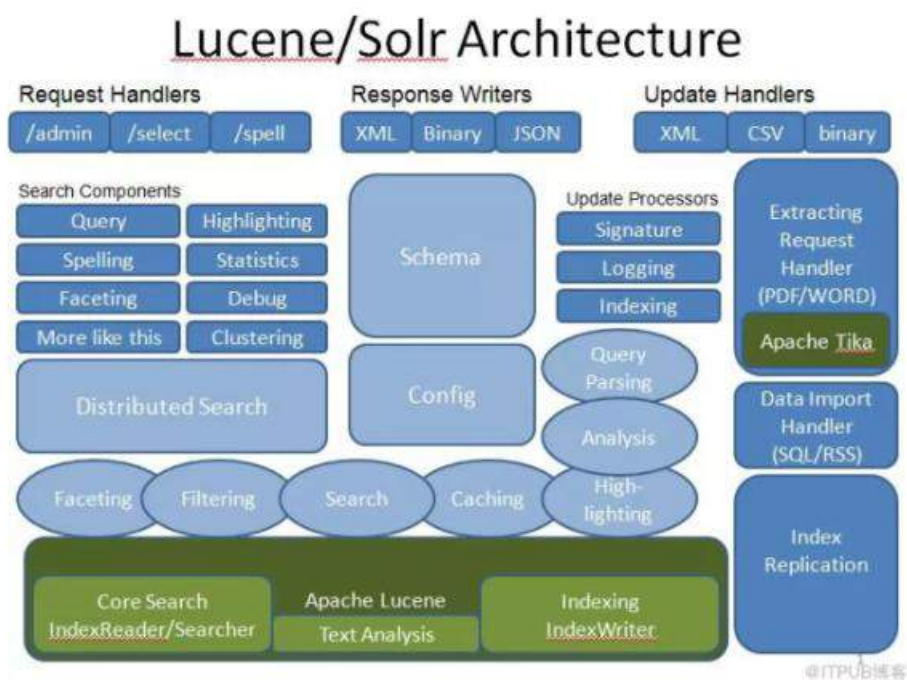
Solr 是第一个基于 Lucene 核心库功能完备的搜索引擎产品，诞生远早于 Elasticsearch，早期在全文搜索领域，Solr 有非常大的优势，几乎完全压倒 Elastic，在近几年大数据发展时代，Elastic 由于其分布式特性，满足了很多大数据的处理需求，特别是后面 ELK 这个概念的流行，几乎完全忘记了 Solr 的存在，虽然也推出了 Solr-Coud 分布式产品，但已经基本无优势。

接触过几个数据类公司，全文搜索都基于 Solr 构建，且是单节点模式，偶然出现一些问题，找咨询顾问排查问题，人员难找，后面都迁移到 Elasticsearch 之上。

现在市面上几乎大小公司都在使用 Elasticsearch，除了老旧系统有的基于 Solr 的，新系统项目应该全部是 Elasticsearch。

个人认为有以下几个原因：

- ES 比 Solr 更加友好简洁，门槛更低。
- ES 比 Solr 产品功能特点更加丰富，分片机制，数据分析能力。
- ES 生态发展，Elastic-stack 整个技术栈相当全，与各种数据系统都很容易集成。
- ES 社区发展更加活跃，Solr 几乎没有专门的技术分析大会。



图示：Solr产品功能模块内部架构图

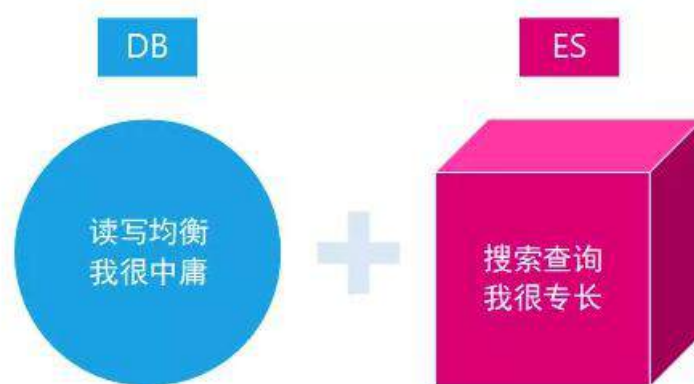
本次竞争中，Elasticsearch 完胜。

3、RDBMS

关系型数据库与 Elasticsearch 相比主要优点是事务隔离机制无可替代，但其局限性很明显，如下：

- 关系型数据库查询性能，数据量超过百万级千万级之后下降厉害，本质是索引的算法效率不行，B+ 树算法不如倒排索引算法高效。
- 关系型数据库索引最左原则限制，查询条件字段不能任意组合，否则索引失效，相反 Elasticsearch 可以任意组合，此场景在数据表关联查询时特别明显，Elasticsearch 可以采用大宽表解决，而关系型数据库不能。
- 关系型数据库分库分表之后多条件查询，难于实现，Elasticsearch 天然分布式设计，多个索引多个分片皆可联合查询。
- 关系型数据库聚合性能低下，数据量稍微多点，查询列基数多一点性能下降很快，Elasticsearch 在聚合上采用的是列式存储，效率极高。
- 关系型数据库侧重均衡性，Elasticsearch 侧重专一查询速度。

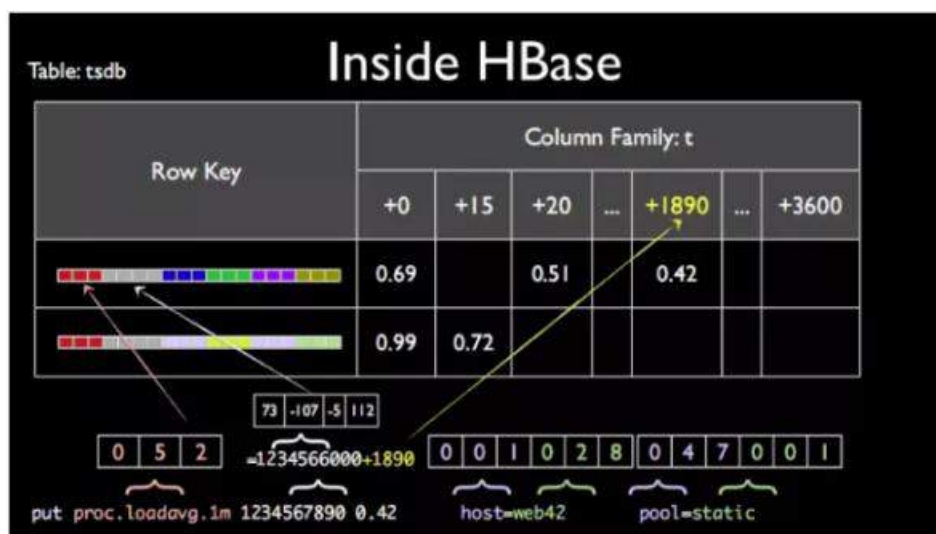
若数据无需严格事务机制隔离，个人认为都可以采用 Elasticsearch 替代。若数据既要事务隔离，也要查询性能，可以采用 DB 与 ES 混合实现，详细见笔者的博客文章《[DB 与 ES 混合应用之数据实时同步](#)》



图示：RDBMS与ES各自优势示意图

4、OpenTSDB

OpenTSDB 内部基于 HBase 实现，属于时间序列数据库，主要针对具有时间特性和需求的数据，进行过数据结构的优化和处理，从而适合存储具有时间特性的数据，如监控数据、温度变化数据等，小米公司开源监控体系 open-falcon 的就是基于 OpenTSDB 实现。



图示：OpenTSDB时间序列数据库内部实现

Elastic 产品本身无意时间序列这个领域，随着 ELK 的流行，很多公司采用 ELK 来构建监控体系，虽然在数值类型上不像时间序列数据库做过特别处理，但由于其便利的使用，以及生态技术栈的优势，我们也接受了这样的事实。

Elasticsearch 构建时间序列很简单，性能也相当不错：

- 索引创建规则，可以按年、按月、按周、按星期、按天、按小时等都创建索引，非常便利。
- 数据填充方面，定制一个时间字段做区分排序，其余的字段无需。
- 数据查询方面，除了按实际序列查询外，还可以有更多的搜索条件。

除非对于时间序列数据有非常苛刻的监控需求，否则选择 Elasticsearch 会更加合适一些。

5、HBase

HBase 是列式数据库的代表，其内部有几个致命设计大大限制了它的应用范围：

- 访问 HBase 数据只能基于 Rowkey，Rowkey 设计的好坏直接决定了 HBase 使用优劣。
- 本身不支持二级索引，若要实现，则需要引入第三方。

关于其各种技术原理就不多说了，说说它的一些使用情况。

公司所属物流速运行业，一个与车辆有关的项目，记录所有车辆行驶轨迹，车载设备会定时上报车子的轨迹信息，后端数据存储基于 HBase，数据量在几十 TB 级以上，由于业务端需要依据车辆轨迹信息计算它的公里油耗以及相关成本，所以要按查询条件批量查询数据，查询条件有一些非 rowkey 的字段，如时间范围，车票号，城市编号等，这几乎无法实现，原来暴力的做过，性能问题堪忧。此项目的问题首先也在于 rowkey 难设计满足查询条件的需求，其次是二级索引问题，查询的条件很多。

如果用列式数据库仅限于 Rowkey 访问场景，其实采用 Elastic 也可以，只要设计好 _id，与 HBase 可以达到相同的效果。

如果用列式数据库查询还需要引入三方组件，那还不如直接在 Elasticsearch 上构建更直接。

除非对使用列式数据库有非常苛刻的要求，否则Elasticsearch更具备通用性，业务需求场景适用性更多。



图示：列式数据库内部数据结构示意图

6、MongoDB

MongoDB 是文档型数据库的代表，数据模型基于 Bson，而 Elasticsearch 的文档数据模型是 Json，Bson 本质是 Json 的一种扩展，可以相互直接转换，且它们的数据模式都是可以自由扩展的，基本无限制。MongoDB 本身定位与关系型数据库竞争，支持严格的事务隔离机制，在这个层面实际上与 Elasticsearch 产品定位不一样，但实际工作中，几乎没有公司会将核心业务数据放在 MongoDB 上，关系型数据库依然是第一选择。若

超出这个定位，则 Elasticsearch 相比 MongoDB 有如下优点：

- 文档查询性能，倒排索引 / KDB-Tree 比 B+Tree 厉害。
- 数据的聚合分析能力，ES本身提供了列式数据 doc_value，比 Mongo 的行式要快不少。
- 集群分片副本机制，ES架构设计更胜一筹。
- ES 特色功能比 MongoDB 提供的更多，适用的场景范围更宽泛。
- 文档数据样例，ObjectId 由 MongoDB 内置自动生成。

```
{
  "_id": "ObjectId('52ffc4a5d85242602e000000')",
  "building": "22 A, Indiana Apt",
  "pincode": 123456,
  "city": "Los Angeles",
  "state": "California"
}
```



公司刚好有个项目，原来数据层基于 MongoDB 设计构建的，查询问题不少，后面成功迁移到 Elasticsearch 平台上，服务器数据量从15台降低到3台，查询性能还大幅度提升十倍，详细可阅读笔者另一篇文章《[从 MongoDB 迁移到 ES 后，我们减少了80%的服务器](#)》

抛开数据事务隔离，Elasticsearch 可以完全替代 MongoDB。

7、Clickhouse

ClickHouse 是一款 MPP 查询分析型数据库，近几年活跃度很高，很多头部公司都引入其中。我们为什么要引入呢，原因可能跟其他头部公司不太一样，如下：

- 笔者长期从事大数据工作，经常会碰到数据聚合的实时查询需求，早期我们会选择一款关系型数据库来做聚合查询，如 MySQL/PostgreSQL，稍微不注意就容易出现性能瓶颈。
- 后面引入 Elasticsearch 产品，其基于列式设计以及分片架构，性能各方面确实明显优于单节点的关系型数据库。
- Elasticsearch 局限性也很明显，一是数据量超过千万或者亿级时，若聚合的列数太多，性能也到达瓶颈；二是不支持深度二次聚合，导致一些复杂的聚合需求，需要人工编写代码在外部实现，这又增加很多开发工作量。
- 后面引入了 ClickHouse，替代 Elasticsearch 做深度聚合需求，性能表现不错，在数据量千万级亿级表现很好，且资源消耗相比之前降低不少，同样的服务器资源可以承担更多的业务需求。

ClickHouse 与 Elasticsearch 一样，都采用列式存储结构，都支持副本分片，不同的是 ClickHouse 底层有一些独特的实现，如下：

- MergeTree 合并树表引擎，提供了数据分区、一级索引、二级索引。
- Vector Engine 向量引擎，数据不仅仅按列存储，同时还按向量(列的一部分)进行处理，这样可以更加高效地使用 CPU



图示：ClickHouse在大数据平台中的位置

8、Druid

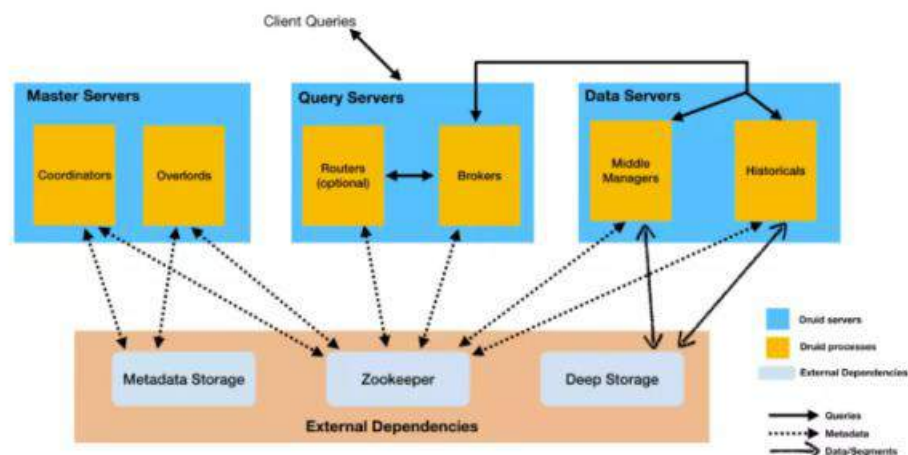
Druid 是一个大数据 MPP 查询型数据产品，核心功能 Rollup，所有的需要 Rollup 原始数据必须带有时间序列字段。Elasticsearch 在 6.3.X 版本之后推出了此功能，此时两者产品形成竞争关系，谁高谁下，看应用场景需求。

Druid 样本数据，必须带有 time 时间字段。

```
{
  "time": "2015-09-12T00:46:58.771Z",
  "channel": "#en.wikipedia",
  "cityName": null,
  "comment": "added project",
  "countryIsoCode": null,
  "countryName": null,
  "isAnonymous": false,
  "isMinor": false,
  "isNew": false,
  "isRobot": false,
  "isUnpatrolled": false,
  "metroCode": null,
  "namespace": "Talk",
  "page": "Talk:Oswald Tilghman",
  "regionIsoCode": null,
  "regionName": null,
  "user": "GELongstreet",
  "delta": 36,
  "added": 36,
  "deleted": 0
}
```

笔者之前负责过公司所有 Elasticsearch 技术栈相关数据项目，当时也有碰到一些实时聚合查询返回部分数据的需求，但我们的需求不太一样，索引数据属于离线型更新，每天都会全部删除并重新创建索引插入数据，此时使用 Elastic 的版本是 6.8.X，仅支持离线型数据 Rollup，所以此功能没用上，Elastic 在 7.2.X 版本之后才推出实时 Rollup 功能。

- Druid 更加专注，产品设计围绕 Rollup 展开，Elastic 只是附带；
- Druid 支持多种外接数据，直接可以对接 Kafka 数据流，也可以直接对接平台自身内部数据；而 Elastic 仅支持内部索引数据，外部数据需要借助三方工具导入到索引里；
- Druid 在数据 Rollup 之后，会丢弃原始数据；Elastic 在原有索引基础之后，生成新的 Rollup 之后的索引数据；
- Druid 与 Elastic 的技术架构非常类似，都支持节点职责分离，都支持横向扩展；
- Druid 与 Elastic 在数据模型上都支持倒排索引，基于此的搜索与过滤。



图示：Druid 产品技术架构体系示意图

关于 Rollup 这个大数据分析领域，若有大规模的 Rollup 的场景需求，个人更倾向于 Druid。

结语

总结：

- Elasticsearch 产品功能全面，适用范围广，性能也不错，综合应用是首选。
- Elasticsearch 在搜索查询领域，几乎完胜所有竞争产品，在笔者的技术栈看来，关系型数据库解决数据事务问题，Elasticsearch 几乎解决一切搜索查询问题。
- Elasticsearch 在数据分析领域，产品能力偏弱一些，简单通用的场景需求可以大规模使用，但在特定业务场景领域，还是要选择更加专业的数据产品，如前文中提到的复杂聚合、大规模 Rollup、大规模的 Key-Value。
- Elasticsearch 越来越不像一个搜索引擎，更像是一个全能型的数据产品，几乎所有行业都在使用，业界非常受欢迎。
- Elasticsearch 用得好，下班下得早。

注：

- 1、内容来源于笔者实际工作中运用多种技术栈实现场景需求，得出的一些实战经验与总结思考，提供后来者借鉴参考。
- 2、本文围绕Elastic的竞争产品对比仅限概要性分析，粒度较粗，深度有限，之后会有更加专业深入竞争产品分析文章，敬请期待。

ES既是搜索引擎又是数据库？真的有那么全能吗？

作者介绍：

李猛(ynuosoft)，Elastic-stack产品深度用户，ES认证工程师，2012年接触Elasticsearch，对Elastic-Stack开发、架构、运维等方面有深入体验，实践过多种Elasticsearch项目，最暴力的大数据分析应用，最复杂的业务系统应用；业余为企业提供Elastic-stack咨询培训以及调优实施。

ES认知

1、ES是什么

1) Elasticsearch是搜索引擎

Elasticsearch在搜索引擎数据库领域排名绝对第一，内核基于Lucene构建，支持全文搜索是职责所在，提供了丰富友好的API。个人早期基于Lucene构建搜索应用，需要考虑的因素太多，自接触到Elasticsearch就再无自主开发搜索应用。普通工程师要想掌控Lucene需要一些代价，且很多机制并不完善，需要做大量的周边辅助程序，而Elasticsearch几乎都已经帮你做完了。

2) Elasticsearch不是搜索引擎

说它不是搜索引擎，估计很多从业者不认可，在个人涉及到的项目中，传统意义上用Elasticsearch来做全文检索的项目占比越来越少，多数时候是用来做精确查询加速，查询条件很多，可以任意组合，查询速度很快，替代其它很多数据库复杂条件查询的场景需求；甚至有的数据库产品直接使用Elasticsearch做二级索引，如HBase、Redis等。Elasticsearch由于自身的一些特性，更像一个多模数据库。

357 systems in ranking, May 2020

Rank			DBMS	Database Model	Score		
May 2020	Apr 2020	May 2019			May 2020	Apr 2020	May 2019
1.	1.	1.	Oracle	Relational, Multi-model	1345.44	+0.02	+59.89
2.	2.	2.	MySQL	Relational, Multi-model	1282.64	+14.29	+63.67
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1078.30	-5.12	+6.12
4.	4.	4.	PostgreSQL	Relational, Multi-model	514.80	+4.95	+35.91
5.	5.	5.	MongoDB	Document, Multi-model	438.99	+0.57	+30.92
6.	6.	6.	IBM Db2	Relational, Multi-model	162.64	-2.99	-11.80
7.	7.	7.	Elasticsearch	Search engine, Multi-model	149.13	+0.22	+0.51
8.	8.	8.	Redis	Key-value, Multi-model	143.48	-1.33	-4.93
9.	9.	11.	SQLite	Relational	123.03	+0.04	+0.14
10.	10.	9.	Microsoft Access	Relational	119.90	-2.02	-23.88

3) Elasticsearch是数据库

Elasticsearch使用Json格式来承载数据模型，已经成为事实上的文档型数据库，虽然底层存储不是Json格式，同类型产品有大名鼎鼎的MongoDB，不过二者在产品定位上有差别，Elasticsearch更加擅长的基于查询搜索的分析型数据库，倾向OLAP；MongoDB定位于事务型应用层面OLTP，虽然也支持数据分析，笔者简单应用过之后再无使用，谁用谁知道。

4) Elasticsearch不是数据库

Elasticsearch不是关系型数据库，内部数据更新采用乐观锁，无严格的ACID事务特性，任何企图将它用在关系型数据库场景的应用都会有很多问题，很多其它领域的从业者喜欢拿这个来作为它的缺陷，重申这不是Elasticsearch的本质缺陷，是产品设计定位如此。



图示：Elastic擅长的应用场景

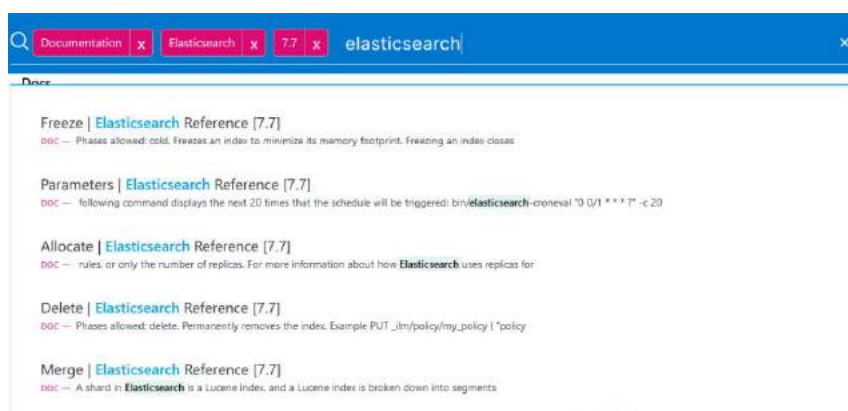
2、ES做什么

Elasticsearch虽然是基于Lucene构建，但应用领域确实非常宽泛。

1) 全文检索

Elasticsearch靠全文检索起步，将Lucene开发包做成一个数据产品，屏蔽了Lucene各种复杂的设置，为开发人员提供了很友好的便利。很多传统的关系型数据库也提供全文检索，有的是基于Lucene内嵌，有的是基于自研，与Elasticsearch比较起来，功能单一，性能也表现不是很好，扩展性几乎没有。

如果，你的应用有全文检索需求，建议你优先迁移到Elasticsearch平台上来，其提供丰富的Full text queries会让你惊讶，一次用爽，一直用爽。



2) 应用查询

Elasticsearch最擅长的就是查询，基于倒排索引核心算法，查询性能强于B-Tree类型所有数据产品，尤其是关系型数据库方面。当数据量超过千万或者上亿时，数据检索的效率非常明显。

个人更看重的是Elasticsearch在通用查询应用场景，关系型数据库由于索引的左侧原则限制，索引执行必须有严格的顺序，如果查询字段很少，可以通过创建少量索引提高查询性能，如果查询字段很多且字段无序，那索引就失去了意义；相反Elasticsearch是默认全部字段都会创建索引，且全部字段查询无需保证顺序，所以我们在业务应用系统中，大量用Elasticsearch替代关系型数据库做通用查询，自此之后对于关系型数据库的查询就很排斥，除了最简单的查询，其余的复杂条件查询全部走Elasticsearch。

3) 大数据领域


Elasticsearch已经成为大数据平台对外提供查询的重要组成部分之一。大数据平台将原始数据经过迭代计算，之后结果输出到一个数据库提供查询，特别是大批量的明细数据。

这里会面临几个问题，一个问题是大批量明细数据的输出，如何能在极短的时间内写到数据库，传统上很多数据平台选择关系型数据库提供查询，比如MySQL，之前在这方面吃过不少亏，瞬间写入性能极差，根本无法满足要求。另一个问题是对外查询，如何能像应用系统一样提供性能极好的查询，不限制查询条件，不限制字段顺序，支持较高的并发，支持海量数据快速检索，也只有Elasticsearch能够做到比较均衡的检索。

从官方的发布版本新特性来看，Elasticsearch志在大数据分析领域，提供了基于列式存储的数据聚合，支持的聚合功能非常多，性能表现也不错，笔者有幸之前大规模深度使用过，颇有感受。

Elasticsearch为了深入数据分析领域，产品又提供了数据Rollup与数据Transform功能，让检索分析更上一层楼。在数据Rollup领域，Apache Druid的竞争能力很强，笔者之前

做过一些对比，单纯的比较确实不如Druid，但自Elasticsearch增加了Transfrom功能，且单独创建了一个Transfrom的节点角色，个人更加看好Elasticseach，跳出了Rollup基于时间序列的限制。



timestamp	domain	gender	clicked
2011-01-01T00:01:35Z	bieber.com	Female	1
2011-01-01T00:03:03Z	bieber.com	Female	0
2011-01-01T00:04:51Z	ultra.com	Male	1
2011-01-01T00:05:33Z	ultra.com	Male	1
2011-01-01T00:05:53Z	ultra.com	Female	0
2011-01-01T00:06:17Z	ultra.com	Female	1
2011-01-01T00:23:15Z	bieber.com	Female	0
2011-01-01T00:38:51Z	ultra.com	Male	1
2011-01-01T00:49:33Z	ultra.com	Female	1
2011-01-01T00:49:53Z	ultra.com	Female	0

timestamp	domain	gender	clicked
2011-01-01T00:00:00Z	bieber.com	Female	1
2011-01-01T00:00:00Z	ultra.com	Male	3
2011-01-01T00:00:00Z	ultra.com	Female	2

4) 日志检索

著名的ELK三件套，讲的就是Elasticsearch，Logstash，Kibana，专门针对日志采集、存储、查询设计的产品组合。很多第一次接触到Elasticsearch的朋友，都会以为Elasticsearch是专门做日志的，其实这些都是误解，只是说它很擅长这个领域，在此领域大有作为，名气很大。

日志自身特点没有什么通用的规范性，人为的随意性很大，日志内容也是任意的，更加需求全文检索能力，传统技术手段本身做全文检索很是吃力。而Elasticsearch本身起步就是靠全文检索，再加上其分布式架构的特性，非常符合海量日志快速检索的场景。今天如果还发现有IT从业人员用传统的技术手段做日志检索，应该要打屁股了。

如今已经从ELK三件套发展到Elastic Stack了，新增加了很多非常有用的产品，大大增强了日志检索领域。

5) 监控领域

指标监控，Elasticsearch进入此领域比较晚，却赶上了好时代，Elasticsearch由于其倒排索引核心算法，也是支持时序数据场景的，性能也是相当不错的，在功能性上完全压住时序数据库。

Elasticsearch搞监控得益于其提供的Elastic Stack产品生态，丰富完善，很多时候监控需要立体化，除了指标之外，还需要有各种日志的采集分析，如果用其它纯指标监控产品，如Promethues，遇到有日志分析的需求，还必须使用Elasticsearch，这对于技术栈来说，又扩增了，相应的掌控能力会下降，个人精力有限，无法同时掌握很多种数据产品，如此选择一个更加通用的产品才符合现实。

	ES	InfluxData	Prometheus	Graphite	OpenTSDB
数据模型	labels	labels	labels	dot -separated	labels
写入性能	*****	*****			***
压缩编码	****	*****			*****
读取性能	*****	*****			*****
数据生命周期管理	✓	✓	✓	✓	手动
集群化支持	✓	✓商业版	单机	单机	✓
降精度（预聚合）	✓	✓	×	✓	×
权限管理	✓	✓商业版	×	×	×

6) 机器学习

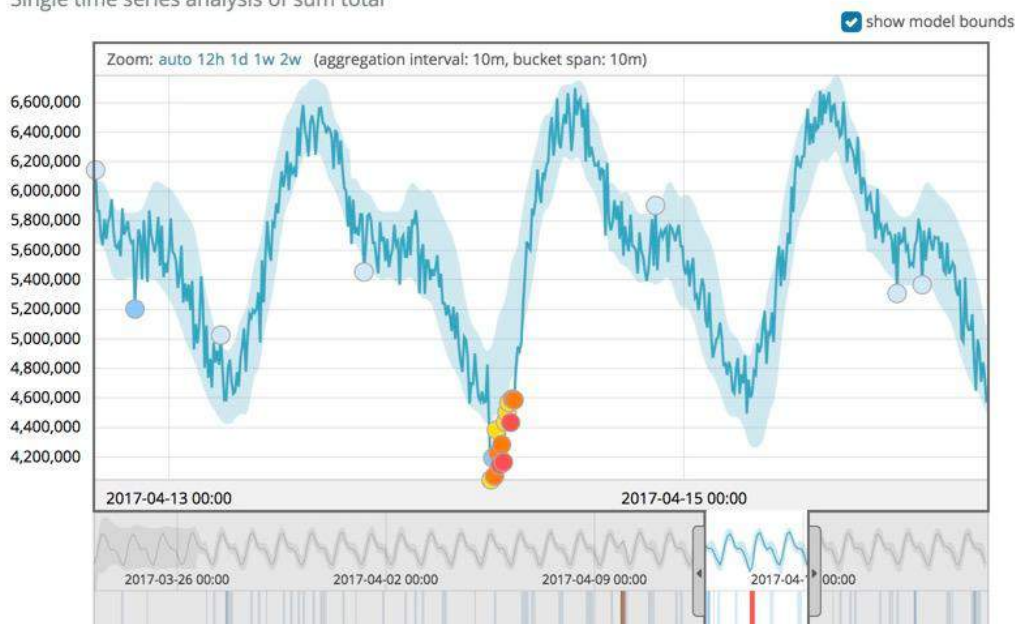
机器学习最近几年风吹的很大，很多数据产品都集成了，Elasticsearch也必须有，而且做的更好，真正将机器学习落地成为一个产品，简化使用，所见所得；而不像其它数据产品，仅仅集成算法包，使用者还必须开发很多应用支持。

Elasticsearch机器学习提供了两种方式，一种是异常检测类型，属于无监督学习，采用聚类模型，通常应用在安全分析领域，检测异常访问等；一种是数据帧分析，属于分类与回归，属于监督学习，可用于在业务模型领域，如电商行业，价格模型分析。

Elasticsearch本身是数据平台，集成了部分机器学习算法，同时又集成了Kibana可视化操作，使得从数据采集、到模型训练、到模型预测应用都可以一键式完成。

Elasticsearch提供的机器学习套件，个人认为最应该应用在数据质量这个领域，帮助大数据平台自动检测数据质量，从而降低人力提供效率。

Single time series analysis of sum total



图示：机器学习应用示意图（截图）

需求等级

Elasticsearch整个的技术栈非常复杂，涉及到的理论与技术点非常多，完全掌握并不现实，作为一个IT从业者，首先是定位好自己的角色，依据角色需求去学习掌握必备的知识。以下是笔者对于一个技术产品的划分模型：

1、概念

Elasticsearch涉及到的概念很多，核心概念其实就那么几个，对于一个新手来说，掌握概念目的是为了建立起自己的知识思维模型，将之后学习到的知识点做一个很好的归纳划分；对于一个其它数据产品的老手来说，掌握概念的目的是为了与其它数据产品划分比较，深入的了解各自的优劣，在之后工作中若有遇到新的业务场景，可以迅速做出抉择。

IT从业者普遍都有个感受，IT技术发展太快了，各种技术框架产品层出不穷，学习掌握太难了，跟不上节奏。其实个人反倒觉得变化不大，基础理论核心概念并没有什么本质的发展变化，无非是工程技术实操变了很多，但这些都是需要深入实践才需要的，对于概念上无需要。

作为一个技术总监，前端工程师工作1~2年的问题都可以问倒他，这是大家对于概念认知需求不一样。



图示：Elasticsearch核心概念

2、开发

开发工程师的职责是将需求变成可以落地运行的代码。Elasticsearch的应用开发工作总结起来就是增删改查，掌握必备的ES REST API，熟练运用足以。笔者之前任职某物流速运公司，负责Elasticsearch相关的工作，公司Elasticsearch的需求很多，尤其是查询方面，ES最厉害的查询是DSL，这个查询语法需要经常练习使用，否则很容易忘记，当每次有人询问时，都安排一个工程师专门负责各种解答，他在编写DSL方面非常熟练，帮助了很多的工程师新手使用Elasticsearch，屏蔽了很多细节，若有一些难搞定的问题，会由我来解决，另外一方面作为负责人的我偶然还要请他帮忙编写DSL。

Elasticsearch后面提供了SQL查询的功能，但比较局限，复杂的查询聚合必须回到DSL。

```
GET /_search
{
  "query": { ❶
    "bool": { ❷
      "must": [
        { "match": { "title": "Search" } }},
        { "match": { "content": "Elasticsearch" } }
      ],
      "filter": [ ❸
        { "term": { "status": "published" } },
        { "range": { "publish_date": { "gte": "2015-01-01" } } }
      ]
    }
  }
}
```

```
GET /_search
{
  "query": { ❶
    "bool": { ❷
      "must": [
        { "match": { "title": "Search" } }},
        { "match": { "content": "Elasticsearch" } }
      ],
      "filter": [ ❸
        { "term": { "status": "published" } }},
        { "range": { "publish_date": { "gte": "2015-01-01" } } }
      ]
    }
  }
}
```

图示：DSL语法复杂度较高

3、架构

Elasticsearch集群架构总体比较复杂，首先得深入了解Elasticsearch背后实现的原理，包括集群原理、索引原理、数据写入过程、数据查询过程等；其次要有很多案例实战的机会，遇到很多挑战问题，逐一排除解决，增加自己的经验。

对于开发工程师来说，满足日常需求开发无需掌握这些，但对于Elasticsearch技术负责人，就非常有必要了，面对各种应用需求，要能从架构思维去平衡，比如日志场景集群需求、大数据分析场景需求、应用系统复杂查询场景需求等，从实际情况设计集群架构以及资源分配等。

4、运维

Elasticsearch本质是一个数据库，也需要有专门的DBA运维，只是更偏重应用层面，所以运维职责相对传统DBA没有那么严苛。对于集群层面必须掌握集群搭建，集群扩容、集群升级、集群安全、集群监控告警等；另外对于数据层面运维，必须掌握数据备份与还原、数据的生命周期管理，还有一些日常问题诊断等。

5、源码

Elasticsearch本身是开源，阅读源码是个很好的学习手段，很多独特的特性官方操作文档并没有写出来，需要从源码中提炼，如集群节点之间的连接数是多少，但对于多数Elasticsearch从业者来说，却非必要。了解到国内主要是头部大厂需要深入源码定制化改造，更多的是集中在应用的便捷性改造，而非结构性的改造，Elastic原厂公司有几百人的团队做产品研发，而国内多数公司就极少的人，所以从产量上来说，根本不是一个等级的。

如果把Elasticsearch比喻为一件军事武器，对于士兵来说，熟练运用才是最重要的，至于改造应该是武器制造商的职责，一个士兵可以使用很多武器装备，用最佳的组合才能打赢一场战争，而不是去深入原理然后造轮子，容易本末倒置。

6、算法

算法应该算是数据产品本质的区别，关系型数据库索引算法主要是基于B-Tree，Elasticsearch索引算法主要是倒排索引，算法的本质决定了它们的应用边界，擅长的应用领域。

通常掌握一个新的数据产品时，个人的做法是看它的关键算法。早期做过一个地理位置搜索相关的项目，基于某个坐标搜索周边的坐标信息，开始的时候采用的是三角函数动态计算的方式，数据量大一点，扫描一张数据表要很久；后面接触到Geohash算法，按照算法将坐标编码，存储在数据库中，基于前缀匹配查询，性能高效几个数量级，感叹算法的伟大；再后面发现有专门的数据库产品集成了Geohash算法，使用起来就更简单了。

Elasticsearch集成很多算法，每种算法实现都有它的应用场景。

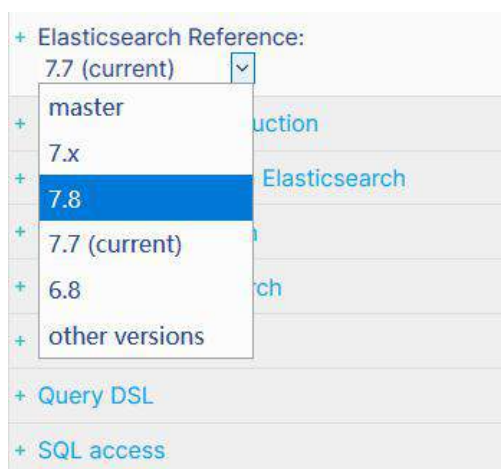
拥抱ES的方法

1、官方文档

Elasticsearch早期出过一本参考手册《Elastic权威指南》，是一本很好的入门手册，从概念到实战都有涉及，缺点是版本针对的2.0，过于陈旧，除去核心概念，其余的皆不适用，当前最新版本已经是7.7了，跨度太大，Elasticsearch在跨度大的版本之间升级稍微比较麻烦，索引数据几乎是不兼容的，升级之后需要重建数据才可。

Elasticsearch当前最好的参考资料是官方文档，资料最全，同步发布版本，且同时可以参考多个版本。Elasticsearch官方参考文档也是最乱的，什么资料都有，系统的看完之后感觉仍在此山中，有点类似一本字典，看完了字典，依然写不好作文；而且资料还是英文的，至此就阻挡了国内大部分程序进入。

但想要学习Elasticsearch，官方文档至少要看几遍，便于迅速查询定位。



图示：官方文档截图说明

2、系统学习

Elasticsearch成名很早，国内也有很多视频课程，多数比较碎片，或是纸上谈兵，缺乏实战经验。Elasticsearch有一些专门的书籍，建议购买阅读，国内深度一些的推荐

《Elasticsearch源码解析与优化实战》，国外推荐《Elasticsearch实战》，而且看书还有助于培养系统思维。

Elasticsearch技术栈功能特性很多，系统学习要保持好的心态，持之以恒，需要很长时间，也需要参考很多资料。

3、背后原理

Elasticsearch是站在巨人肩膀上产品，背后借鉴了很多设计思想，集成了很多算法，官方的参考文档在技术原理探讨这块并没有深入，仅仅点到为止。想要深入了解，必须得另辟蹊径。

Elastic官方的博客有很多优质的文章，很多人因为英文的缘故会忽视掉，里面有很多关键的实现原理，图文并茂，写得非常不错；另外国内一些云厂商由于提供了Elasticsearch云产品，需要深度定制开发，也会有一些深入原理系列的文章，可以去阅读参考，加深理解。对于已经有比较好的编程思维的人，也可以直接去下载官方源码，设置断点调试阅读。

4、项目实战

项目实战是非常有效的学习途径，考过驾照的朋友都深有体会，教练一上来就直接让你操练车，通过很多次的练习就掌握了。Elasticsearch擅长的领域很多，总结一句话就是“非强事务ACID场景皆可适用”，所以可以做的事情也很多。

日志领域的需求会让你对于数据写入量非常的关心，不断的调整优化策略，提高吞吐量，降低资源消耗；业务系统的需求会让你对数据一致性与时效性特别关心，从其它数据库同步到ES，关注数据同步的速度，关注数据的准确性，不断的调整你的技术方案与策略；大数据领域的需求会让你对于查询与聚合特别关注，海量的数据需要快速的检索，也需要快速的聚合结果。

项目实战的过程，就是一个挖坑填坑的过程，实战场景多了，解决的问题多了，自然就掌握得很好了。

之前笔者在前公司任职时，所有涉及到的Elasticsearch疑难杂症都会找我解决，有一些项目采用别的数据产品问题比较多，也来找我评估更换ES是否合适，以及给出相关建议。笔者认为最好的学习方式是找到组织，找到经验丰富的大咖，持续交流学习，成长最快也最好。

DB 与 Elasticsearch 混合之应用系统场景分析探讨

简介：从技术、业务两个层面探讨，为什么要使用 DB 结合 ES 混用的模式。

作者介绍

李猛，Elastic Stack深度用户，通过Elastic工程师认证，2012年接触Elasticsearch，对Elastic Stack技术栈开发、架构、运维等方面有深入体验，实践过多种大中型项目；为企业提供Elastic Stack咨询培训以及调优实施；多年实战经验，爱捣腾各种技术产品，擅长大数据，机器学习，系统架构。

名词解释

354 systems in ranking, March 2020

Rank			DBMS	Database Model	Score		
Mar 2020	Feb 2020	Mar 2019			Mar 2020	Feb 2020	Mar 2019
1.	1.	1.	Oracle	Relational, Multi-model	1340.64	-4.11	+61.50
2.	2.	2.	MySQL	Relational, Multi-model	1259.73	-7.92	+61.48
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1097.86	+4.11	+50.01
4.	4.	4.	PostgreSQL	Relational, Multi-model	513.92	+6.98	+44.11
5.	5.	5.	MongoDB	Document, Multi-model	437.61	+4.28	+36.27
6.	6.	6.	IBM Db2	Relational, Multi-model	162.56	-2.99	-14.64
7.	7.	9.	Elasticsearch	Search engine, Multi-model	149.17	-2.98	+6.38
8.	8.	8.	Redis	Key-value, Multi-model	147.58	-3.84	+1.46
9.	9.	7.	Microsoft Access	Relational	125.14	-2.92	-21.07
10.	10.	10.	SQLite	Relational	121.95	-1.41	-2.92

db-engine 当前综合排名

DB: database，泛指关系型数据库，具有严格事务隔离机制的数据类库产品，如mysql、sqlserver、postgresql、oracle、db2 等，db-engine 综合排名前面的全部是关系型数据库；

ES: Elasticsearch，最好的开源搜索引擎产品，NoSQL 非关系型数据库，不具备严格事务隔离机制，当前 db-engine 综合排名第七；

应用: 本文泛指业务应用系统，是 OLTP 场景，非 OLAP 场景，大量运用事务型数据库产品的业务系统；

索引：在关系型数据库中是指数据检索的算法，在 Elasticsearch 是指数据存储的虚拟空间概念，类同数据库中的表。

背景需求

- 为什么单一 DB 不能满足应用系统？
- 为什么单一 ES 不能满足应用系统？
- 为什么要使用 DB 结合 ES 混用的模式？

而不是结合其它 NoSQL？比如 MongoDB？下面从技术层面与业务层面分析探讨。

技术层面



DB 与 ES 的技术特性说明

DB 局限性

关系型数据库索引基于最左优先原则，索引要按照查询需求顺序提前创建，不具备任意字段组合索引能力比如 某 xxx 表有 30 个字段，若要求依据任意字段组合条件查询，此时关系型数据库显然无法满足这种灵活需求，包括当下很受欢迎的 NoSQL 明星产品 MongoDB 也不能满足。

关系型数据库支持有限度的关联查询，一般在业务应用系统中，关联表会控制在 2~3 个表（个人观点：有 3 个表关联的业务场景需要由技术架构师评估是否容许，开发工程师只容许 2 个表关联），且单表的数据量是要平衡考虑才可以，跨同构数据库源关联就更加不容许。

那跨异构数据库源呢？不是不容许，实在是有心无力。关系型数据库普遍采用 B+ 树数据结构实现索引查询，面对超大数据量处理有天然的瓶颈，数据量超过百万千万级，复杂点的查询，性能下降很快，甚至无法运行。

关系型数据库虽然也支持主从同步，支持读写分离，但集群是一种松散式的架构，集群节点感知能力脆弱，不成体系，弹性扩展能力不行。

ES 互补性

Elasticsearch 基于 Lucene 核心库构建，以倒排索引算法为基础，集成多种高效算法。

Elasticsearch 默认为所有字段创建索引，任索引字段可组合使用，且查询效率相当高，相比关系型数据库索更加高效灵活。在业务系统中，有很多场景都需要这种任意字段组合查询的能力，简称通用搜索。

Elasticsearch 的数据模型采用的是 Free Scheme 模式，以 JSON 主体，数据字段可以灵活添加，字段层级位置也可以灵活设置，关系数据库中需要多表关联查询，在 Elasticsearch 中可以通过反范式的关联能力，将多个业务表数据合并到一个索引中，采用对象嵌套的方式。

Elasticsearch 天然分布式设计，副本与分片机制使得集群具备弹性扩展能力，可以应付超大的数据量查询，在单索引数据量十亿级也可以在亚秒内响应查询。

Elasticsearch 的索引支持弹性搜索，可以指定一个索引搜索，也可以指定多个索引搜索，搜索结果由 ES 提供过滤合并，这就为业务系统提供了很灵活的操作空间，特别是在实时数据与历史数据方面，统一查询条件语法皆可执行。

Elasticsearch 支持数据字段与索引字段分离，默认情况下，数据字段与索引字段是同时启用，实际在业务场景中，数据表中很多字段无需检索，只是为了存储数据，在查询时方便取回；很多检索字段也无需存储原始数据，只是检索过滤使用。

DB 不等于 ES

关系数据库定位全能型产品，强事务机制，数据写入性能一般，数据查询能力一般。

Elasticsearch 定位查询分析型产品，弱事务机制，查询性能非常不错。DB 与 ES 各有优势劣势，不能等同取代。

业务层面



业务领域复杂度

在进入互联网/移动互联网/物联网之后，业务系统数据量的几何倍数增长，传统应当策略当然是采用分库分表机制，包括物理层面分库分表，逻辑层面分区等。非重要数据可以采用非关系型数据库存储，重要的业务数据一定是采用关系数据库存储，比如物流速运行业的客户订单数据，不容许有丢失出错。

面对复杂业务系统需求，当下最流行的解决方式是采用微服务架构模式，微服务不仅仅局限上层应用服务，更深层次的是底层数据服务（微服务不在本次讨论之内），基于领域模型思维拆分分解。应用服务拆分成大大小小数个，可以几十个几百个，也可以更多，数据服务也拆分成大大小小数个。

业务查询复杂度

分库分表解决了数据的存储问题，但需要做合并查询却是个很麻烦的事，业务系统的查询条件一般是动态的，无法固定，更加不可能在分库分表时按照所有动态条件设计，这似乎代价太大。一般会选择更强大的查询数据库，比如 Elasticsearch 就非常合适。

微服务架构模式解决了业务系统的单一耦合问题，但业务系统的查询复杂度确实提高不少，复杂点的应用服务执行一次查询，需要融合多个领域数据服务才能完成，特别是核心领域数据服务，几乎贯穿一个系统所有方面，比如物流速运行业订单领域。

DB 与 ES 结合

业务数据存储由关系型数据库负责，有强事务隔离机制，保障数据不丢失、不串乱、不覆盖，实时可靠。

业务数据查询由 Elasticsearch 负责，分库分表的数据合并同步到 ES 索引；跨领域库表数据合并到同步 ES 索引，这样就可以高效查询。

DB 与 ES 结合问题



DB 与 ES 结合的问题

关于DB与Elasticsearch混合主要有以下两个问题：同步实时性、数据一致性，本文不探讨此问题，后续会有专题讲述如何解决。

混合场景

前面已经论证了关系型数据库与 Elasticsearch 混合使用的必要性，接下来是探讨混合场景下的业务场景数据模型映射。

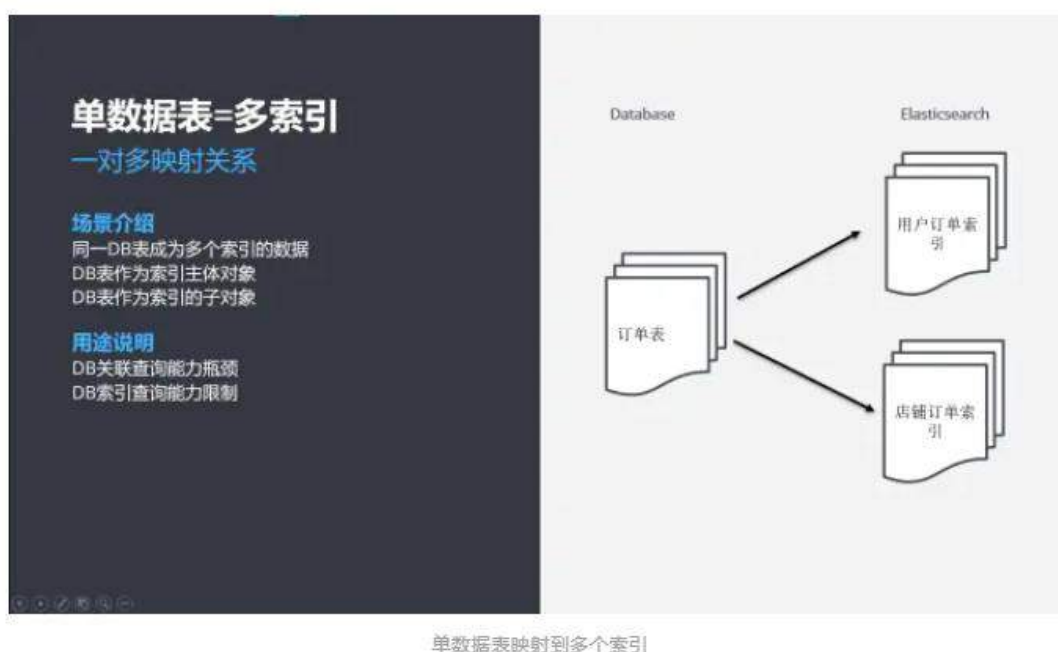
单数据表 -> 单索引



单数据表映射到单一索引

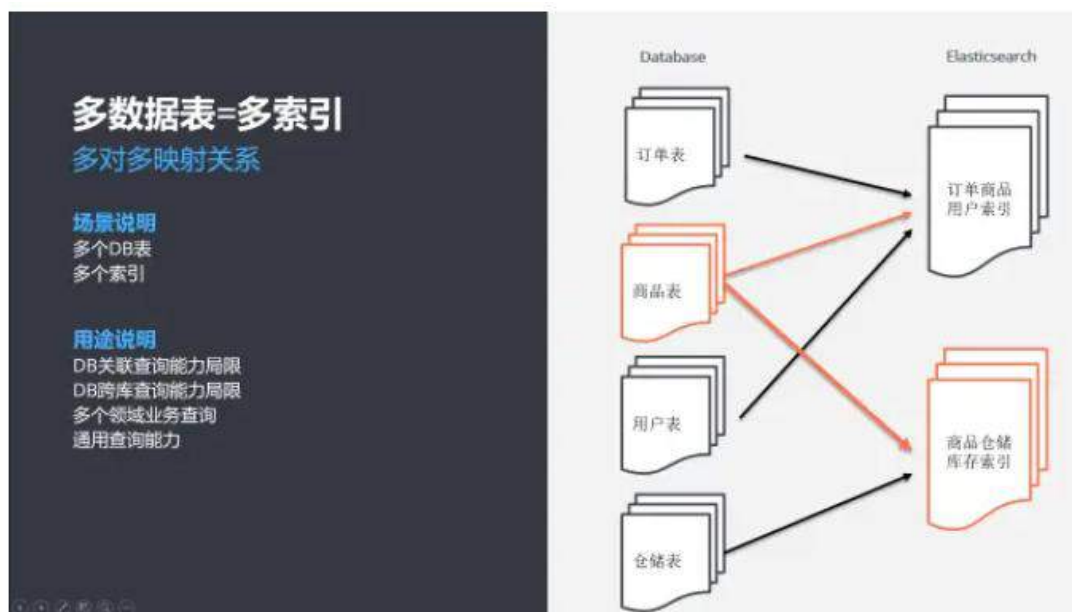
- 1) 一对一映射关系，关系数据库有多少个表，Elasticsearch 就有多少个索引；
- 2) 关系数据库提供原始数据源，Elasticsearch 替代数据库成为查询引擎，替代列表查询场景；
- 3) 单数据表为水平分库分表设计，需要借助Elasticsearch 合并查询，如图：电商行业订单场景，日均订单量超过百万千万级，后端业务系统有需求合并查询；
- 4) 单数据表业务查询组合条件多，数据库索引查询能力局限，需要借助 Elasticsearch 全字段索引查询能力，主要替代表查询场景。如：电商行业商品搜索场景，商品基础字段超过几十个，几乎全部都可以组合搜索。

单数据表 -> 多索引



- 1) 一对多映射关系，单数据表映射到多个索引中；
- 2) 单数据表即作为 A索引的主体对象，一对一映射；
- 3) 单数据表也作为 B索引的子对象，嵌入到主体对象下面；
- 4) 基于微服务架构设计，在业务系统中，业务系统划分多个子领域，子领域也可以继续细分，如电商行业，订单领域与商品领域，订单表需要映射到订单索引，也需要与商品索引映射。

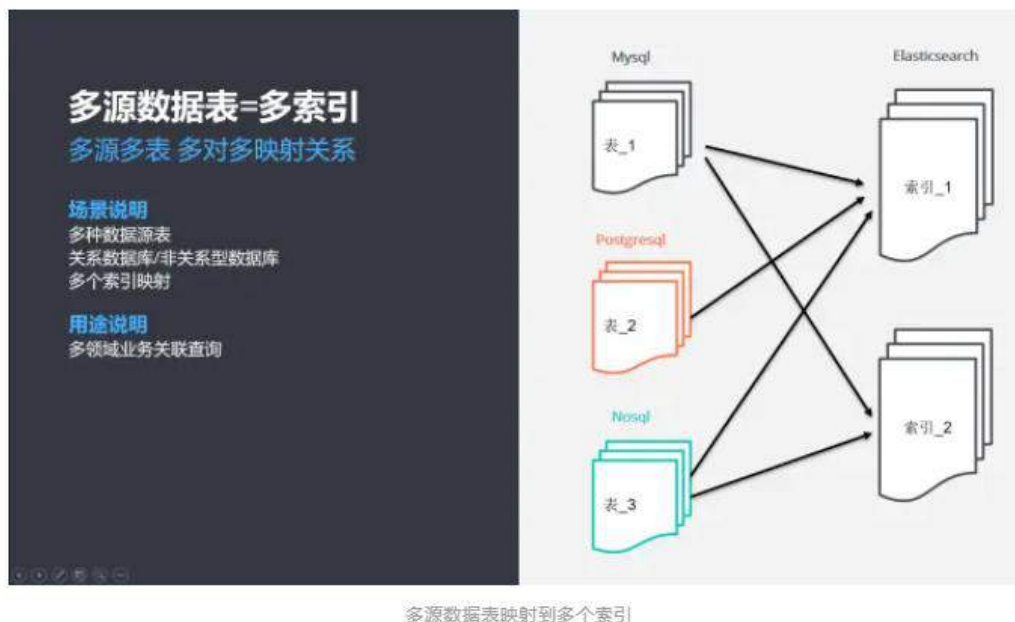
多数据表 -> 多索引



多数据表映射到多个索引

- 1) 多对多映射关系，多个数据表映射到多个索引，复杂度高；
- 2) 一个中型以上的业务系统，会划分成多个领域，单领域会持续细分为多个子领域，领域之间会形成网状关系，业务数据相互关联；
- 3) 数据库表关联查询效率低，跨库查询能力局限，需要借助 Elasticsearch 合并；
- 4) 按照领域需求不同，合并为不同的索引文件，各索引应用会有差异，部分是通用型的，面向多个领域公用；部分是特殊型的，面向单个领域专用。

多源数据表 -> 多索引



- 1) 多源多表，多对多映射关系，数据表与索引之间的映射关系是交叉型的，复杂度最高；
- 2) 一个大中型业务系统，不同的业务场景会采用不同的数据存储系统；
- 3) 关系型数据库多样化，如 A 项目采用 MySQL，B 项目采用 PostgreSQL，C 项目选用 SQLServer，业务系统通用型的查询几乎不能实现；
- 4) 非关系型数据库多样化，如 A 项目采用键值类型的 Redis，B 项目选用文档型的 MongoDB，业务系统同样也不能实现通用型查询；
- 5) 基于异构数据源通用查询的场景需求，同样需要借助 Elasticsearch 合并数据实现。

结语

Elasticsearch 虽然早期定位是搜索引擎类产品，后期定位数据分析类产品，属于 NoSQL 阵营，且不支持严格事务隔离机制，但由于其先进的特性，在应用系统中也是可以大规模使用，能有效弥补了关系型数据库的不足。

本文主旨探讨了 DB 与 ES 混合的需求背景与应用场景，目的不是评选 DB 与 ES 谁更优劣，是要学会掌握 DB 与 ES 平衡，更加灵活的运用到应用系统中去，满足不同的应用场景需求，解决业务需求问题才是评判的标准。

初次使用 Elasticsearch 遇多种分词难题？那是你没掌握这些原理

简介：命名有包含搜索关键词的文档，但结果却没有？存进去的文档被分成哪些词(term)了？自定义分词规则，但感觉好麻烦呢，无从下手？

作者介绍

魏彬，普翔科技 CTO，开源软件爱好者，中国第一位 Elastic 认证工程师，《Elastic日报》和《ElasticTalk》社区项目发起人，被 elastic 中国公司授予 2019 年度合作伙伴架构师特别贡献奖。对 Elasticsearch、Kibana、Beats、Logstash、Grafana 等开源软件有丰富的实践经验，为零售、金融、保险、证券、科技等众多行业的客户提供过咨询和培训服务，帮助客户在实际业务中找准开源软件的定位，实现从 0 到 1 的落地、从 1 到 N 的拓展，产生实际的业务价值。

初次接触 Elasticsearch 的同学经常会遇到分词相关的难题，比如如下这些场景：

- 1、为什么命名有包含搜索关键词的文档，但结果里面就没有相关文档呢？
- 2、我存进去的文档到底被分成哪些词(term)了？
- 3、我得自定义分词规则，但感觉好麻烦呢，无从下手

如果你遇到过类似的问题，希望本文可以解决你的疑惑。

一、上手

让我们从一个实例出发，如下创建一个文档：

```
1. PUT test/doc/1
2. {
3.   "msg": "Eating an apple a day keeps doctor away"
4. }
```


然后我们做一个查询，我们试图通过搜索 `eat` 这个关键词来搜索这个文档

```
1.  POST test/_search
2.  {
3.    "query":{
4.      "match":{
5.        "msg":"eat"
6.      }
7.    }
8.  }
9.  }
```

ES 的返回结果为 0。这不太对啊，我们用最基本的字符串查找也应该能匹配到上面新建的文档才对啊！

各位不要急，我们先来看看什么是分词。

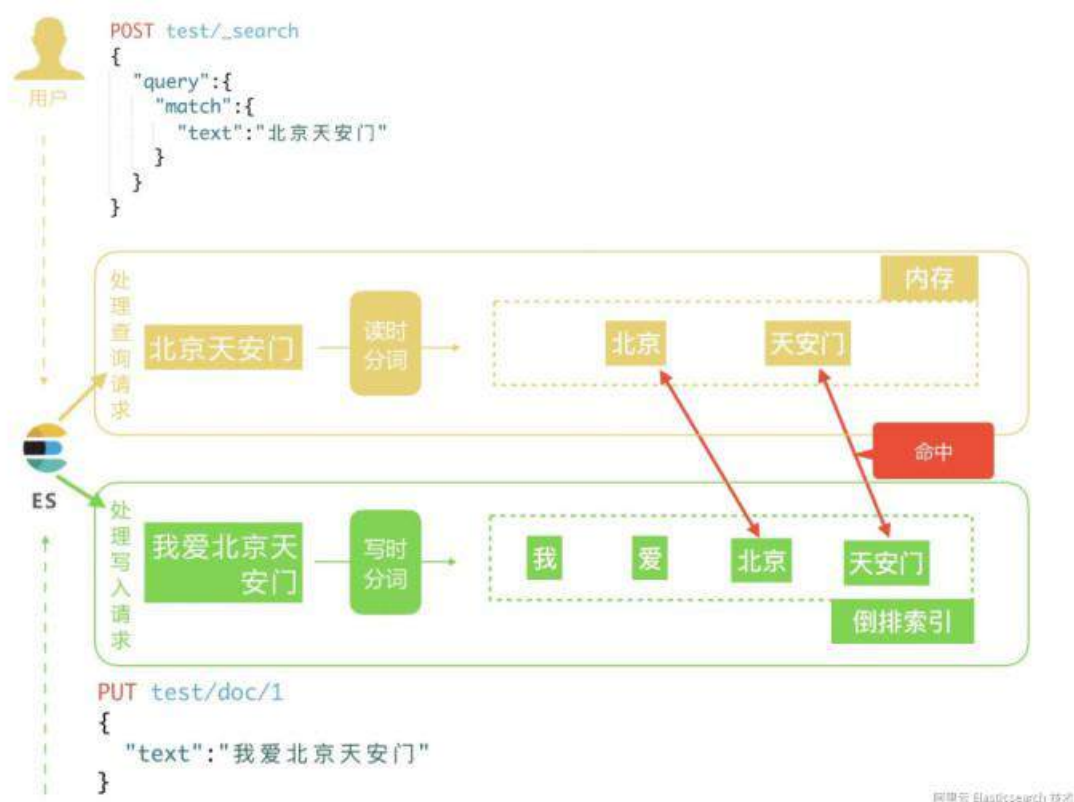
二、分词

搜索引擎的核心是倒排索引（这里不展开讲），而倒排索引的基础就是分词。所谓分词可以简单理解为将一个完整的句子切割为一个个单词的过程。在 es 中单词对应英文为 `term`。我们简单看个例子：



ES 的倒排索引即是根据分词后的单词创建，即 我、爱、北京、天安门这4个单词。这也意味着你在搜索的时候也只能搜索这4个单词才能命中该文档。

实际上 ES 的分词不仅仅发生在文档创建的时候，也发生在搜索的时候，如下图所示：



读时分词发生在用户查询时，ES 会即时地对用户输入的关键词进行分词，分词结果只存在内存中，当查询结束时，分词结果也会随即消失。而写时分词发生在文档写入时，ES 会对文档进行分词后，将结果存入倒排索引，该部分最终会以文件的形式存储于磁盘上，不会因查询结束或者 ES 重启而丢失。

ES 中处理分词的部分被称作分词器，英文是 Analyzer，它决定了分词的规则。ES 自带了很多默认的分词器，比如 Standard、Keyword、Whitespace 等等，默认是 Standard。当我们在读时或者写时分词时可以指定要使用的分词器。

三、写时分词结果

回到上手阶段，我们来看下写入的文档最终分词结果是什么。通过如下 api 可以查看：

```
1.      POST test/_analyze
2.      {
3.        "field": "msg",
4.        "text": "Eating an apple a day keeps doctor away"
5.      }
```

其中 test 为索引名，_analyze 为查看分词结果的 endpoint，请求体中 field 为要查看的字段名，text 为具体值。该 api 的作用就是请告诉我在 test 索引使用 msg 字段存储一段文本时，es 会如何分词。

返回结果如下：

```
6.      {
7.        "tokens": [
8.          {
9.            "token": "eating",
10.           "start_offset": 0,
11.           "end_offset": 6,
12.           "type": "<ALPHANUM>",
13.           "position": 0
14.         },
15.         {
16.           "token": "an",
17.           "start_offset": 7,
18.           "end_offset": 9,
19.           "type": "<ALPHANUM>",
20.           "position": 1
21.         },
22.         {
23.           "token": "apple",
```

```
24.     "start_offset": 10,
25.     "end_offset": 15,
26.     "type": "<ALPHANUM>",
27.     "position": 2
28.   },
29.   {
30.     "token": "a",
31.     "start_offset": 16,
32.     "end_offset": 17,
33.     "type": "<ALPHANUM>",
34.     "position": 3
35.   },
36.   {
37.     "token": "day",
38.     "start_offset": 18,
39.     "end_offset": 21,
40.     "type": "<ALPHANUM>",
41.     "position": 4
42.   },
43.   {
44.     "token": "keeps",
45.     "start_offset": 22,
46.     "end_offset": 27,
47.     "type": "<ALPHANUM>",
48.     "position": 5
49.   },
50.   {
51.     "token": "doctor",
52.     "start_offset": 28,
53.     "end_offset": 34,
54.     "type": "<ALPHANUM>",
55.     "position": 6
56.   },
```

```
57.    {
58.      "token": "away",
59.      "start_offset": 35,
60.      "end_offset": 39,
61.      "type": "<ALPHANUM>",
62.      "position": 7
63.    }
64.  ]
65. }
```

返回结果中的每一个 token 即为分词后的每一个单词，我们可以看到这里是没有 eat 这个单词的，这也解释了在上手中我们搜索 eat 没有结果的情况。如果你去搜索 eating，会有结果返回。

写时分词器需要在 mapping 中指定，而且一经指定就不能再修改，若要修改必须新建索引。如下所示我们新建一个名为 msg_english 的字段，指定其分词器为 english：

```
1.    PUT test/_mapping/doc
2.    {
3.      "properties": {
4.        "msg_english": {
5.          "type": "text",
6.          "analyzer": "english"
7.        }
8.      }
9.    }
```

四、读时分词结果

由于读时分词器默认与写时分词器默认保持一致，拿上手 中的例子，你搜索 msg 字段，那么读时分词器为 Standard，搜索 msg_english 时分词器则为 english。这种默认设定也是非常容易理解的，读写采用一致的分词器，才能尽最大可能保证分词的结果是可以匹配的。

然后 ES 允许读时分词器单独设置，如下所示：

```
1.      POST test/_search
2.      {
3.        "query":{
4.          "match":{
5.            "msg":{
6.              "query": "eating",
7.              "analyzer": "english"
8.            }
9.          }
10.        }
11.      }
```

如上 analyzer 字段即可以自定义读时分词器，一般来讲不需要特别指定读时分词器。

如果不单独设置分词器，那么读时分词器的验证方法与写时一致；如果是自定义分词器，那么可以使用如下的 api 来自行验证结果。

```
1.      POST _analyze
2.      {
3.        "text":"eating",
4.        "analyzer":"english"
5.      }
```

返回结果如下：

```
1.      {
2.        "tokens": [
3.          {
4.            "token": "eat",
5.            "start_offset": 0,
6.            "end_offset": 6,
7.            "type": "<ALPHANUM>",
```



```
8.      "position": 0
9.      }
10.     ]
11.    }
```

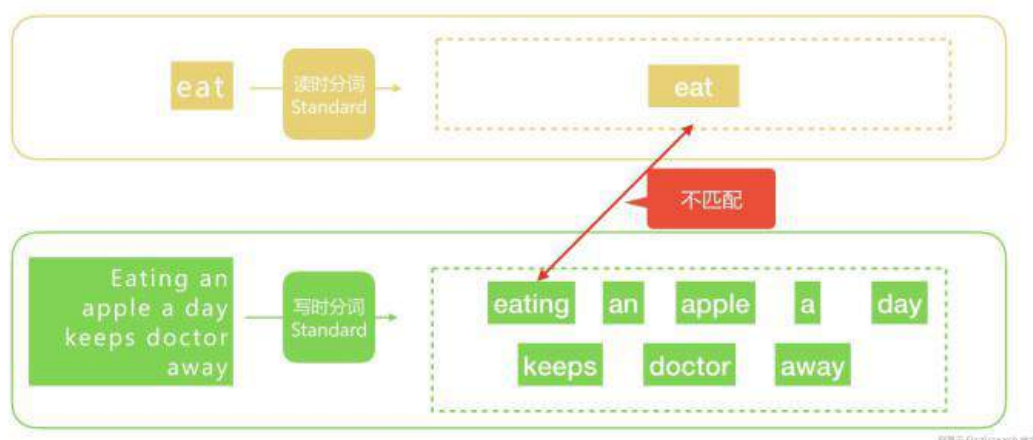
由上可知 english分词器会将 eating处理为 eat，大家可以再测试下默认的 standard分词器，它没有做任何处理。

五、解释问题

现在我们再来看下 上手 中所遇问题的解决思路。

- 1、查看文档写时分词结果
- 2、查看查询关键词的读时分词结果
- 3、比对两者是否有命中

我们简单分析如下：



由上图可以定位问题的原因了。

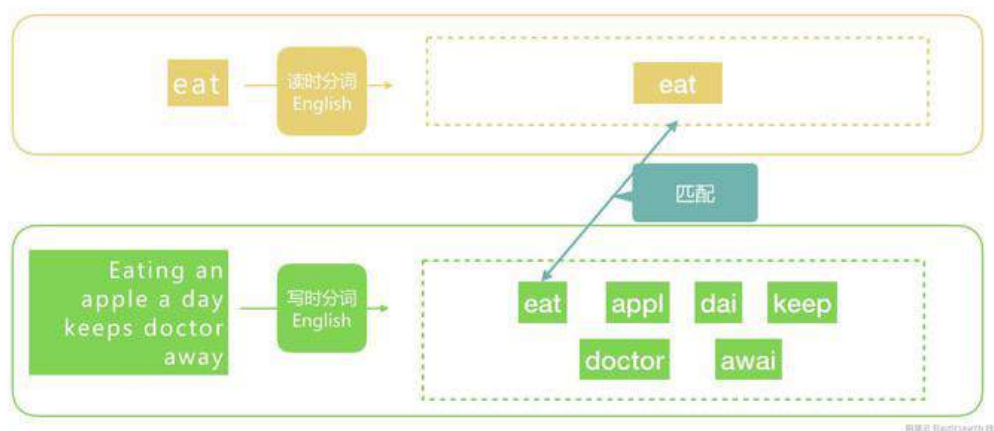
六、解决需求

由于 eating只是 eat的一个变形，我们依然希望输入 eat时可以匹配包含 eating的文档，那么该如何解决呢？

答案很简单，既然原因是在分词结果不匹配，那么我们就换一个分词器呗~ 我们可以先试下 ES 自带的 english 分词器，如下：

```
1.      # 增加字段 msg_english, 与 msg 做对比
2.      PUT test/_mapping/doc
3.      {
4.      "properties": {
5.      6."msg_english":{
6.      "type":"text",
7.      "analyzer": "english"
8.      }
9.      }
10.     }# 写入相同文档
11.     PUT test/doc/1
12.     {
13.     "msg":"Eating an apple a day keeps doctor away",
14.     "msg_english":"Eating an apple a day keeps doctor away"
15.     }# 搜索 msg_english 字段
16.     POST test/_search
17.     {
18.     "query": {
19.     "match": {
20.     "msg_english": "eat"
21.     }
22.     }
23.     }
```

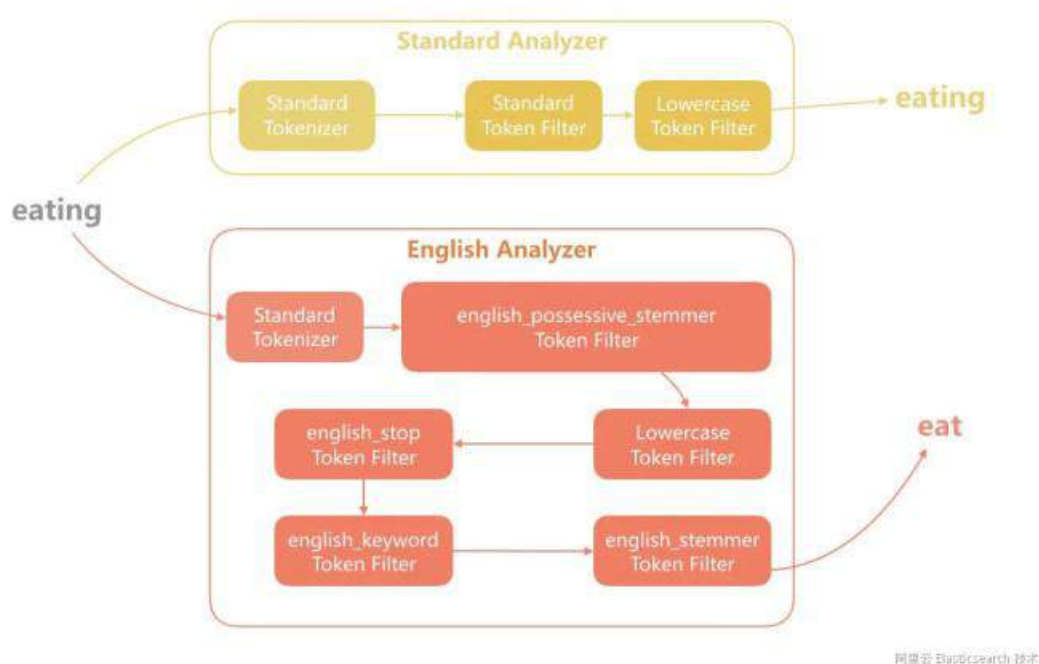
执行上面的内容，我们会发现结果有内容了，原因也很简单，如下图所示：



由上图可见 english分词器会将 eating分词为 eat，此时我们搜索 eat或者 eating肯定都可以匹配对应的文档了。至此，需求解决。

七、深入分析

最后我们来看下为什么english分词器可以解决我们遇到的问题。一个分词器由三部分组成：char filter、tokenizer 和 token filter。各部分的作用我们这里就不展开了，我们来看下 standard和english分词器的区别。



从上图可以看出，english分词器在 Token Filter 中和 Standard不同，而发挥主要作用的就是 stemmer，感兴趣的同学可以自行去看起它的作用。

八、自定义分词

如果我们不使用 english分词器，自定义一个分词器来实现上述需求也是完全可行的，这里不详细讲解了，只给大家讲一个快速验证自定义分词器效果的方法，如下：

```
1.      POST _analyze
2.      {
3.        "char_filter": [],
4.        "tokenizer": "standard",
5.        "filter": [
6.          "stop",
7.          "lowercase",
8.          "stemmer"
9.        ],
10.       "text": "Eating an apple a day keeps doctor away"
11.     }
```

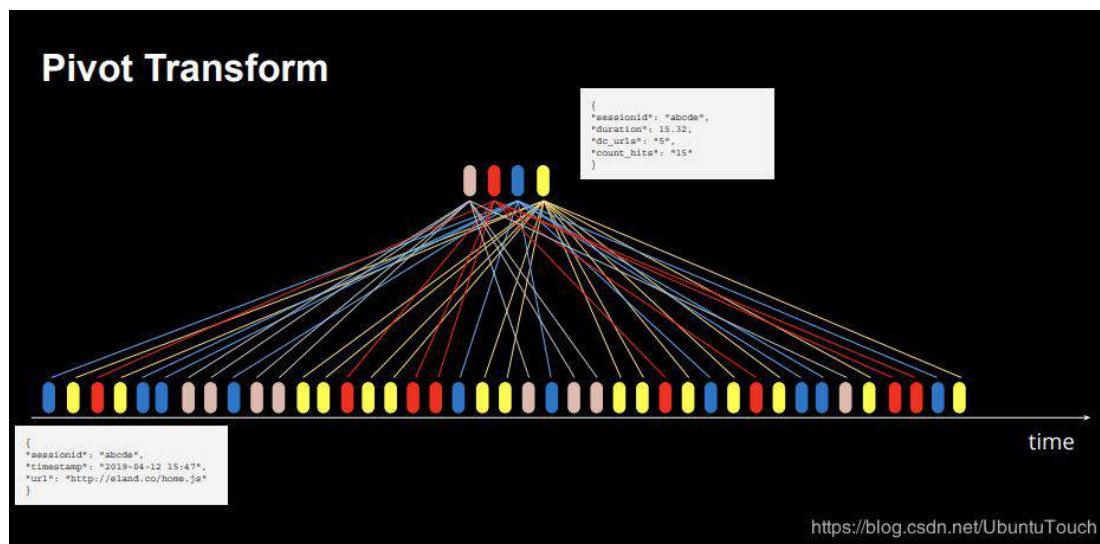
通过上面的 api 你可以快速验证自己要定制的分词器，当达到自己需求后，再将这一部分配置加入索引的配置。

至此，我们再看开篇的三个问题，相信你已经心里有答案了，赶紧上手去自行测试下吧！

Transforms数据透视让Elasticsearch数据更易分析

简介：Transforms 使您能够从 Elasticsearch 索引中检索信息，对其进行转换并将其存储在另一个索引中。使您能够透视数据并创建以实体为中心的索引，这些索引可以汇总实体的行为。这会将数据组织成易于分析的格式。让我们使用Kibana示例数据来演示如何使用变换来透视和汇总数据。

本文作者：Elastic 中国社区布道师——刘晓国



准备数据

在今天的练习中，我们将以 eCommerce 订单的样本例子来做练习。

首先，准备阿里云elasticsearch 6.7 版本环境，并使用创建的账号密码登录Kibana，将数据导入到 Elasticsearch 中：

The image shows two screenshots of the Kibana interface. The top screenshot is the 'Home' page, which is divided into 'Observability' and 'Security' sections. Under 'Observability', there are four cards: 'APM', 'Logs', 'Metrics', and 'SIEM'. Each card has a description and an 'Add' button. Below these cards, there are three options: 'Add sample data' (highlighted with a red box), 'Upload data from log file', and 'Use Elasticsearch data'. The bottom screenshot is the 'Add data' page, which has tabs for 'All', 'Logs', 'Metrics', 'SIEM', and 'Sample data'. The 'Sample data' tab is selected, showing three cards: 'Sample eCommerce orders', 'Sample flight data', and 'Sample web logs'. Each card has a description and an 'Add data' button. The 'Add data' button for 'Sample eCommerce orders' is highlighted with a red box.

Home

Observability

APM
APM automatically collects in-depth performance metrics and errors from inside your applications.
[Add APM](#)

Logs
Ingest logs from popular data sources and easily visualize in preconfigured dashboards.
[Add log data](#)

Metrics
Collect metrics from the operating system and services running on your servers.
[Add metric data](#)

Security

SIEM
Centralize security events for interactive investigation in ready-to-go visualizations.
[Add events](#)

Add sample data
[Load a data set and a Kibana dashboard](#)

Upload data from log file
[Import a CSV, NDJSON, or log file](#)

Use Elasticsearch data
[Connect to your Elasticsearch index](#)

Add data

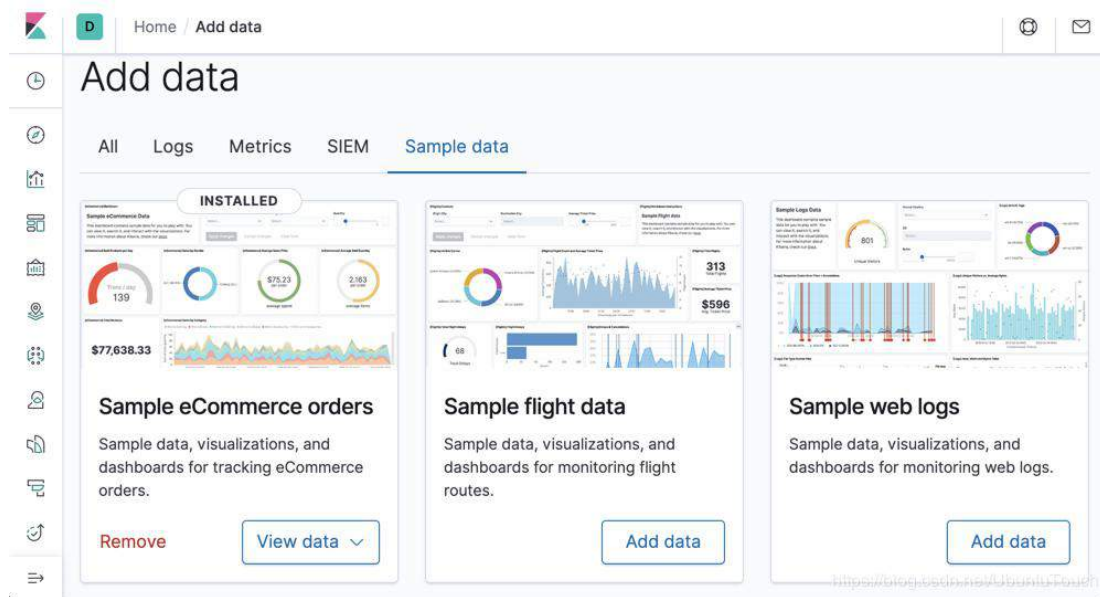
Sample data

Sample eCommerce orders
Sample data, visualizations, and dashboards for tracking eCommerce orders.
[Add data](#)

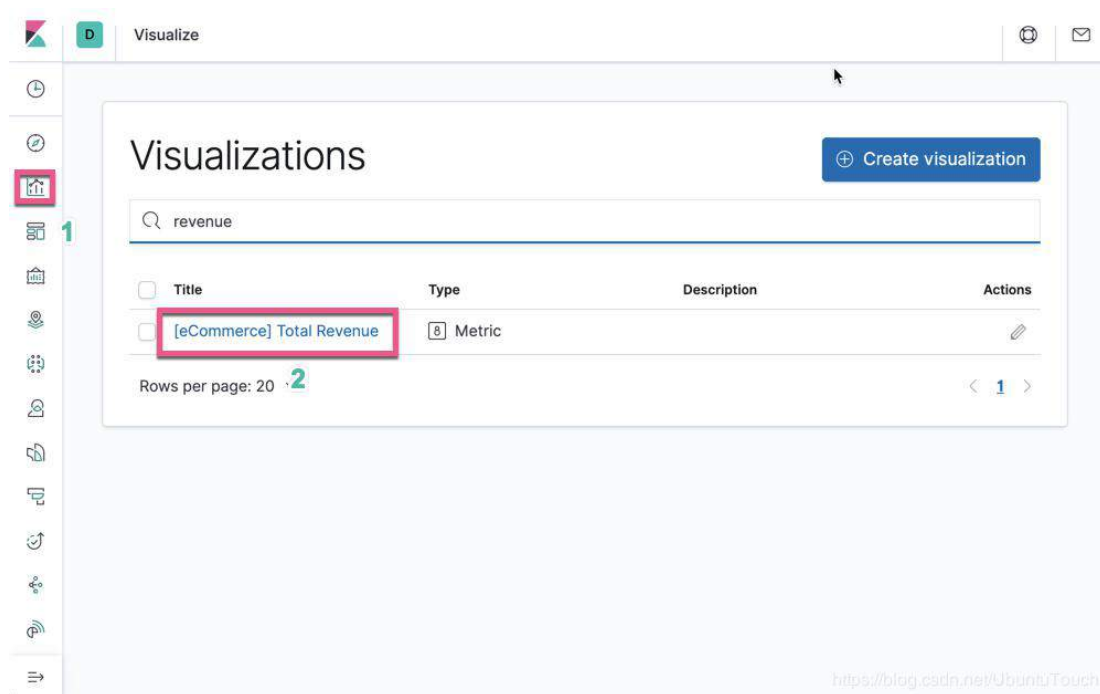
Sample flight data
Sample data, visualizations, and dashboards for monitoring flight routes.
[Add data](#)

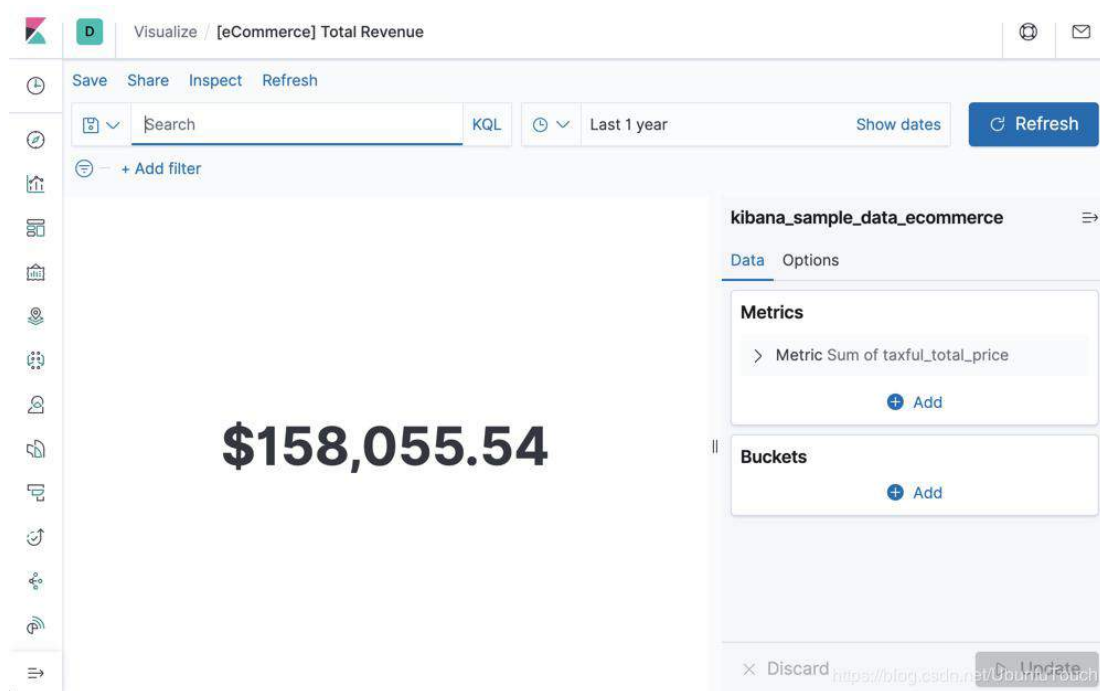
Sample web logs
Sample data, visualizations, and dashboards for monitoring web logs.
[Add data](#)

点击 Add data 按钮:



这样我们就完成了 eCommerce 数据的导入。如果您还不熟悉 kibana_sample_data_ecommerce 索引，请使用 Kibana 中的 Revenue 仪表板浏览数据。考虑一下你可能想从此电子商务数据中获得什么见解：



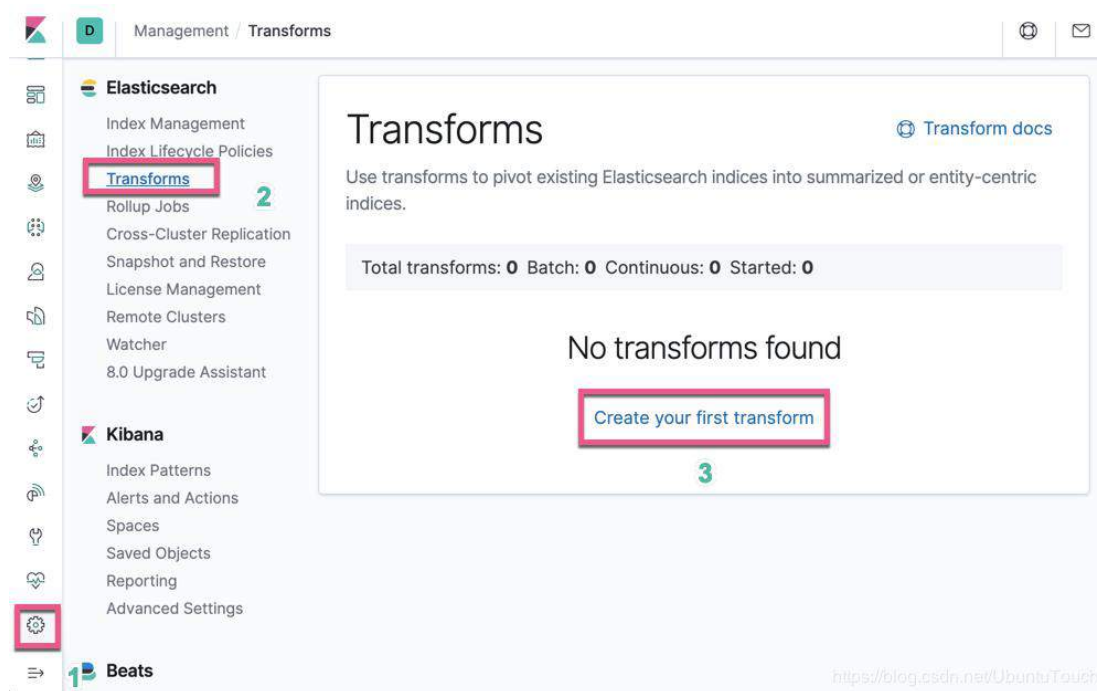


使用各种选项进行分组和汇总

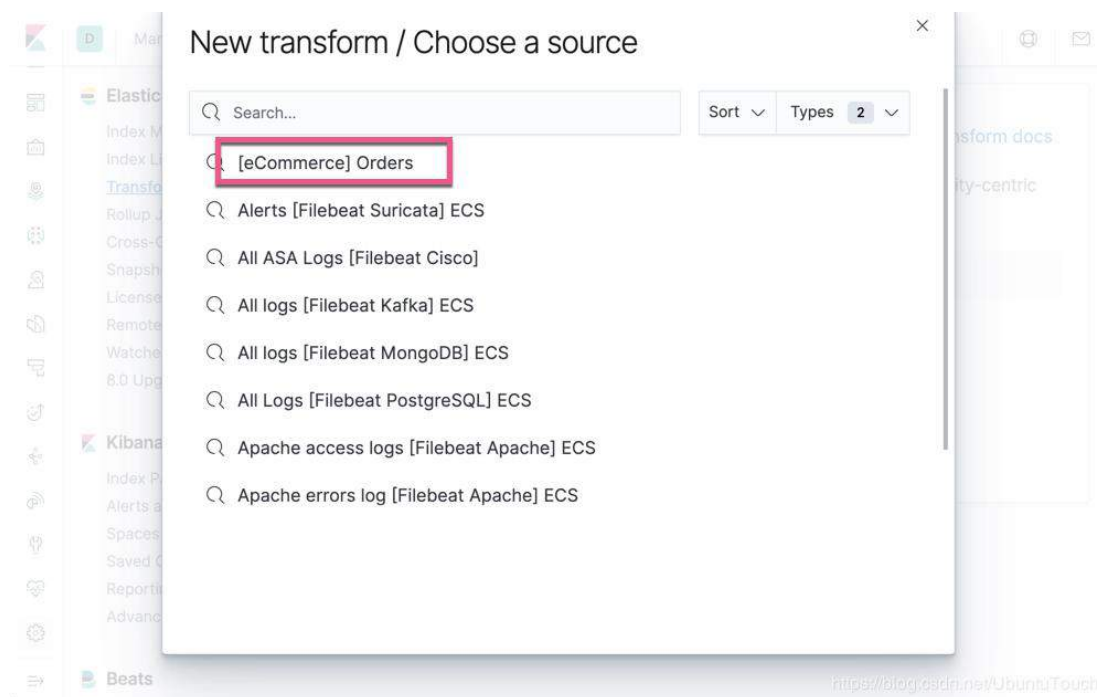
透视数据涉及使用至少一个字段对其进行分组并应用至少一项聚合。你可以预览转换后的数据，然后继续进行操作！

例如，你可能想按产品ID对数据进行分组，并计算每种产品的销售总数及其平均价格。另外，你可能希望查看单个客户的行为，并计算每个客户总共花费了多少以及他们购买了多少种不同类别的产品。或者，你可能需要考虑货币或地理位置。转换和解释这些数据最有趣的方式是什么？

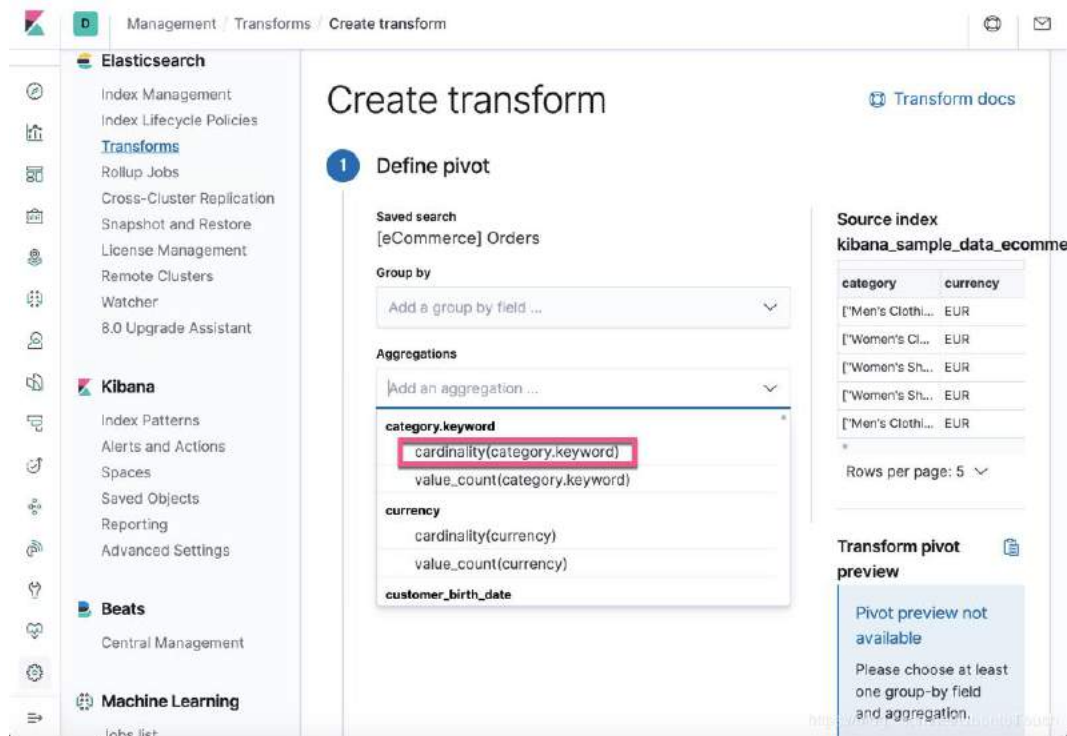
下面，我们来做一个练习。打开 Kibana：



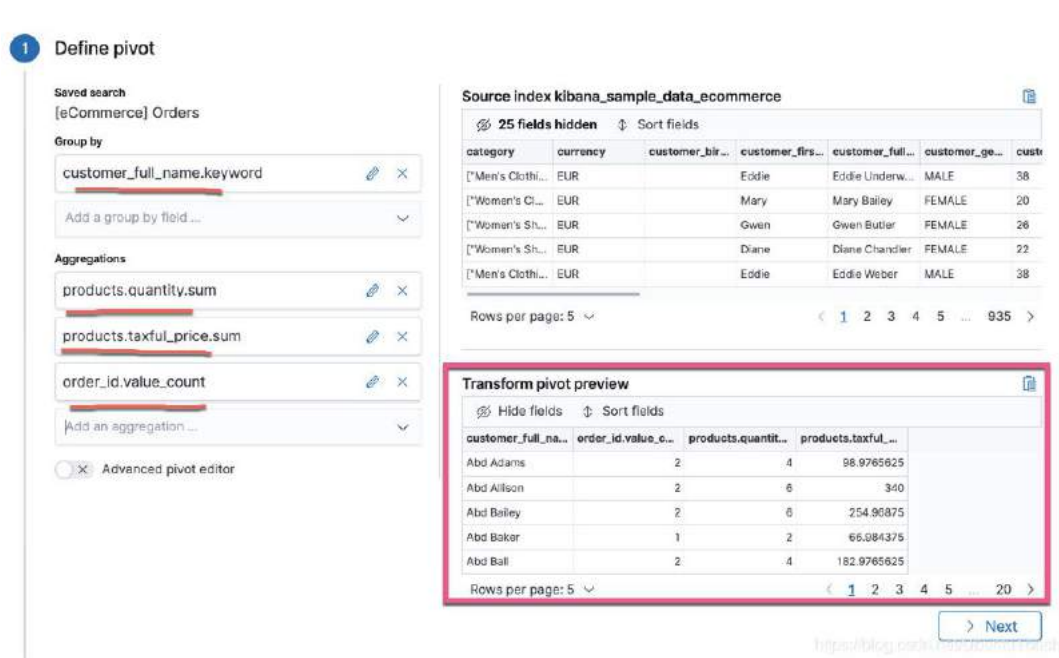
我们点击上面的 Create your first transform:



点击上面的 [eCommerce] Orders:



我们在上面进行一些感兴趣的项进行选择：



我们在屏幕的右方可以看到 Transform pivot preview。它显示的就像是一个表格的形式，而里面的数据是我们原始的数据里没的信息，比如它含有 products.quantity 的总和等。

我们点击当前页面的 Next 按钮：

[eCommerce] Orders

Group by

customer_full_name.keyword

Aggregations

products.quantity.sum

products.taxful_price.sum

order_id.value_count

transform preview

customer_full_name.keyword	order_id.value_count	products.quantity.sum	products.taxful_price.sum
Abd Adams	2	4	98.9765625
Abd Allison	2	6	340
Abd Bailey	2	6	254.96875
Abd Baker	1	2	66.984375
Abd Ball	2	4	182.9765625

Rows per page: 5

2 Transform details

Transform ID

ecommerce-customer-sales

Transform description

Summary of sales per customer

Optional descriptive text.

Destination index

ecommerce-customer-sales

☒ Create index pattern

☐ Continuous mode

< Previous

Next >

点击上面的 Next：

products.taxful_price.sum

order_id.value_count

Abd Ball

2

4

182.9765625

Rows per page: 5

2 Transform details

Transform ID

ecommerce-customer-sales

Transform description

Summary of sales per customer

Destination index

ecommerce-customer-sales

3 Create

Create and start

Create

Copy to clipboard

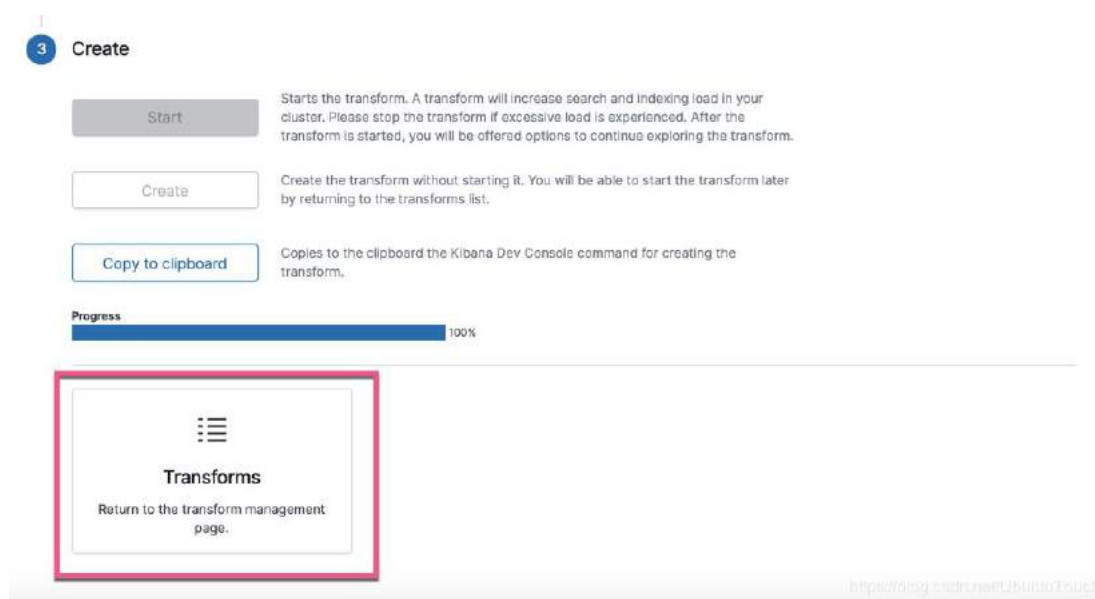
Creates and starts the transform. A transform will increase search and indexing load in your cluster. Please stop the transform if excessive load is experienced. After the transform is started, you will be offered options to continue exploring the transform.

Create the transform without starting it. You will be able to start the transform later by returning to the transforms list.

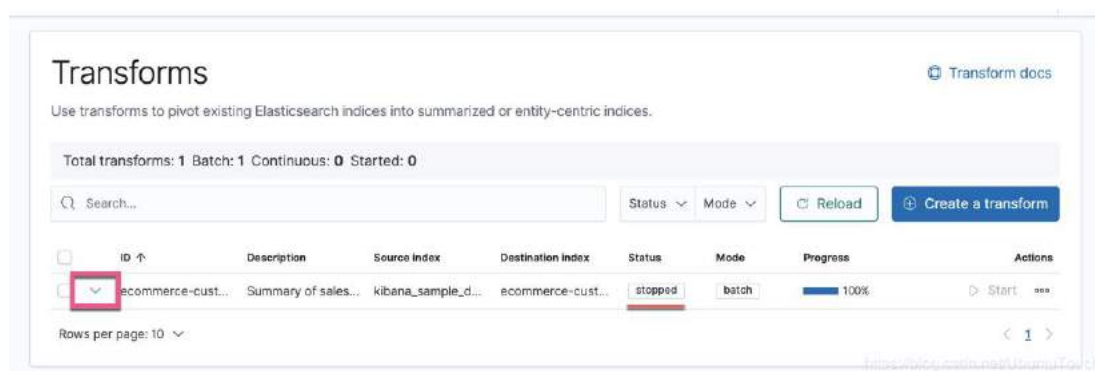
Copies to the clipboard the Kibana Dev Console command for creating the transform.

https://blog.csdn.net/ < Previous

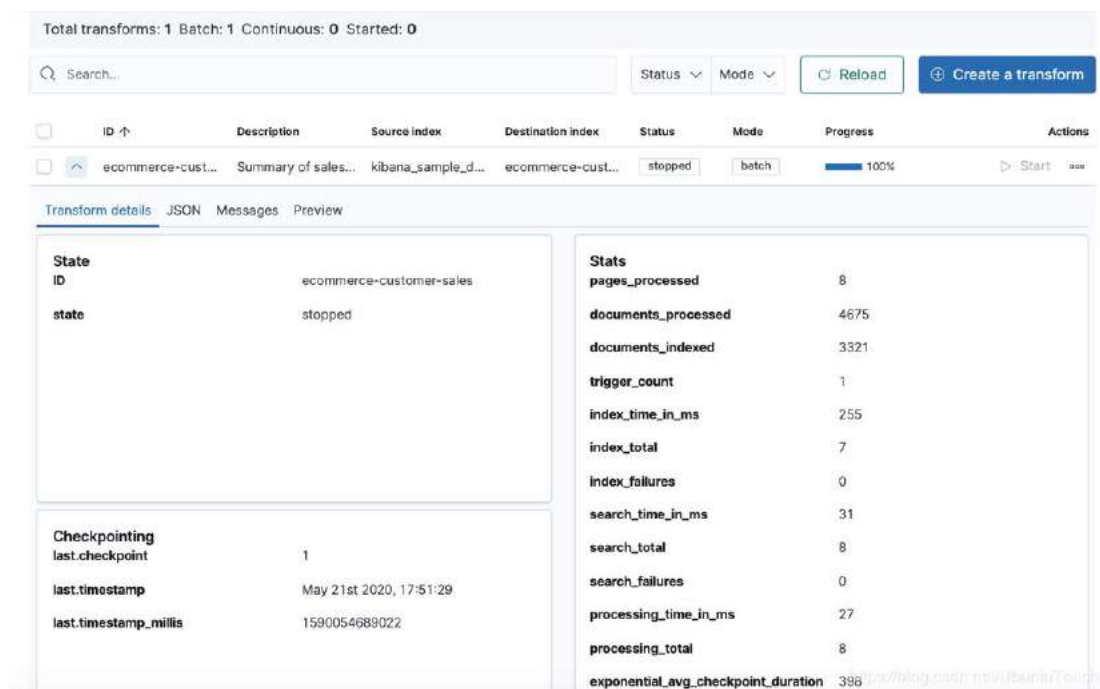
我们点击上面的 Create and start 按钮：



上面的 progress 显示 transform 的进度。已经完成100%了。点击上面的红色框，我们就可以回到 Transform 的管理页面了。

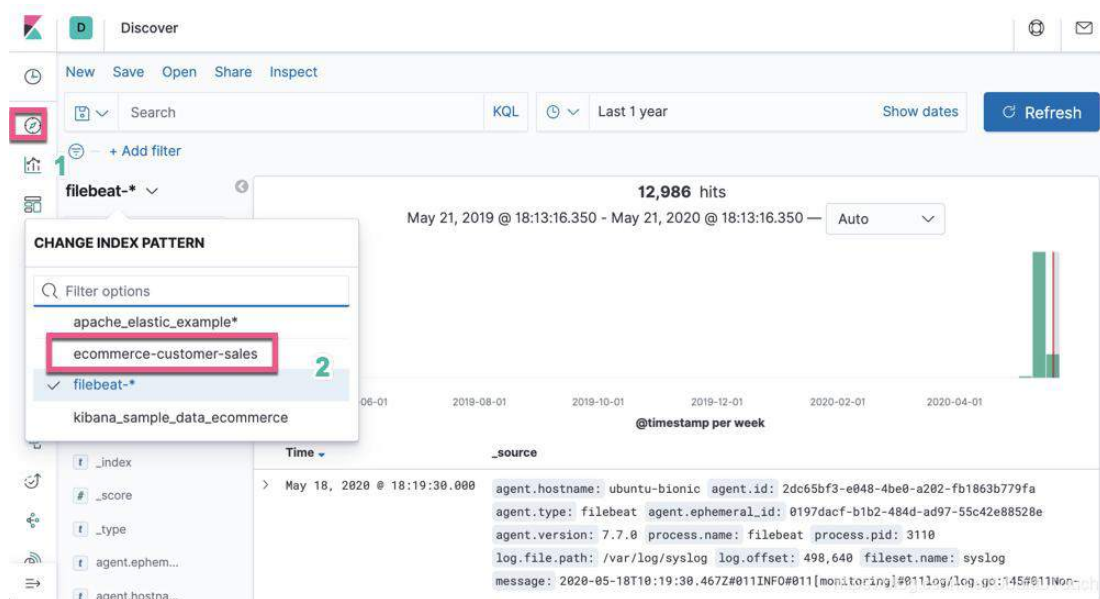


在上面，它显示我们的 Transform 已经完成了。状态是 stopped。这是因为我们的数据量还不是很大的缘故。我们点击上面的向下的扩展箭头：

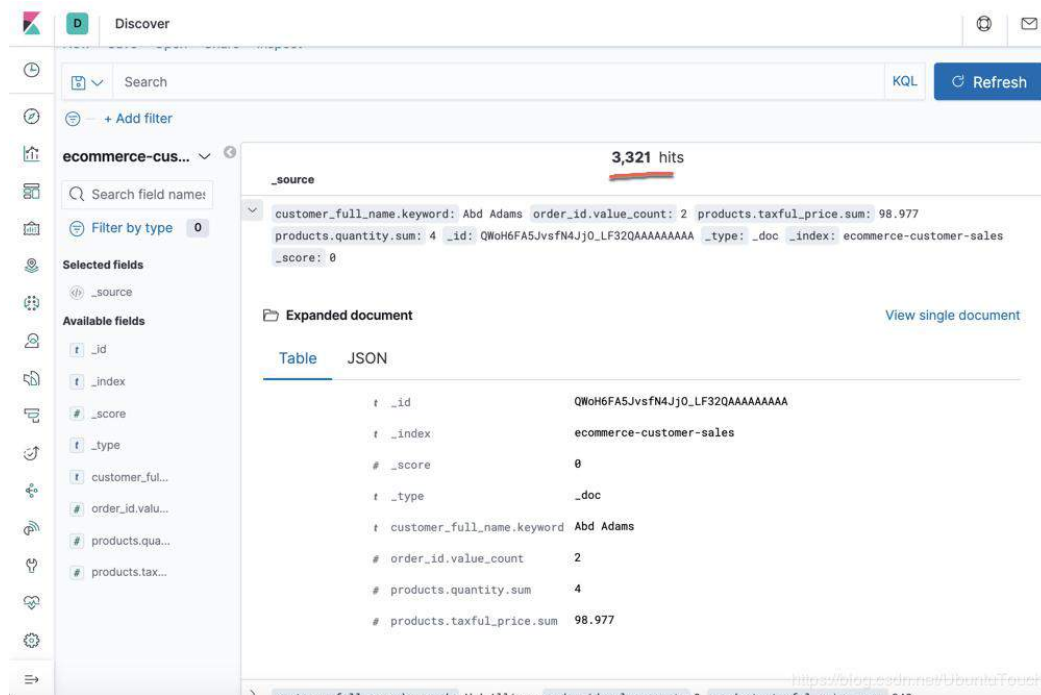


我们可以看到所有的转换的细节。

我们接下来到 Discover 中去看我们最新生产的一个索引：ecommerce-customer-sales



我们选中 ecommerce-customer-sales 索引：



在上面，我们可以看到有3321个文档，而且每个文档里含有的信息如上所示。它显示了当前用户的花费信息。我们可以针对这个所以来进行搜索。这些数据在很多时候非常有用，比如在进行机器学习时，我们可以生产这样的索引，对数据进行分析。

使用 API 来完成 transform

上面我们使用了 Kibana 中的 GUI 来完成这个工作。实际上我们也可以使用 API 的方式来完成这个工作。

我们先定义一个需要被使用的 pipeline:

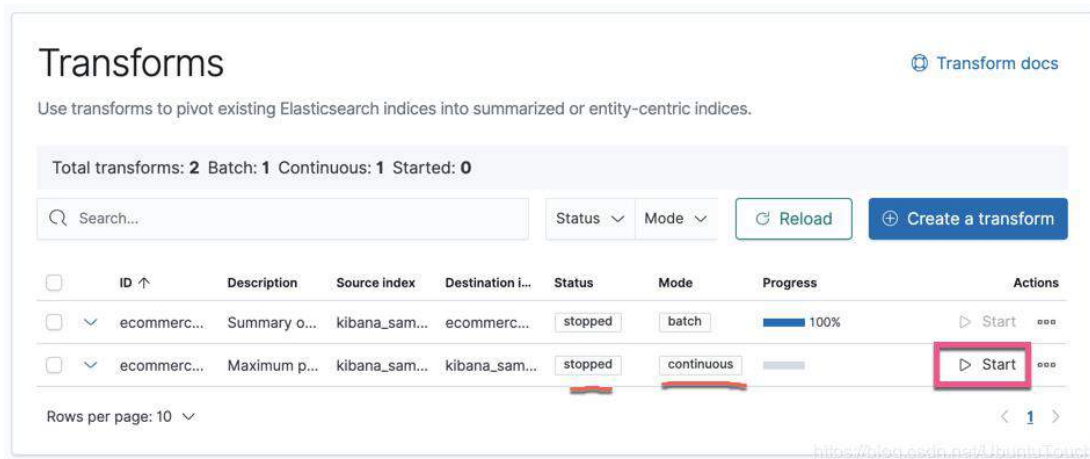
```
1. PUT _ingest/pipeline/add_timestamp_pipeline{
2.   "description": "Adds timestamp to documents",
3.   "processors": [
4.     {
5.       "script": {
6.         "source": "ctx['@timestamp'] = new Date().getTime();"
7.       }
8.     }
9.   ]
10. }
```

执行上面的指令。然后执行下面的指令：

```
1. PUT _transform/ecommerce_transform{
2.   "source": {
3.     "index": "kibana_sample_data_ecommerce",
4.     "query": {
5.       "term": {
6.         "geoip.continent_name": {
7.           "value": "Asia"
8.         }
9.       }
10.    },
11.    },
12.    "pivot": {
13.      "group_by": {
14.        "customer_id": {
15.          "terms": {
16.            "field": "customer_id"
17.          }
18.        }
19.      },
20.      "aggregations": {
21.        "max_price": {
22.          "max": {
23.            "field": "taxful_total_price"
24.          }
25.        }
26.      }
27.    },
28.    "description": "Maximum priced ecommerce data by customer_id in Asia",
29.    "dest": {
```

```
30.   "index": "kibana_sample_data_ecommerce_transform",
31.   "pipeline": "add_timestamp_pipeline"
32. },
33.   "frequency": "5m",
34.   "sync": {
35.     "time": {
36.       "field": "order_date",
37.       "delay": "60s"
38.     }
39.   }
```

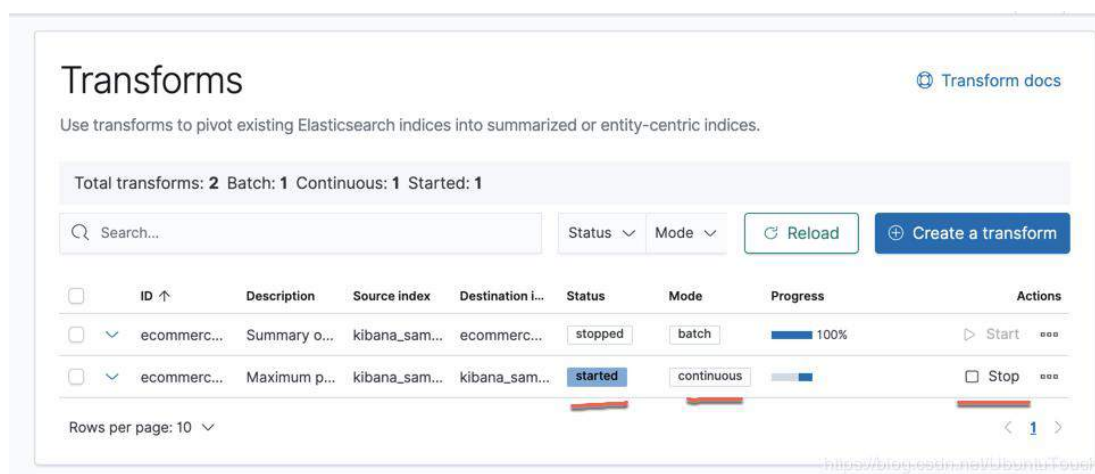
在上面，我们定义了一个 持续 transform，也就是说每隔5分钟的时间，它会检查最新的数据，并进行转换。这个在 `frequency` 里有定义。当我们执行上面的命令后，我们可以在 `transform` 的管理页面看到：



我们看到一个新的 Transform 已经生产，而且它是一个 `continuous` 的 transform。我们可以点击 `Start` 按钮来执行它。我们也可以使用如下的 API 来启动这个 transform：

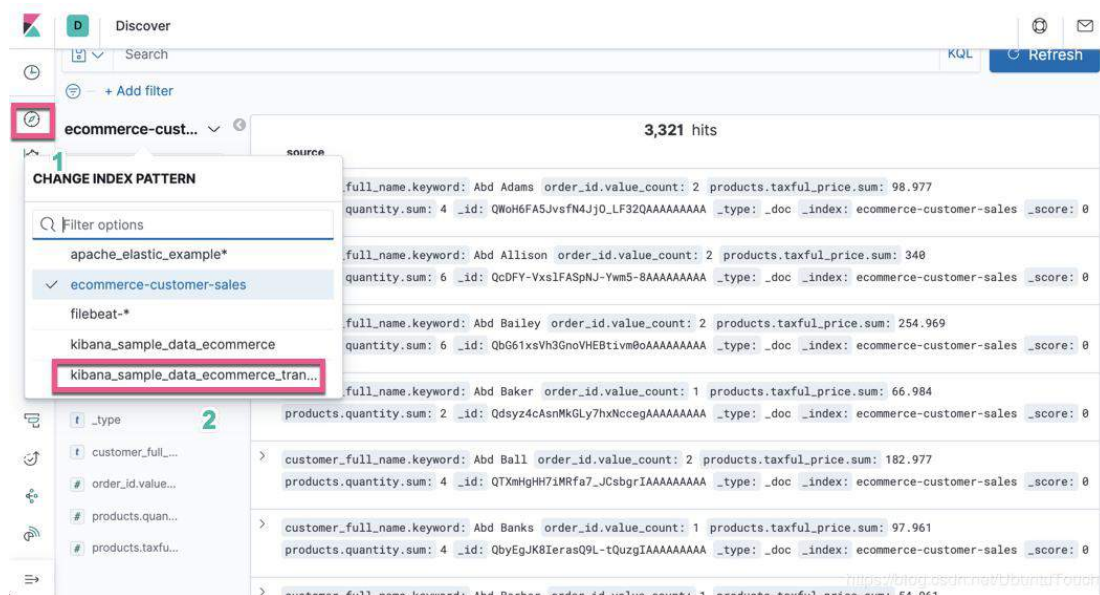
```
POST _transform/ecommerce_transform/_start
```

当我们执行完上面的命令后，我们再次查看 `transform` 的管理页面：



从上面我们可以看出来这个 transform 已经被启动，而且是一种在运行的状态。我们可以点击 Stop 来停止这个 transform，如果我们不想运行的话。

由于上面的命令没有为新创建的索引 kibana_sample_data_ecommerce_transform 创建一个 index pattern，我们需要自己手动来创建一个 index pattern。等我们创建完后，打开 Discover 来查看新的 transform 索引：





从上面我们可以看到有 13 个文档，这是因为我们只关心 Asia 的数据。所有的数据是以 customer_id 来分组的。它显示了这个 customer 的最大价格。在上面我们看到我们也有一个通过 pipeline 写入的当前时间。

有兴趣的开发者，可以尝试写入一个新的文档到 kibana_sample_data_ecommerce 索引，并且是 Asia 的，我们可以看看是否有多一个文档在 kibana_sample_data_ecommerce_transform 索引中。

我们可以通过如下的 API 来删除这个 transform:

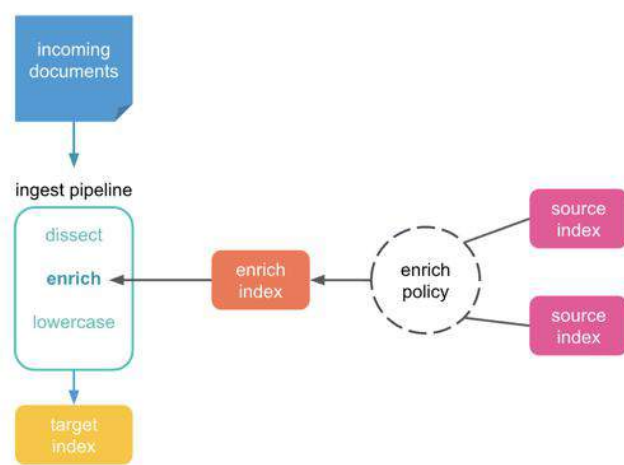
1. POST `_transform/ecommerce_transform/_stop`
2. DELETE `_transform/ecommerce_transform`

Observability: 使用 Elastic Stack 分析地理空间数据

本文作者：Elastic 中国社区布道师——刘晓国

在之前的文章 “[Elastic Stack 实现地理空间数据采集与可视化分析](#)”，我详述了如何从 OpenSky Network API 接口把数据导入到 Elasticsearch，并对这些数据进行可视化分析。也许针对很对的情况这个已经很满足了，因为它确实可以帮我们从很多实时数据中提取很多有用用的东西。

在今天的文章中，我们将参考之前的文章 “[如何使用 Elasticsearch ingest 节点来丰富日志和指标](#)”。我们可以利用 Elasticsearch ingest 节点来更加丰富我们的数据，并对这些数据做更进一步的分析。



<https://blog.csdn.net/VJbuntuTouch>

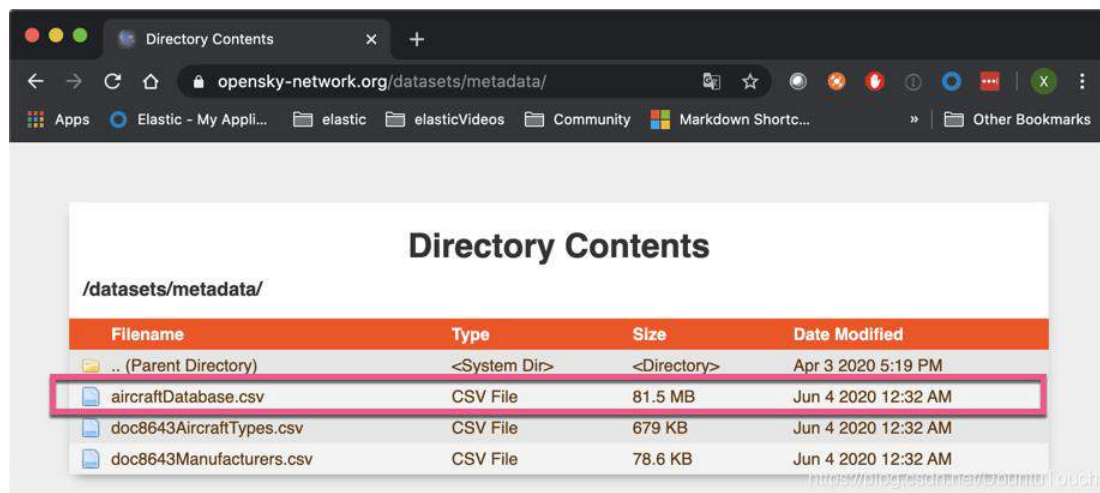
为了达到这个目的，我们必须首先了解在之前索引中的 `icao` 字段。这个字段的意思是：

ICAO 机场代码或位置指示器是由四个字母组成的代码，用于指定世界各地的机场。这些代码由国际民用航空组织定义并发布在国际民航组织7910号文件：位置指示器中，供空中交通管制和航空公司运营（例如飞行计划）使用。

我们之前的每个文档是这样的：


```
1. {
2.   "velocity": 0.0,
3.   "icao": "ad0851",
4.   "true_track": 264.38,
5.   "time_position": 1591190152,
6.   "callsign": "AAL2535",
7.   "origin_country": "United States",
8.   "position_source": "ADS-B",
9.   "spi": false,
10.  "request_time": 1591190160,
11.  "last_contact": 1591190152,
12.  "@timestamp": "2020-06-03T13:16:03.723Z",
13.  "on_ground": true,
14.  "location": "32.7334,-117.2035"
15. }
```

另外，我们可以在地址 <https://opensky-network.org/datasets/metadata/> 找到一个如下文件：



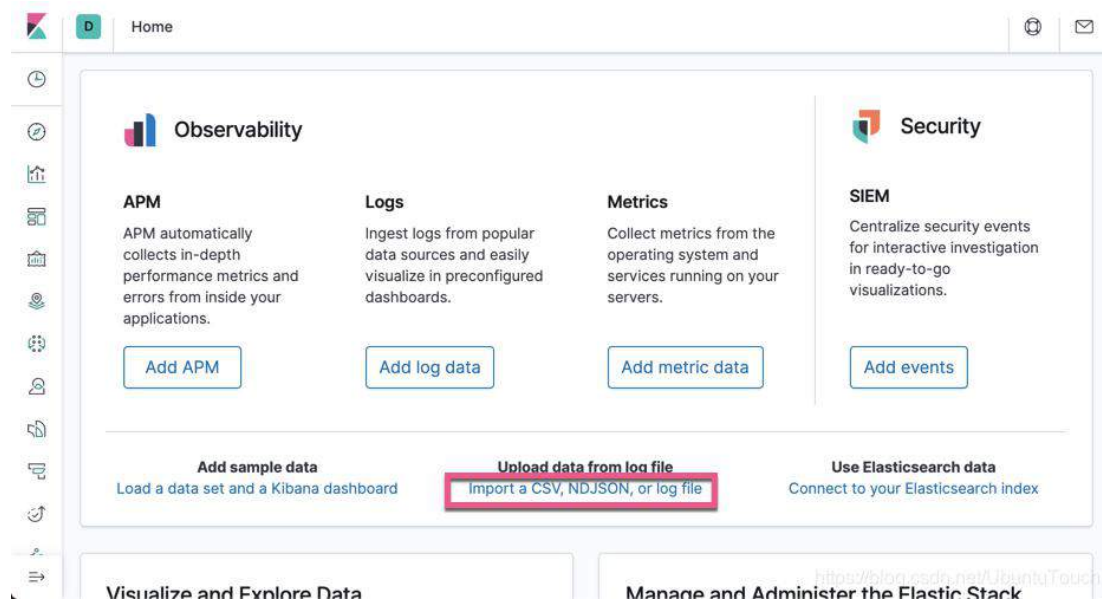
在这里，我们可以找到一个叫做 aircraftDatabase.csv 的文件。它里面的内容如下：

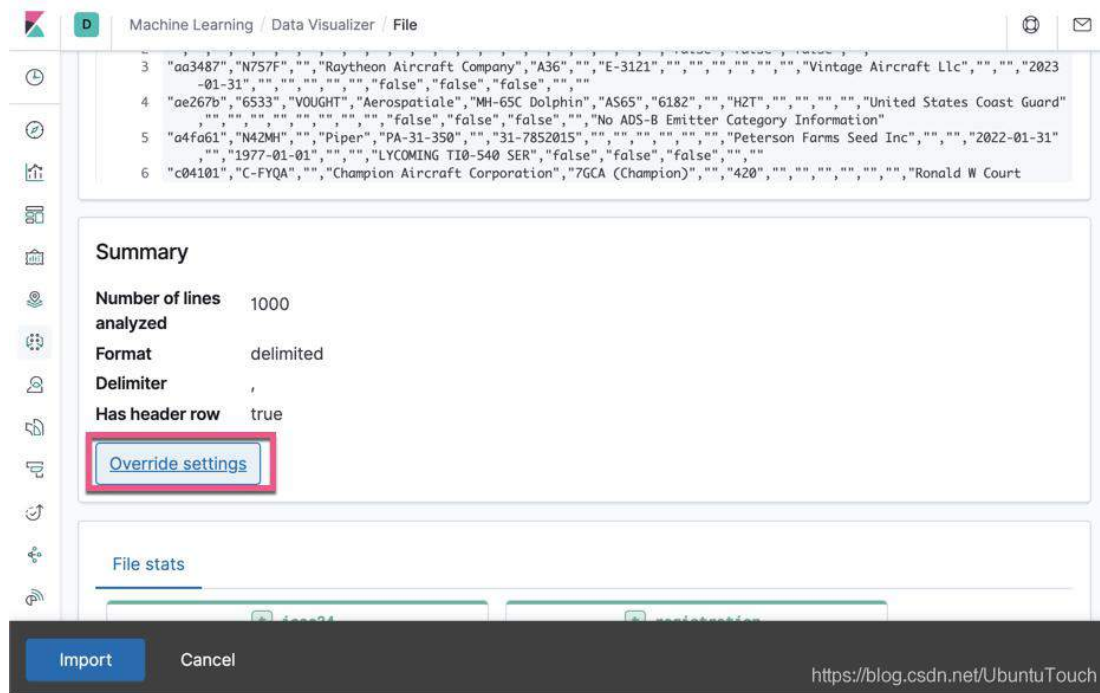
1	icao24	registration	manufacturericao	manufacturername	model	typecode	serialnumber	linenumb
2								
3	aa3487	N757F		Raytheon Aircraft Company	A36		E-3121	
4	ae267b	6533	VOUGHT	Aerospaiale	MH-65G Dolphin	AS65	6182	
5	a4fa61	N42MH		Piper	PA-31-350		31-7852015	
6	c04101	C-FYQA		Champion Aircraft Corporation	7GGA (Champion)		420	
7	391927	F-GGJH	ROBIN	Robin	DR.400 160 Chevalier	DR40	1795	
8	a61c16	N493TR	CIRRUS	Cirrus Design Corp	SR22T	S22T	776	
9	503c21	LY-KNA		Impulse Aircraft	Impulse 100	ZZZZ		
10	aa6735	N77FK	GULFSTREAM AEROSPACE	Gulfstream Aerospace	GIV SP	GLF4	1357	
11	3d3191	D-ERAF	GROB	Grob	G-115 E	G115	82085/E	
12	abc886	N8587J		Cesana	150G		15068487	
13	abb55c	N85385		Aeronca	7AC		7AC-4074	
14	a12b4e	N17462		Waco	YKS-7		4604	
15	3d1391	D-EFQV	DORNIER	Dornier	Do 27 B-1	DO27	175	
16	a64fa4	N50551		Cesana	150J		15069392	
17	ab218f	N816VP		Powrachute Llc	PEGASUS		A512PEG	

在上面的表格中，我们发现有一个叫做 icao24 的字段。这个字段和我们之前的文档可以进行关联，从而我们可以得到更多关于某个航班的更多信息。

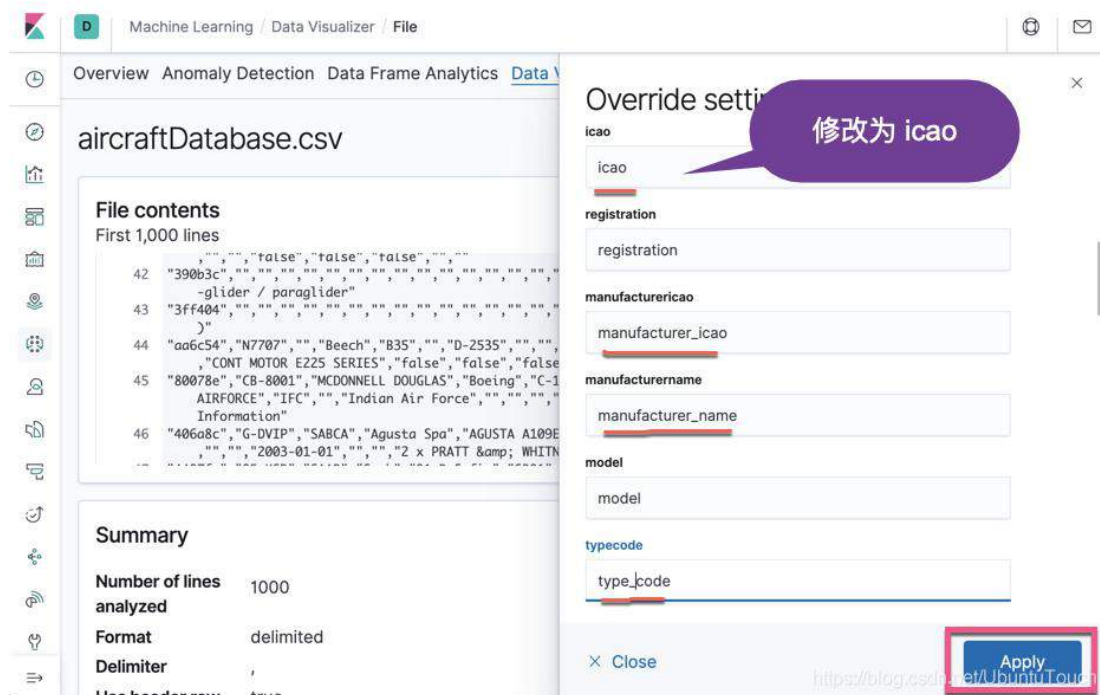
创建 enrich index

由于下载的文档时一个是一个 csv 的文件。我们可以使用 data visualizer 来导入。

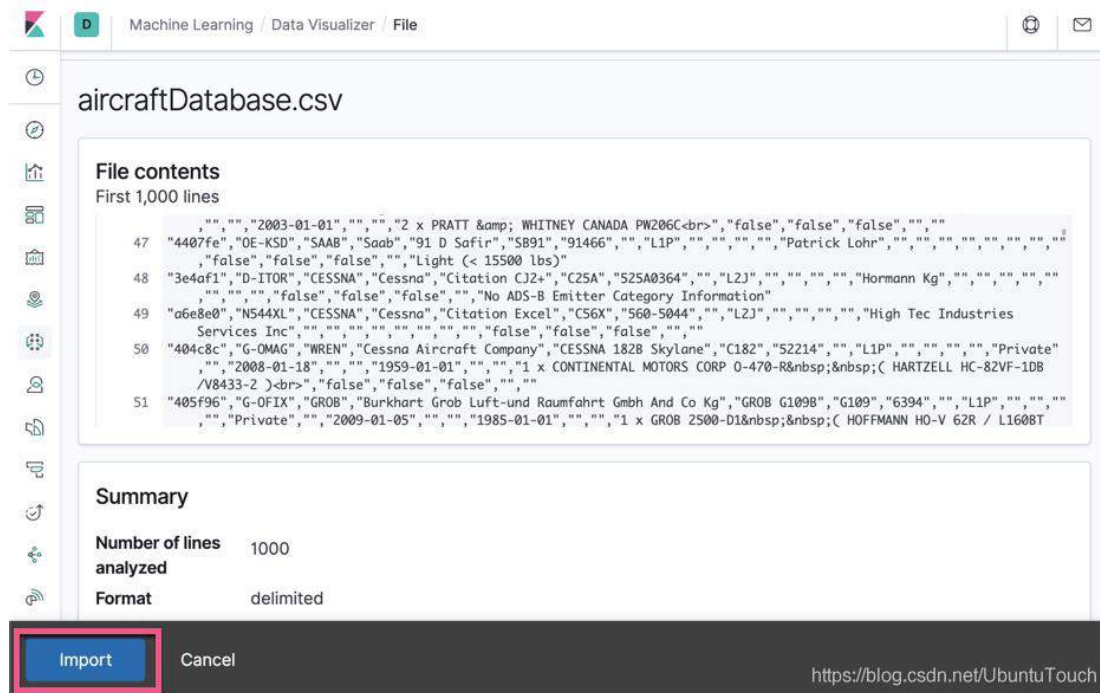




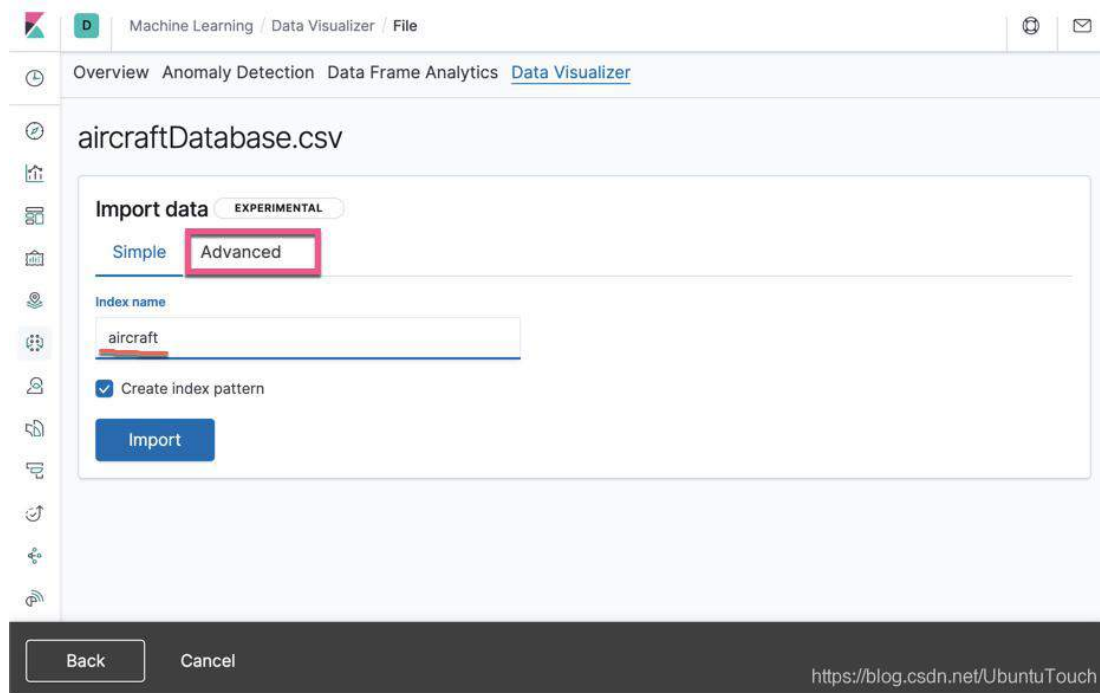
点击上面的 Override settings 链接:



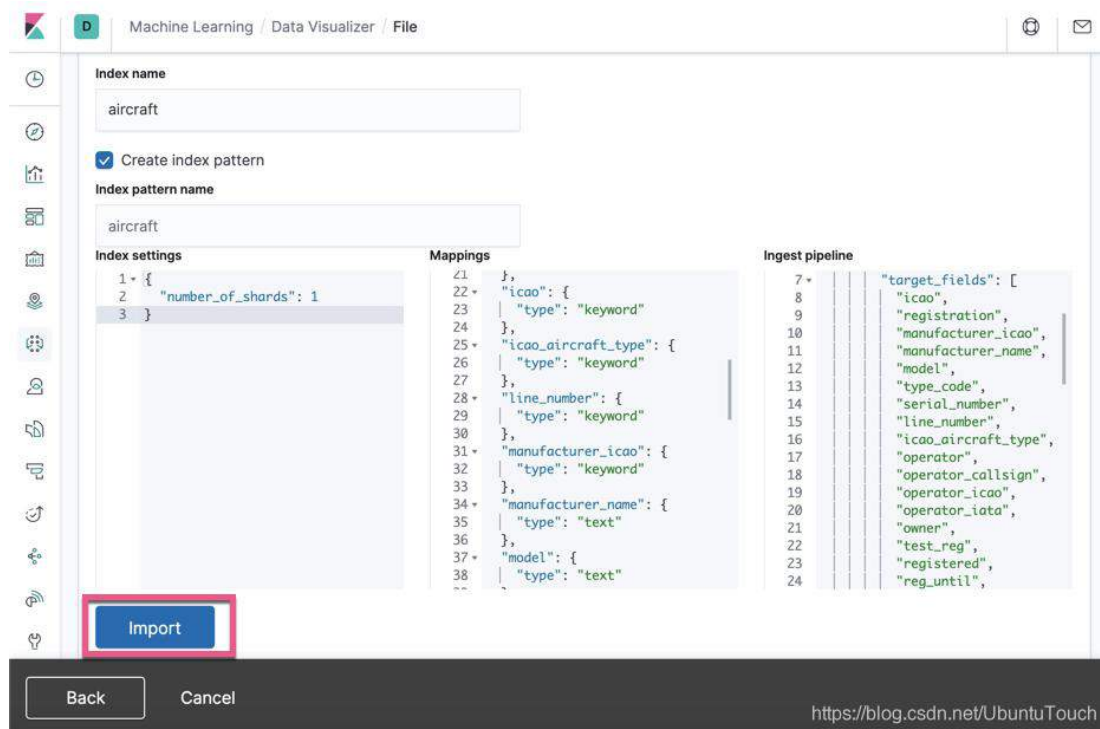
点击 Apply 按钮:



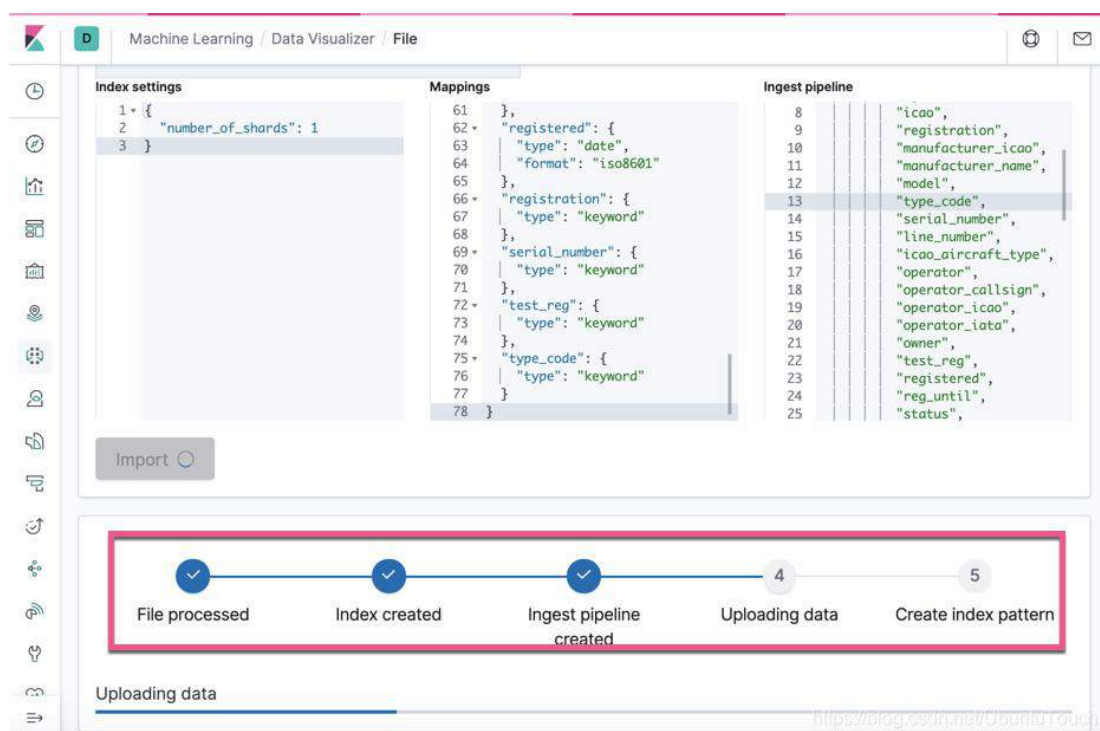
点击上面的 Import 按钮：



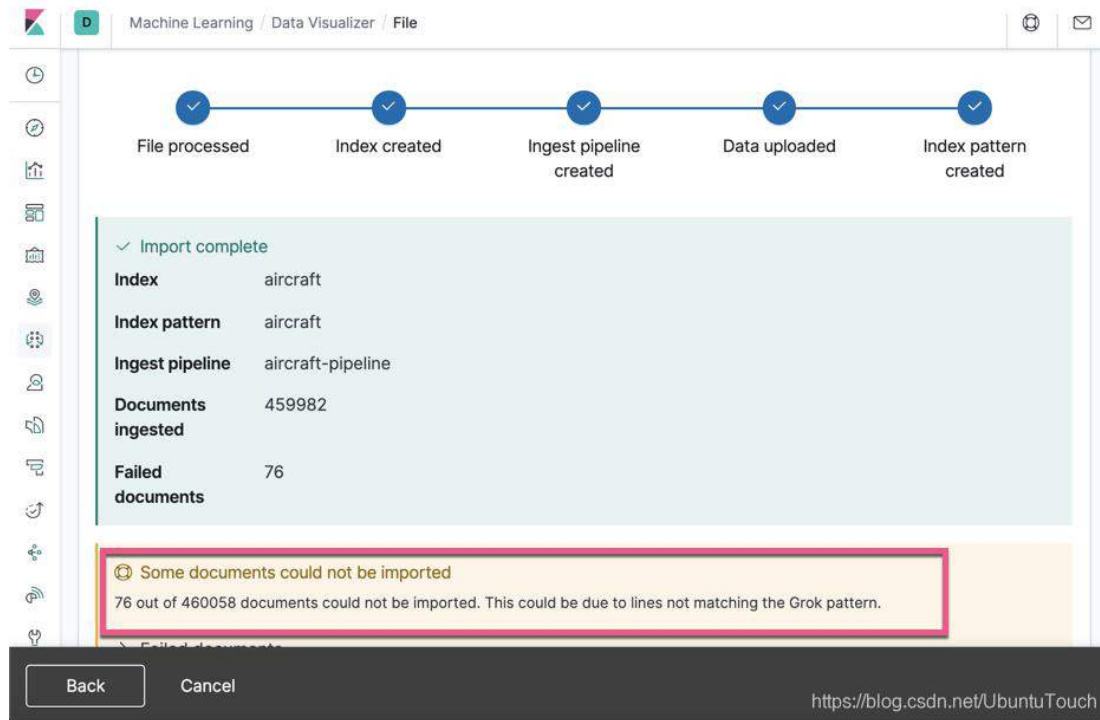
我们把这个索引的名称称为 aircraft。点击 Advanced:



再次确认 mapping，如果没有问题的话，点击 Import 按钮：



由于这个文件比较大，所以需要一点时间来进行导入：



等完成后，我们可以在 Elasticsearch 中找到一个叫做 aircraft 的索引：

GET _cat/indices

1	green	open	.security-7	dFooTYaQT9a8VMKpNJAwtA	1	0	53	3	84kb	84kb
2	green	open	.logstash	1r8o4E_-TdejIsVzZttFug	1	0	1	0	6.3kb	6.3kb
3	green	open	.apm-custom-link	44W9JLw7RuSjK-n6M0Laeg	1	0	0	0	208b	208b
4	green	open	.kibana_task_manager_1	9hykvAnkQqKGv4nkq4f1bw	1	0	5	3	63.1kb	63.1kb
5	yellow	open	aircraft	ENJTok9hR1yo9gkX21854g	1	1	459982	0	86.9mb	86.9mb
6	green	open	.apm-agent-configuration	VdVqHQxbQuegZy-7hyGqEg	1	0	0	0	208b	208b
7	green	open	.async-search	ws4vrw6ASeg-soGqiT9WDA	1	0	6	5	205.7kb	205.7kb
8	yellow	open	flights	V1YsIkrISReg5bImZCNDlg	1	1	1803570	0	343.5mb	343.5mb
9	green	open	.kibana_1	5yP0-PG7S2Ckj1T05eAx3w	1	0	316	5	158.9kb	158.9kb
10										

<https://blog.csdn.net/UbuntuTouch>

上面显示有一个新的 aircraft 的索引生成了。

创建 Enrich policy

接下来，我们来创建 enrich policy。它告诉我们如何丰富数据。在 Kibana 中打入如下的命令：

1. PUT /_enrich/policy/**flights_policy**

2. {

3. "match": {

4. "enrich_fields": [

5. "acars",

6. "adsb",

7. "built",

8. "category_description",

9. "engines",

10. "first_flight_date",

11. "icao_aircraft_type",

12. "line_number",

13. "manufacturer_icao",

14. "manufacturer_name",

15. "model",

16. "modes",

17. "notes",

18. "operator",

19. "operator_callsign",

20. "operator_iata",

21. "operator_icao",

22. "owner",

23. "reg_until",


```
24. "registered",
25. "registration",
26. "seat_configuration",
27. "serial_number",
28. "status",
29. "test_reg",
30. "type_code"
31. ],
32. "indices": [
33. "aircraft"
34. ],
35. "match_field": "icao"
36. }
37.}
```

我们使用 **execute enrich policy API** 为该策略创建enrich索引：

```
POST /_enrich/policy/flights_policy/_execute
```

接着，我们创建一个叫做 flights_aircraft_enrichment 的 pipeline：

```
1. PUT /_ingest/pipeline/flights_aircraft_enrichment

2. {

3.   "description": "joins incoming ADSB state info with richer aircraft metadata",

4.   "processors": [

5.     {

6.       "enrich": {

7.         "field": "icao",

8.         "policy_name": "flights_policy",

9.         "target_field": "aircraft"
```

到此为止，我们已经成功地创建了 丰富策略。接下来，我们将展示如何使用这个 pipeline 来丰富我们的数据。

丰富数据

为了能够使用我们上面定义好的 pipeline，我们重参考之前的文章 “[Observability: 使用 Elastic Stack 分析地理空间数据（一）](#)” 里的 `flights_logstash.conf` 文件，并修改如下的 output 部分：

```
1.  output {
2.    stdout {
3.      codec => rubydebug
4.    }
5.
6.    elasticsearch {
7.      manage_template => "false"
8.      index => "flights"
9.      # pipeline => "flights_aircraft_enrichment"
10.     hosts => "localhost:9200"
11.   }
12. }
```

我们把上面的这一行的注释拿掉：

```
# pipeline => "flights_aircraft_enrichment"
```

这样变成了：

```
1.  output {
2.    stdout {
3.      codec => rubydebug
4.    }
```

在启动 Logstash 之前，我们可以先删除之前的 flights 索引：

```
DELETE flights
```

再接着执行如下的命令：

```
1.  PUT flights
2.  {
3.    "mappings": {
4.      "properties": {
5.        "@timestamp": {
6.          "type": "date"
7.        },
8.        "baro_altitude": {
9.          "type": "float"
```

```
10. },
11. "callsign": {
12. "type": "keyword"
13. },
14. "geo_altitude": {
15. "type": "float"
16. },
17. "icao": {
18. "type": "keyword"
19. },
20. "last_contact": {
21. "type": "long"
22. },
23. "location": {
24. "type": "geo_point"
25. },
26. "on_ground": {
27. "type": "boolean"
28. },
29. "origin_country": {
30. "type": "keyword"
31. },
32. "position_source": {
33. "type": "keyword"
34. },
35. "request_time": {
36. "type": "long"
37. },
38. "spi": {
39. "type": "boolean"
40. },
41. "squawk": {
42. "type": "long"
43. },
44. "time_position": {
45. "type": "long"
46. },
47. "true_track": {
```

```

48. "type": "float"
49. },
50. "velocity": {
51. "type": "float"
52. },
53. "vertical_rate": {
54. "type": "float"
55. }
56. }
57. }
58. }

```

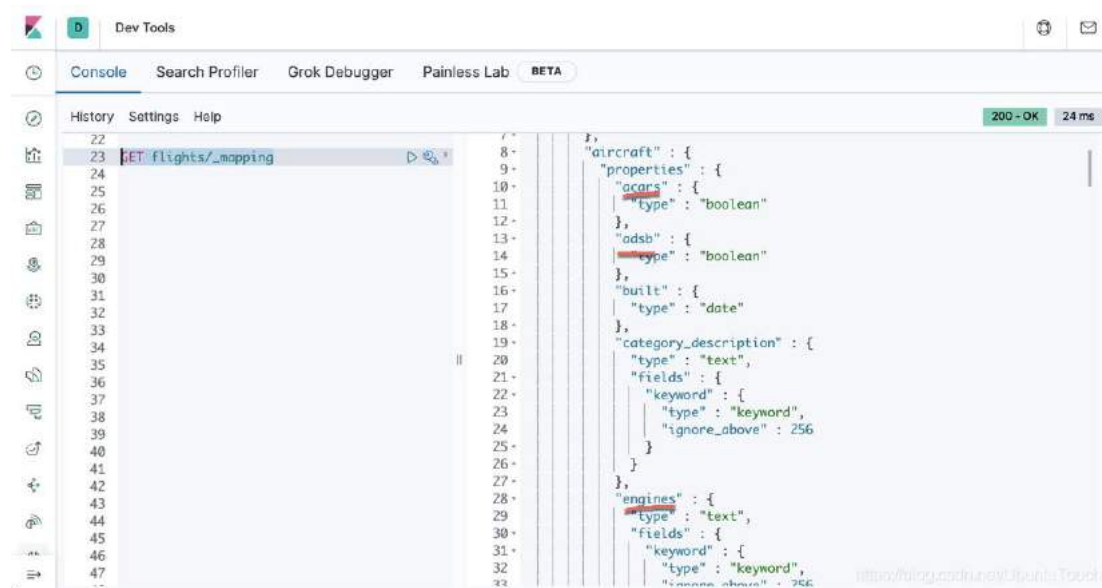
重新运行 Logstash:

```
sudo ./bin/logstash -f flights_logstash.conf
```

我们在 Kibana 中检查 flights 的 mapping:

```
GET flights/_mapping
```

我们可以看到一些新增加的各个新字段:



我们可以通过 search:

```
1.  "_source" : {
2.  "aircraft" : {
3.  "owner" : "Wells Fargo Trust Co Na Trustee",
4.  "reg_until" : "2021-04-30",
5.  "modes" : false,
6.  "built" : "1984-01-01",
7.  "acars" : false,
8.  "manufacturer_icao" : "BOEING",
9.  "serial_number" : "23018",
10. "manufacturer_name" : "Boeing",
11. "icao_aircraft_type" : "L2J",
12. "operator_callsign" : "GIANT",
13. "operator_icao" : "GTI",
14. "engines" : "GE CF6-80 SERIES",
15. "icao" : "a8a763",
16. "registration" : "N657GT",
17. "model" : "767-281",
18. "type_code" : "B762",
19. "adsb" : false
20. },
21. "true_track" : 272.81,
22. "velocity" : 5.14,
23. "spi" : false,
24. "origin_country" : "United States",
25. "@timestamp" : "2020-06-04T10:41:00.558Z",
26. "request_time" : 1591267250,
27. "time_position" : 1591267168,
28. "last_contact" : 1591267168,
29. "callsign" : "GTI165",
30. "icao" : "a8a763",
```

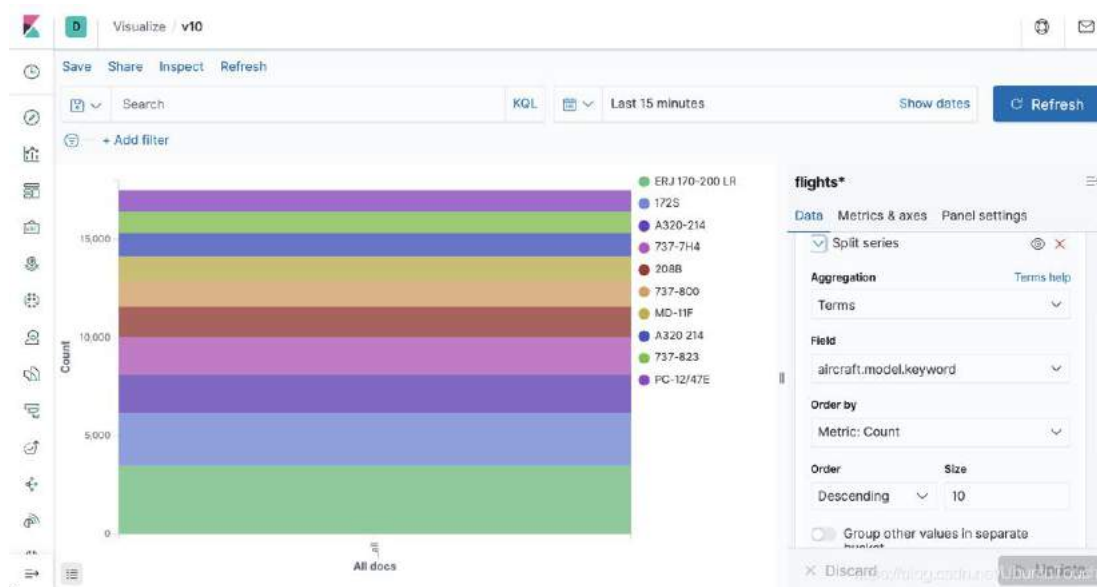
```
31. "location" : "39.0446,-84.6505",
32. "n_ground" : true,
33. "position_source" : "ADS-B"
34. }
35. }
```

我们可看到一个叫做 aircraft 的字段，它含有这个飞机所有被丰富的信息。

运用 Kibana 分析数据

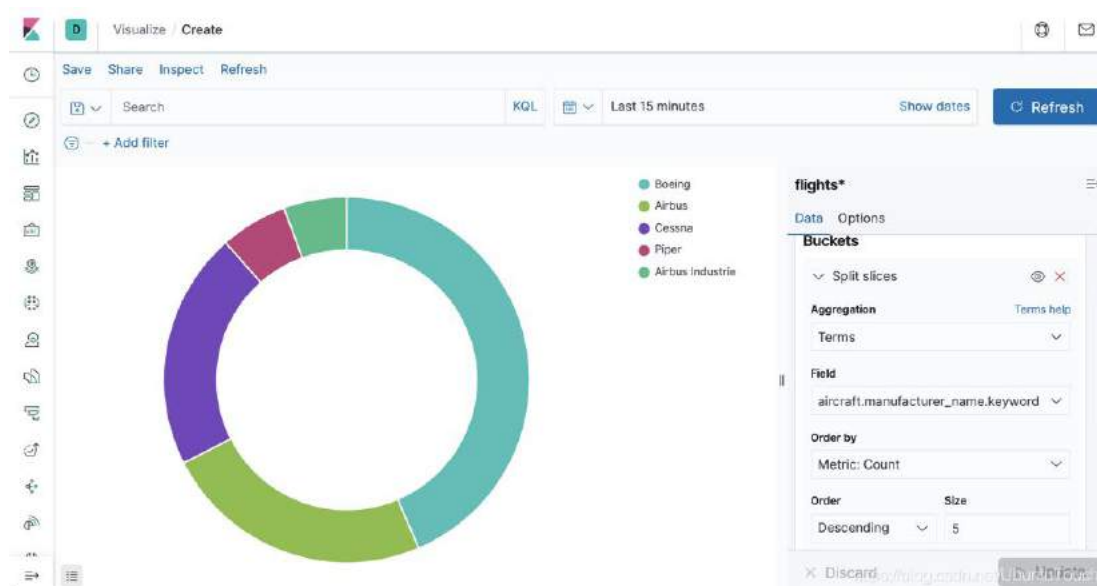
找出前10的飞机型号

因为有了新的字段进来，所以我们必须重新创建新的 index pattern:



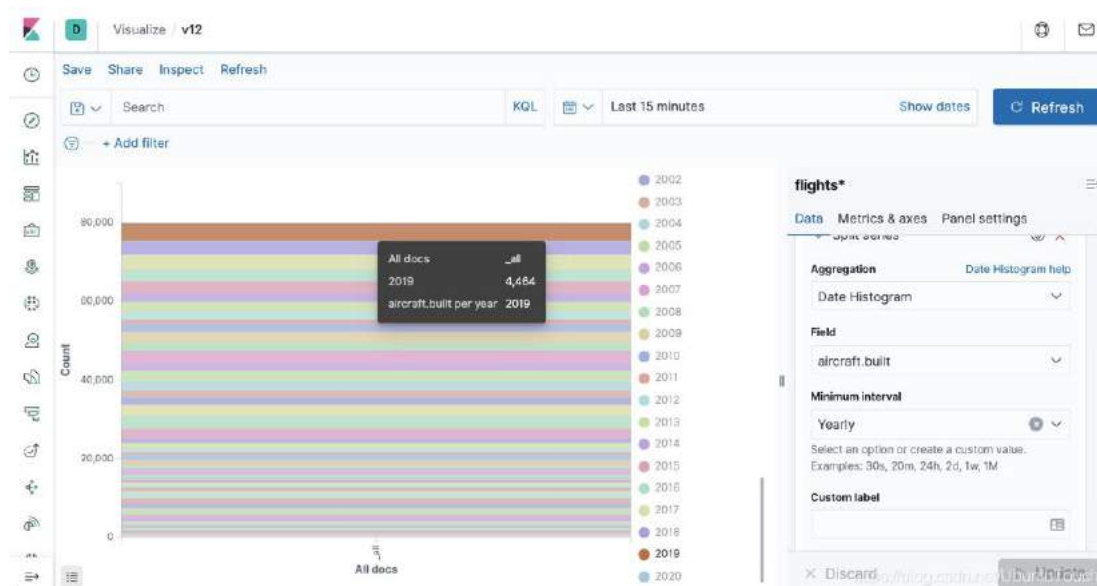
我们可以看到最多的是 PC-12/47E 这个机型。

找出飞机制造商的分布



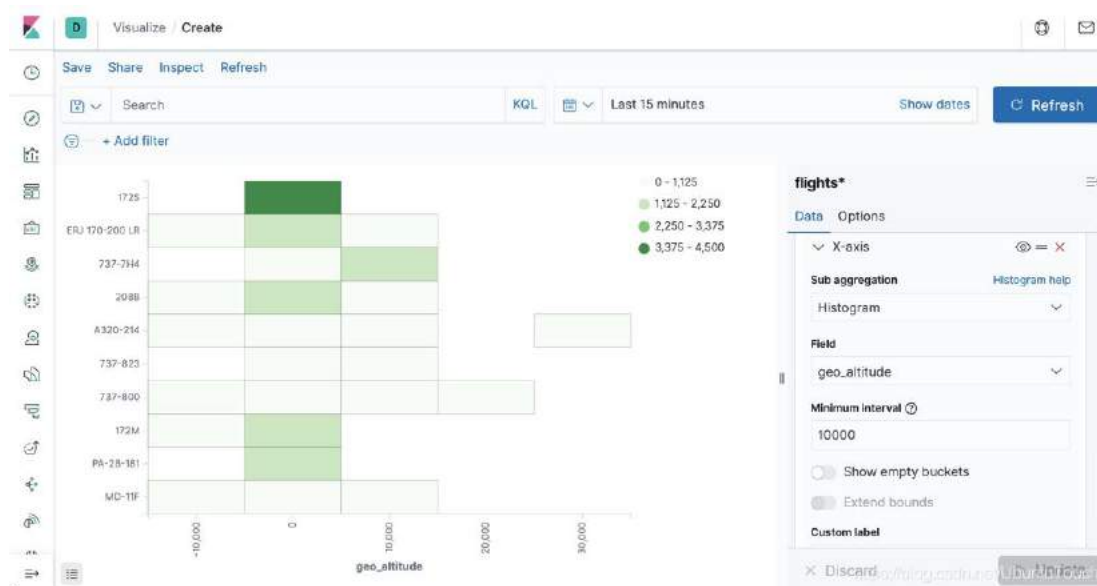
我们看到 BOING 公司的市场份额是最大的。AIRBUS 处于第二的位置。

飞机机龄分布



我们可以看出来最多的飞机是2019年生产的。

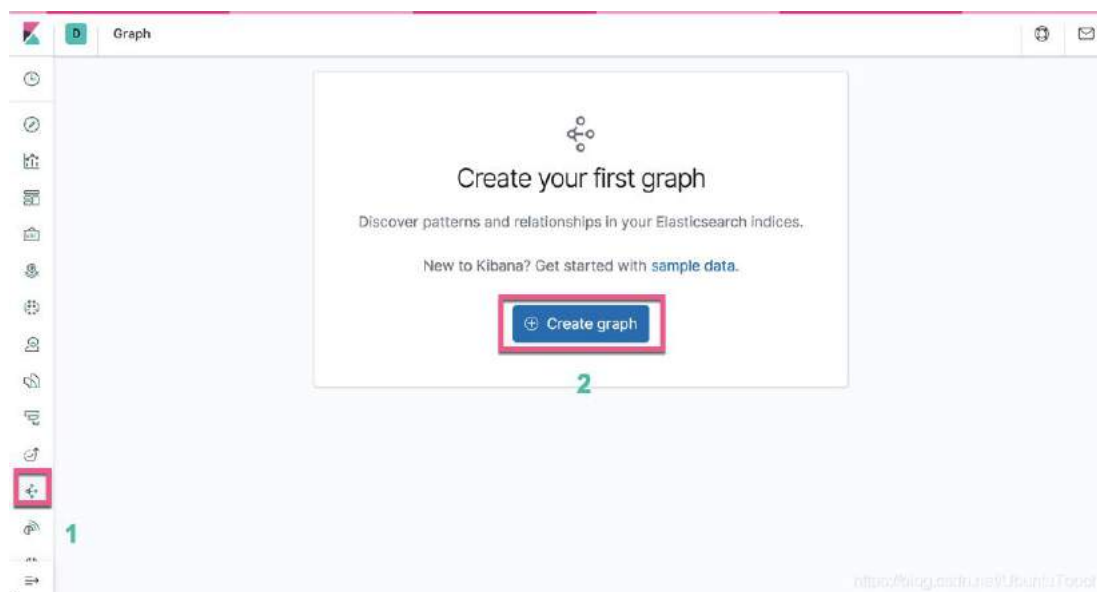
飞机机型和飞行高度的关系



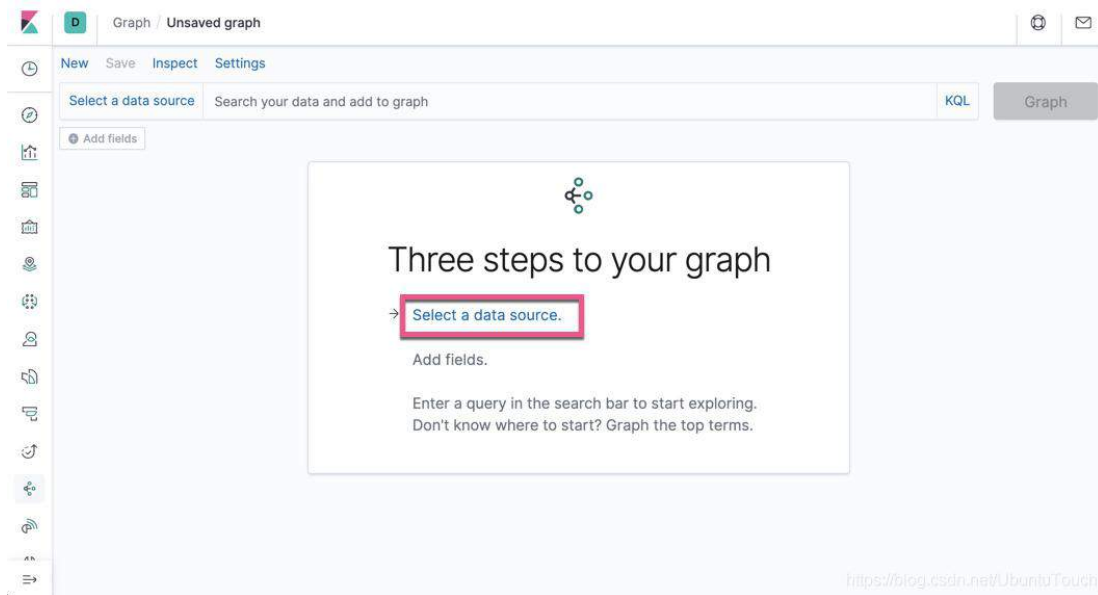
可以看出来 A320-214 飞机飞的是最高的。

Graph

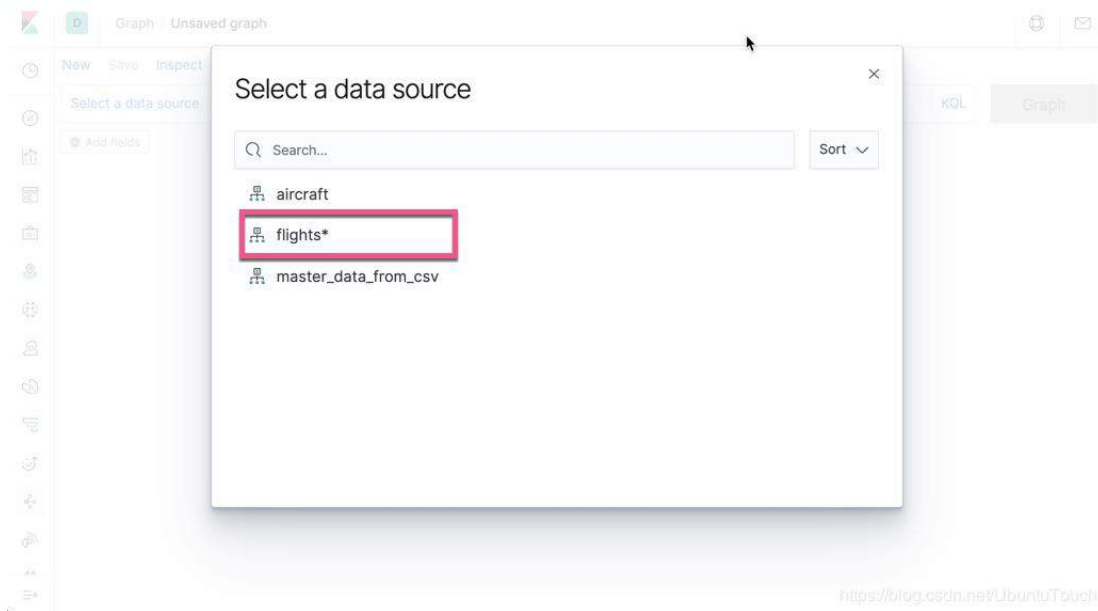
运用 Graph 来找出数据直接的关系。



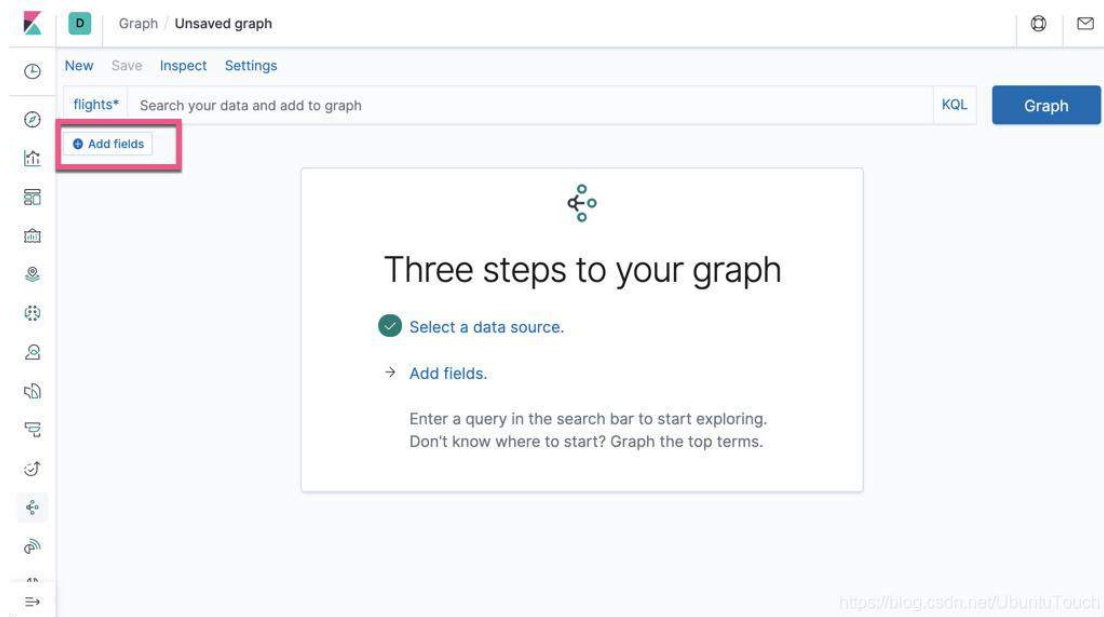
点击 Create graph:



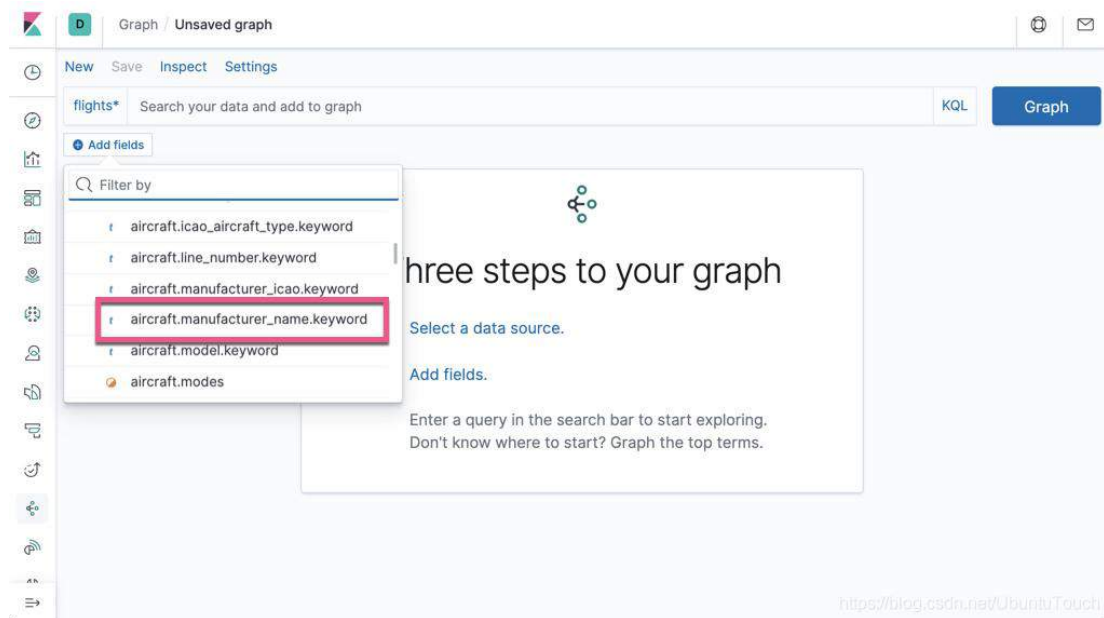
点击 Select a data source:



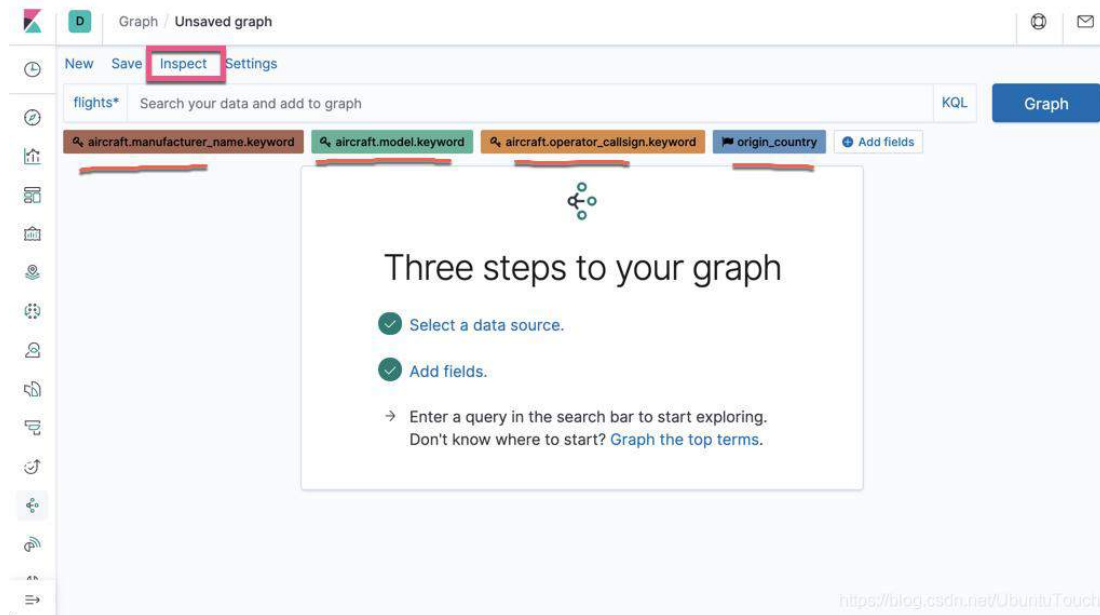
选择 flights* :



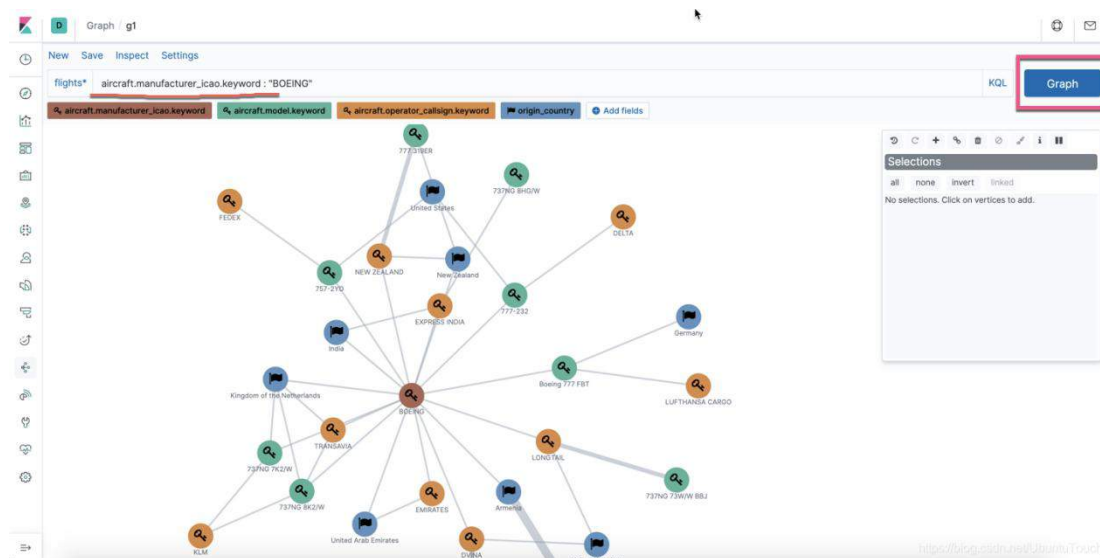
点击 Add fields:



添加 fields:



我们需要保持这个 graph。然后进行搜索:



从上面，我们可看出来 BOING 和我们想要的各个字段之间的关系。

我们从收集的数据可以有更多的其它的分析。在这里，我就不一一枚举了。你们可以做任何你想要的分析。

阿里云 Elasticsearch 向量检索，轻松玩转29个业务场景

简介：我们知道，市面上有不少开源的向量检索库供大家选择使用，例如 Facebook 推出的 Faiss 以及 Nswlib，虽然选择较多，但业务上需要用到向量检索时，依旧要面对四大共性问题。

本文作者：清豆 — 阿里巴巴高级开发工程师

您将了解：

- 1、开源向量检索库存在的问题
- 2、源自阿里Proxima向量检索库的优势
- 3、阿里云向量检索的技术原理
- 4、阿里云向量检索的应用场景
- 5、如何使用阿里云向量检索



寻根问源



图 1：开源向量检索库面临的问题

我们知道，市面上有不少开源向量检索库供大家选择使用，例如 Facebook 推出的 Faiss 以及 Nswlib，虽然选择较多，但当业务上需要用到向量检索时，依旧要面对四大共性问题。

1. 离线方案，增量不可查

市面上大多向量检索库都是对增量数据不友好，有的需要做一次性全量数据来构建索引，有的需要用户接受很少的增量数据，才能保证检索准确率。在业务随时都有实时增量数据的情况下，每次都要重建全量，并利用线上低峰期来切换最新的全量索引。比如在每天晚上12点，我们做一次最新的全量索引，那么你能只能查到前一天的增量数据。

2. 无 Failover 及分布式能力

由于市面上的向量索引库，往往不会考虑分布式场景的使用问题。在索引落盘前，任何一步出现机器故障，比如进程宕机，都会导致索引完全丢失并重新创建一遍。而且随着我们的业务增长数据越来越多，需要使用分布式来做索引构建，是无法任由这些库来满足。

3. 性能弱

对 Elasticsearch 熟悉的同学会知道，X-pack 在 7.x 推出了 dense-vector 的向量检索字段，但对于前两个问题依旧没有解决，在我们实际测试中发现，dense-vector 这个字段，或者说向量检索，是基于线性暴力计算来实现的，基本原理是对每个文档，计算与你 query 的相似度，延时随着数据量增长而成正比例函数关系，在实测2KW的向量数据时，dense-vector 召回时间在 10s 左右。那么在用 X-pack 的向量字段做人脸识别时，对于10秒的响应时间，相信大家是无法接受的。

4. 额外学习成本

由于大多向量检索库是运用 C++ 来实现的，有些也提供了 Python 的接口。而 C++ 的学习成本大家知道，且非常难上手，而且这种重客户端的模式不管是灵活性还是可维护性，都没有办法跟 REST API 交互的方式相媲美。

优势介绍



图 2：阿里云向量检索优势

1、以阿里云 Elasticsearch 插件的形式使用，一键安装。同时由于封装了阿里巴巴达摩院自研的 Proxima 高性能向量检索库。该向量检索库已经应用于阿里集团多个大规模核心生产业务，性能和稳定性都历经了多年考验。

2、基于 Lucene code-c 机制无缝扩展了向量的索引，阿里云 Elasticsearch 分布式容灾能力都与原生 Elasticsearch 完美兼容，支持Failover，横向弹性扩展、快照恢复等容灾能力。

3、基于在线引擎的方案，与原生 Elasticsearch 的NRT近实时机制相同，在线的向量增量数据写入阿里云 Elasticsearch 后，只需Refresh即可查到，最快能够做到秒级的可见性延迟。

4、封装了与原生 Elasticsearch DSL 协议一致的向量 QueryBuilder，阿里云 Elasticsearch 用户无需任何额外的学习成本即可快速上手向量检索引擎。

实现原理

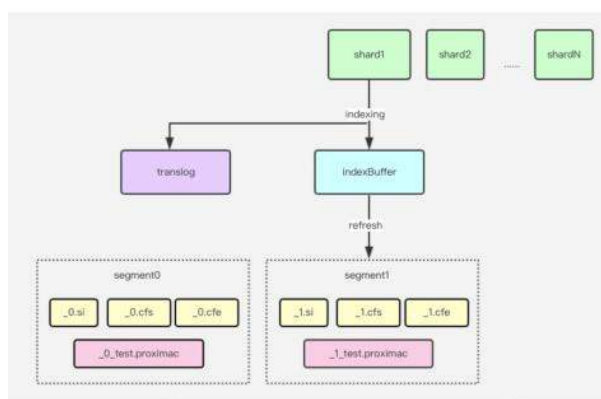


图 3：技术原理

- Lucene索引数据结构的抽象接口
- 可以自定义倒排/正排等索引的实现机制
- 当refresh刷出segment时，调用自定义Codec，构建segment级别的向量索引
- 查询时，通过实现自定义Weight和Scorer，返回当前segment的近似ann文档id和分数

方案的核心是运用 Lucene code—c机制。codec 是索引数据结构的抽象接口，可以自定义倒排/正排等索引；Lucene具体的工作流程是：当refresh刷出segment时，调用自定义 Proxima 的 Code-C，按照segment的力度，构建向量索引，即达摩院的向量索引。在查询时，按照 segment 维度，返回向量检索库 Proxima 找到的文档和分数即可，之后上传阿里云 Elasticsearch，并做全局打分的排序、召回的工作。

	hnsW
top10召回率	98.6%
top50召回率	97.9%
top100召回率	97.4%
延迟(p99)	0.093s
延迟(p90)	0.018s

- 阿里云ES 6.7.0版本
- 机器配置：数据节点16c64g*2 + 100G SSD云盘
- 数据集：sift128维float向量
(<http://corpus-texmex.irisa.fr/>)
- 数据量：2千万

图 4：测试数据

这里是一些测试的数据，我们可以看到，基于 hnsW 算法的向量检索插件，召回率和延迟都比较理想，实现98%的召回率和几十毫秒的延迟。

场景应用

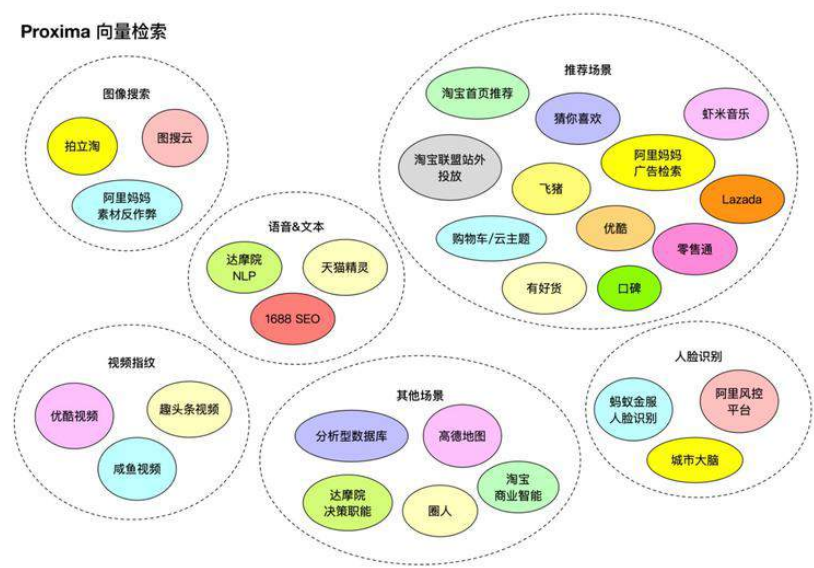


图 5：达摩院向量检索引擎在阿里体系的业务

图五列举了部分达摩院向量检索库在阿里体系的业务应用。在图像搜索应用领域，可以运用在拍立淘、图搜云、阿里妈妈反作弊等。推荐场景应用到了首页的猜你喜欢、推荐、广告检索，虾米音乐、飞猪等核心业务，同时还包括语言、视频之类，同时在天猫精灵、优酷核心场景均有落地。



图 6：向量检索应用领域

典型场景一“人脸识别”

向量检索的典型应用之一就是人脸识别。这个场景是比较简单且好理解的，比如我们的刷脸支付，由于原始数据量不会特别大，但却对召回的准确率和延迟都有非常高的要求。因为我们不会希望，别人刷了脸从我的账户里扣钱。除了刷脸支付，在安全监控场景也有运用，比如我们专有云客户，他们会用阿里云 Elasticsearch 向量检索引擎，来做路口交通的监控系统，通过一些具体的时段、路段标签字段作为一个协同，并根据人脸信息、车辆信息，抽取特征做向量字段，这些字段联合起来做查询，监测异常出现的轨迹。

典型场景二 “商品推荐”

在商品推荐场景中，比如经常看到的淘宝“猜你喜欢”。大家都知道要做到个性化的商品推荐，用户的行为历史、画像、Query 的特征，都是非常重要的一项参数，这些特征我们都可以做 Rebuilding，作为表征用户特征的向量。然后结合用户向量表征数据与商品侧召回的向量，利用一个近似距离的计算，我们就知道这个物品与用户特征之间，他们的相似度是怎么样，那我就可以通过这个相似度的排序，作为商品的推荐的重要参考标准。

对于一些比较长尾的 Query，如果按照传统的倒排和字典的方式，可能无法特别准确的召回文档，甚至可能出现没有结果的情况。这类情况下，向量检索就可以派上用场。比如当我们搜索“日式煎茶”这个物品时，通过精确查询召回的商品会特别的少，甚至可能召回一个完全不同的类目，比如召回“日式煎饼”。

当我们通过提前抽取商品特征以及 Query 特征，来做向量的近似召回就不同了。由于我们抽取了这些特征，所以我在搜索“日式煎茶”的时候，如果精确查询没有结果，我们也可以召回像“日式抹茶”、“日式清茶”这种实际特征相似的商品，作为补充召回的功能。不仅可以提升商品的曝光量，同时提升用户的使用体验。

典型场景三 “智能助理”

在“智能助理”应用场景中，如天猫精灵这种智能音箱，亦或是淘宝的客服系统 — 阿里小蜜这种智能文本客服，这类对话的场景对反馈延迟的要求非常高，并且数据量较少。如果做一些 Query 类目、情感等标签，以及多入口知识库召回，那可能匹配的知识项就非常多了。那么到精排阶段，我们向量检索就可以派上用场，通过精排阶段在向量检索库打分的时候，可以按照近期热点问题、近期促销活动等方式，通过向量近似计算，做到快速打分排序，返回给用户最精准的结果。

所以向量检索应用场景还是非常多的，只要业务上有近似需求，那我们都可以用向量检索的方案来做。

实操一快速运用向量检索

一、安装aliyun-knn插件

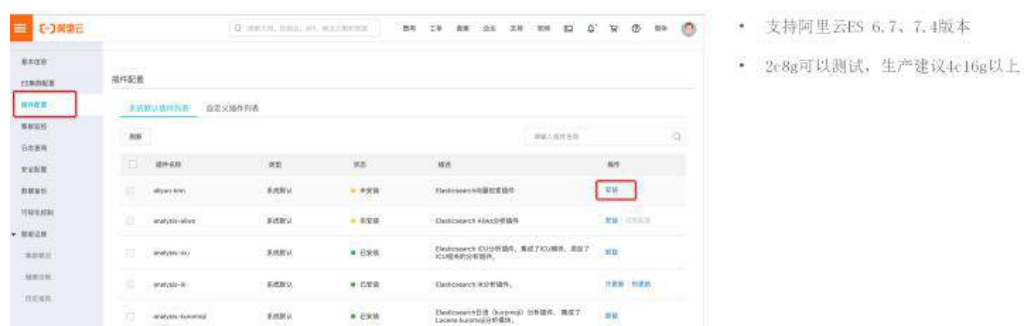


图 7：阿里云 Elasticsearch 控制台

我们可以登录阿里云Elasticsearch控制台页面，向量检索是以插件的形式来提供服务的，从【插件配置】Tab进入后，找到Aliyun-Knn（向量检索）安装即可。他是作为系统的默认插件存在，需要手动安装后使用，目前支持阿里云Elasticsearch 6.7、7.4版本，测试仅需2核8G就可以测试，实际线上生产，我们建议4核16G以上节点来安装。

二、创建索引

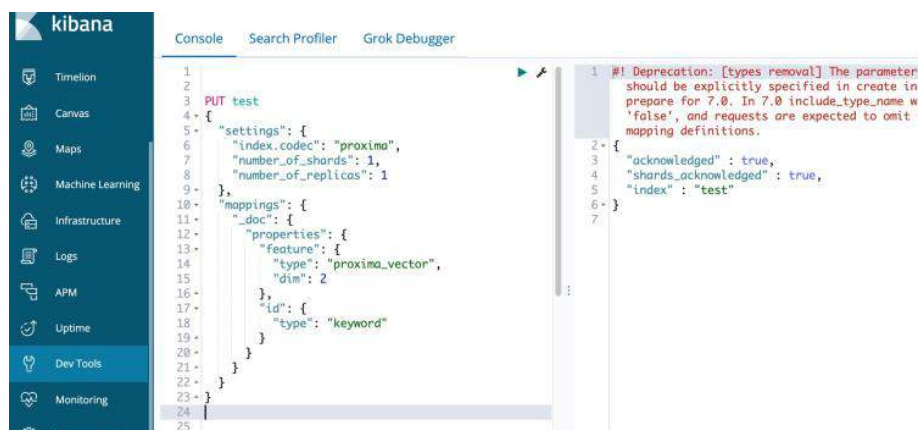


图 8：创建索引

与原生索引一致，图八中是一个 test 索引，除了传统的副本数、shard数以外，还有一个 proxima code-c 的 setting，这个是必须要加的。

在 mapping 的地方，由于 proxima 向量索引是字段，比如图中向量字段叫“feature”，那么我们需要加上feature的properties，“type”就是“proxima_vector”，这个就是指定 proxima 的向量字段，图八中指定了维度为“2”的向量字段，另外增加了一个字段ID，是keyword，演示的时候会用到。

三、添加文档

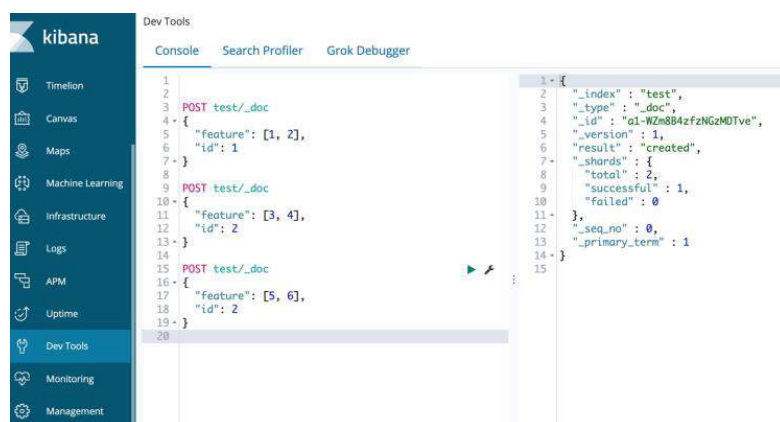


图 9：添加文档

添加文档与原生一样，图九中共写了3个文档，第一个文档ID为“1”，向量是【1,2】；后两个文档都是ID为“2”，而feature为【3,4】、【5,6】，也就是ID为“1”的字段，ID为“2”的字段有2个。

四、近似查询

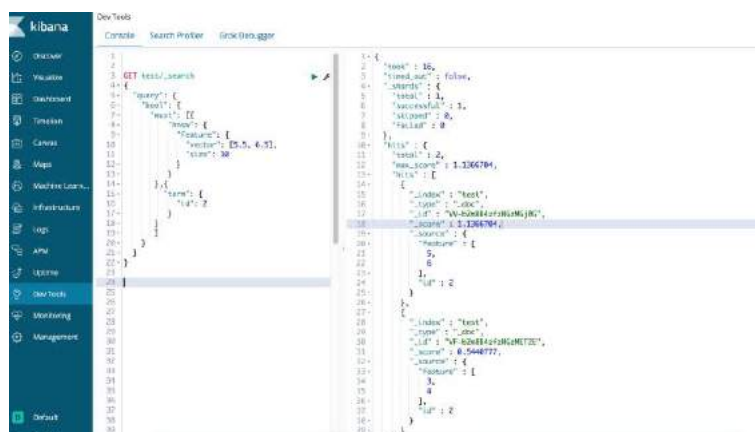


图 10：近似查询

做近似查询时，我们可以看到“hnsf”这个 query，就是我们向量查询的写译。他可以跟其他类型的查询做交并。我们可以看到，向量查询跟term查询是可以放在同一个 must，或者同一个 feature，在这种原生布尔查询里同时协同工作。

如果我们想查特征字段是【5.5, 6.5】，ID为“2”的文档时，那根据图八中第三个文档【5,6】，那么这个文档是最相近的查询结果，那么这个文档的打分是最高的，排在第一位。且由于指定ID为“2”，所以我的 feature 生效仅召回了两个文档，另一个就是【3,4】，而ID为“1”的文档没有被召回。

阿里云 Elasticsearch 索引数据生命周期管理

简介：索引生命周期管理（ILM）是指：ES数据索引从设置，创建，打开，关闭，删除的全生命周期过程的管理；为了降低索引存储成本，提升集群性能和执行效率，我们可以通过对存储在阿里云 Elasticsearch 的数据做生命周期管理。

本文作者：阿里云 Elasticsearch团队—Lettie

阿里云 Elasticsearch（>=6.6）提供 ILM 功能，同时将索引生命周期分为 Hot，Warm、Cold、Delete 4个阶段。

阶段	描述
Hot	主要处理时序数据的实时写入,根据索引的文档数大小时长决定是否调用 Rollover API 来滚动索引。
Warm	主要用来提供查询，索引不再写入。
Cold	查询较少，查询速度变慢，索引不再被更新。
Delete	删除数据

说明：Rollover 配置归档，目前仅支持三种策略，分别是：max_docs、max_size、max_age，其中任何一个条件满足都会触发索引归档操作。

场景介绍

业务场景中存在大量 heartbeat-* 时序索引，且每天新增单个索引大小都在 4MB 左右。数据越多，导致 Shard 数越多，导致增加集群负载过高。为了有效避免该类情况，需要规划不同的 Rollover 策略，滚动 heartbeat-* 开头的历史监控索引，实现 Warm 阶段对索引进行分片收缩及合并段，Cold 阶段将数据从 Hot 节点移动到 Cold (Warm) 节点，并定期删除索引数据。

操作说明

Heartbeat下配置ILM

为了使 Heartbeat 与 阿里云 Elasticsearch 的 ILM 无缝衔接，可在 Beat yml 配置中定义 ES ILM

详细配置参考[Set up index lifecycle management](#)

下载并解压 Heartbeat 安装包，通过以下命令编辑 Heartbeat.yml 配置分别定义 Heartbeat.monitors、setup.kibana、output.elasticsearch。

```
1.   heartbeat.monitors:- type: icmp
2.   schedule: '*/* * * * * *'
3.   hosts: ["47.111.169.233"]setup.template.settings:
4.   index.number_of_shards: 3
5.   index.codec: best_compression
6.   index.routing.allocation.require.box_type: "hot"setup.kibana:
7.   # Kibana Host
8.   # Scheme and port can be left out and will be set to the default
   (http and 5601)
9.   # In case you specify and additional path, the scheme is required:
   http://localhost:5601/path
10.  # IPv6 addresses should always be defined as:
   https://[2001:db8::1]:5601
11.  host: "https://es-cn-
4591jumei000u1zp5.kibana.elasticsearch.aliyuncs.com:5601"output.elastic
search:
12.  # Array of hosts to connect to.
```



```
13.     hosts: ["es-cn-4591jumei000u1zp5.elasticsearch.aliyuncs.com:9200"]
14.     ilm.enabled: true
15.     setup.template.overwrite: true
16.     ilm.rollover_alias: "heartbeat"
17.     ilm.pattern: "{now/d}-000001"
18.     # Enabled ilm (beta) to use index lifecycle management instead
    daily indices.
19.     #ilm.enabled: false
20.     # Optional protocol and basic auth credentials.
21.     #protocol: "https"
22.     username: "elastic"
23.     password: "Elastic@363"
```

ILM 及 Settings 配置说明:

- 1、index.number_of_shards: 指定主分片数，默认是1。
- 2、index.routing.allocation.require.box_type: 将索引数据写入hot节点。
- 3、ilm.enabled: 设置为true，用来启用index lifecycle management。
- 4、setup.template.overwrite: 覆盖原template数据，如果以前已将此版本的索引模板加载到es中，则必须通过该配置重新覆盖。
- 5、ilm.rollover_alias: 定义rollover 别名，默认是heartbeat-[beat.version]，通过该参数可指定别名前缀。
- 6、ilm.pattern: 定义滚动索引的pattern，这里支持date math，默认是[now/d]-000001，当触发条件后，新索引名称在最后一位数字上加1，比如第一次滚动产生的索引名称是heartbeat-2020.04.29-000001，当满足上述定义的rollover中的一项触发滚动后，创建新的索引，名称为heartbeat-2020.04.29-000002。

注意：如果在加载索引模板后修改rollover_alias 或 pattern，必须设置 setup.template.overwrite为true，重写template。

自定义ILM策略

heartbeat支持通过 `./heartbeat setup --ilm-policy` 命令，可加载默认的策略写进Elasticsearch，默认策略可通过 `./heartbeat export ilm-policy` 命令打印在stdout，可以对该命令进行修改，实现手动创建策略。

以下主要介绍如何手动创建策略。

索引生命周期策略支持通过 API 或 Kibana 配置，下面示例是通过 ilm policy API 创建 heartbeat-policy。

提示：通过 kibana --> management-->index lifecycle policies 配置索引生命周期策略。

```
1. PUT /_ilm/policy/heartbeat-policy
2. {
3.   "policy": {
4.     "phases": {
5.       "hot": {
6.         "actions": {
7.           "rollover": {
8.             "max_size": "5mb",
9.             "max_age": "1d",
10.            "max_docs": 100
11.          }
12.        }
13.      },
14.      "warm": {
15.        "min_age": "60s",
16.        "actions": {
17.          "forcemerge": {
18.            "max_num_segments": 1
19.          },
20.          "shrink": {
21.            "number_of_shards": 1
22.          }
23.        }
24.      },
```

```
25.     "cold": {
26.       "min_age": "3m",
27.       "actions": {
28.         "allocate": {
29.           "include": {
30.             "box_type": "warm"
31.           }
32.         }
33.       }
34.     },
35.     "delete": {
36.       "min_age": "1h",
37.       "actions": {
38.         "delete": {}
39.       }
40.     }
41.   }
42. }
43. }
```

Hot:

该策略将在写入达到 5MB 、使用超过1天、doc数超过100时，只要满足任一条件，都会触发 Rollover，系统将创建一个新索引，该索引将重新启动策略，而当前的索引将在滚动更新后等待 60s 后进入 Warm 阶段。

Warm:

索引进入 Warm 阶段后，ILM 会将索引收缩到 1 个分片，将索引强制合并为 1 个段，完成该操作后，索引将等待 3M（从滚动更新时算起）后进入 Cold 阶段。

Cold:

索引进入 Cold 阶段后，ILM将索引从 Hot 节点移动到冷数据（warm）节点，完成操作后，将等待1小时进入删除阶段。

Delete:

索引进入delete阶段，将在1小时后删除。

各个阶段支持不同种类的 Action，详细说明 [请参考Action](#)。

kibana 管理滚动索引

完成以上准备工作后，使用下面命令启动 Heartbeat 服务。

```
# sudo ./heartbeat -e
```

ILM关联模板

进入kibana Index lifecycle policies，为自定义的策略关联 Heartbeat 索引模板。

Add policy "heartbeat-policy" to index template

This will apply the lifecycle policy to all indices which match the index template. [Learn about index templates](#)

Template already has policy

This index template already has the policy heartbeat-policy attached to it. Adding this policy will overwrite that configuration.

Index template

heartbeat

Alias for rollover index

heartbeat

[Cancel](#)

Add policy

索引关联ILM

进入 index management

Index management

Update your Elasticsearch indices individually or in bulk.

Search

Lifecycle status Lifecycle phase

Reload indices

<input type="checkbox"/>	Name	Health	Status	Primaries	Replicas	Docs count	Storage size
<input type="checkbox"/>	test1	green	open	5	1	2	13.6kb
<input type="checkbox"/>	metricbeat-6.7.0-2020.04.28	green	open	1	1	155991	76.6mb
<input type="checkbox"/>	product_info	green	open	5	1	0	2.5kb
<input type="checkbox"/>	es_test_rds1	green	open	5	1	0	2.5kb
<input type="checkbox"/>	filebeat-6.7.0-2020.04.27	green	open	3	1	6212	2.9mb
<input type="checkbox"/>	filebeat-6.7.0-2020.04.26	green	open	3	1	180	664.7kb
<input type="checkbox"/>	myindex	green	open	1	0	3	10kb
<input type="checkbox"/>	heartbeat-2020.04.30-000001	green	open	1	1	21	1.78kb
<input type="checkbox"/>	metricbeat-6.7.0-2020.04.30	green	open	1	1	97845	57.7mb
<input type="checkbox"/>	logs-2020.04.30-1	green	open	5	1	2	15.6kb

Rows per page: 10

< 1 2 3 >

由于该索引默认关联的是 Beat 自带的策略，如默认策略没有生成，需要指定自定义的 policy，点击Manage，需要先remove lifecycle policy。

Elasticsearch

Index Management

Index Lifecycle Policies

Rollup Jobs

Cross Cluster Replication

Remote Clusters

Watcher

License Management

7.0 Upgrade Assistant

Kibana

Index Patterns

Saved Objects

Spaces

Reporting

Advanced Settings

Logstash

Pipelines

Beats

Central Management

Security

Users

Roles

Index management

Update your Elasticsearch indices individually or in bulk.

Search

Health

<input type="checkbox"/>	Name	Health
<input type="checkbox"/>	test1	green
<input type="checkbox"/>	metricbeat-6.7.0-2020.04.28	green
<input type="checkbox"/>	product_info	green
<input type="checkbox"/>	es_test_rds1	green
<input type="checkbox"/>	filebeat-6.7.0-2020.04.27	green
<input type="checkbox"/>	filebeat-6.7.0-2020.04.26	green
<input type="checkbox"/>	myindex	green
<input type="checkbox"/>	heartbeat-2020.04.30-000001	green
<input type="checkbox"/>	metricbeat-6.7.0-2020.04.30	green
<input type="checkbox"/>	logs-2020.04.30-1	green

Rows per page: 10

heartbeat-2020.04.30-000001

Summary Settings Mapping Stats Edit settings

General

Health green Status open

Primaries 1 Replicas 1

Docs Count 24 Docs Deleted

Storage Size 94kb Primary Storage Size

Aliases heartbeat

Index lifecycle management

index lifecycle error

illegal_argument_exception: policy [beats-default-policy] does not exist

Lifecycle policy beats-default-policy Current phase

Current action Current action time 2020-04-30 14:59:43

Failed step

Manage

再为该索引添加新策略

Add lifecycle policy to "heartbeat-2020.04.30-000001" ✕

Lifecycle policy

heartbeat-policy

Index rollover alias

heartbeat

Cancel

Add policy

如下关联成功

heartbeat-2020.04.30-000001

[Summary](#) [Settings](#) [Mapping](#) [Stats](#) [Edit settings](#)

General

Health	● green	Status	open
Primaries	1	Replicas	1
Docs Count	104	Docs Deleted	
Storage Size	136.1kb	Primary Storage Size	
Aliases	heartbeat		

Index lifecycle management

Lifecycle policy	heartbeat-policy	Current phase	hot
Current action	complete	Current action time	2020-04-30 15:06:17
Failed step	-	Show phase definition	

过滤各阶段索引

过滤Hot阶段滚动索引。

Index management

Update your Elasticsearch indices individually or in bulk.

ilm.phase:(hot)

Include rollover indices Include system indices

Lifecycle status Lifecycle phase Reload indices

Name	Health	Status	Primaries	Replicas	Docs count	Storage size
heartbeat-2020.04.30-000002	green	open	1	1	108	475.5kb
heartbeat-2020.04.30-000001	green	open	1	1	108	475.5kb

Rows per page: 10

过滤出处于 Warm 阶段的索引

Index management

Update your Elasticsearch indices individually or in bulk.

ilm.phase:(warm)

Include rollover indices Include system indices

Lifecycle status Lifecycle phase Reload indices

Name	Health	Status	Primaries	Replicas	Docs count	Storage size
heartbeat-2020.05.06-000002	green	open	3	1	108	475.5kb

Rows per page: 10

过滤出 Cold 阶段索引

Index management

Update your Elasticsearch indices individually or in bulk.

ilm.phase:(cold)

Include rollover indices Include system indices

Lifecycle status Lifecycle phase Reload indices

Name	Health	Status	Primaries	Replicas	Docs count	Storage size
shrink-heartbeat-2020.05.06-000001	green	open	1	1	108	50.1kb

Rows per page: 10

ILM策略周期

大家是否有观察到，策略中明明指定max_doc为100，为何 docs count 数达到100多后才滚动？

由于索引生命周期策略默认是10分钟检查一次符合策略的索引，索引可能会超出指定的阈值。 可通过修改 `indices.lifecycle.poll_interval` 参数来控制检查频率。

慎重修改，避免时间间隔太短给节点造成不必要的负载，本测试中将其改成了1m


```
1. PUT _cluster/settings{
2.   "transient": {
3.     "indices.lifecycle.poll_interval":"1m"
4.   }
5. }
```

总结

- 索引必须定义“模板”和“别名”两个条件才可以设置索引生命周期策略。
- 索引添加生命周期策略有两种方式：
 - 1、在索引生命周期中添加管理的模板：可以将策略应用到整个别名覆盖的索引下。
 - 2、对单个索引添加索引生命周期策略：只能覆盖当前索引，新滚动的索引不再受周期策略影响。
- 索引滚动中对策略做了修改，新策略将在下一个滚动索引生效。

PB级数据量背后阿里云 Elasticsearch 的内核优化实践

简介：本文将揭秘阿里云在面对 PB 级数据量挑战下所做的内核优化实践。

本文作者：广富 — 阿里巴巴 Elasticsearch 高级开发工程师

简述：

阿里云 Elasticsearch 云服务平台，是阿里云依托于云上丰富的云计算资源，向用户推出的托管的 Elastic Stack 服务，目前已在全球部署了17个地域，存储了 PB 级别以上的数据量。本文将揭秘阿里云在面对 PB 级数据量挑战下所做的内核优化实践。

一、阿里巴巴 Elasticsearch 云服务平台

阿里云 Elasticsearch 主要包括开源系统和自研系统两部分。开源系统依托于阿里云丰富的云计算资源，如 ECS, VPC, SLB 等，在此基础之上不仅向用户提供了 Elasticsearch, Kibana, Logstash, Beats 等 Elastic Stack 相关组件，还与 Elastic 官方合作向用户提供免费白金版 X-Pack 商业套件，用户可以在商业套件上体验如 Security, Altering, Monitoring, Graph, Reporting, MachineLearning 等高级特性。

在开源系统的基础之上，阿里云ES还提供了自研系统。自研系统部分主要包括 Elasticsearch 增强优化, Eyou 智能诊断系统和 ElasticFlow 离线数据处理平台等。其中 Elasticsearch 增强优化是阿里云在原生 Elasticsearch 基础之上，借助云计算和自研能力，为 Elasticsearch 集群增加了弹性伸缩，冷热数据分离，高可用，向量检索，中文分词等功能，可以帮助用户轻松应对业务压力，集群成本，数据可靠性，查询效果等问题。智能诊断系统 Eyou 可以智能地对用户的集群进行诊断，及时发现集群潜在隐患，并对用户集群给出合理的优化建议。

ElasticFlow 离线数据处理平台致力于解决如全量数据写入和复杂处理逻辑实时写入等情况下的业务瓶颈，为用户集群迁移，数据再处理，实时写入性能优化等提供了新的解决方案。

阿里云 Elasticsearch 致力于打造基于开源生态的，低成本的，场景化的云上 Elasticsearch 解决方案，源于开源，又不止于开源,目前已经覆盖了全部阿里云数据中心，并支持本地化专用云交付和混合云方案。



二、阿里云 Elasticsearch 内核优化实践

阿里云在内核优化方面的实践主要分为通用增强和场景化内核两部分。通用增强针对于通用的Elasticsearch场景进行优化，主要包括性能优化，高可用方案，稳定性保障和成本优化等。场景化内核当前已经提供了对日志场景，搜索场景，时序场景进行的优化，后续会不断持续地推出更多场景优化方案。



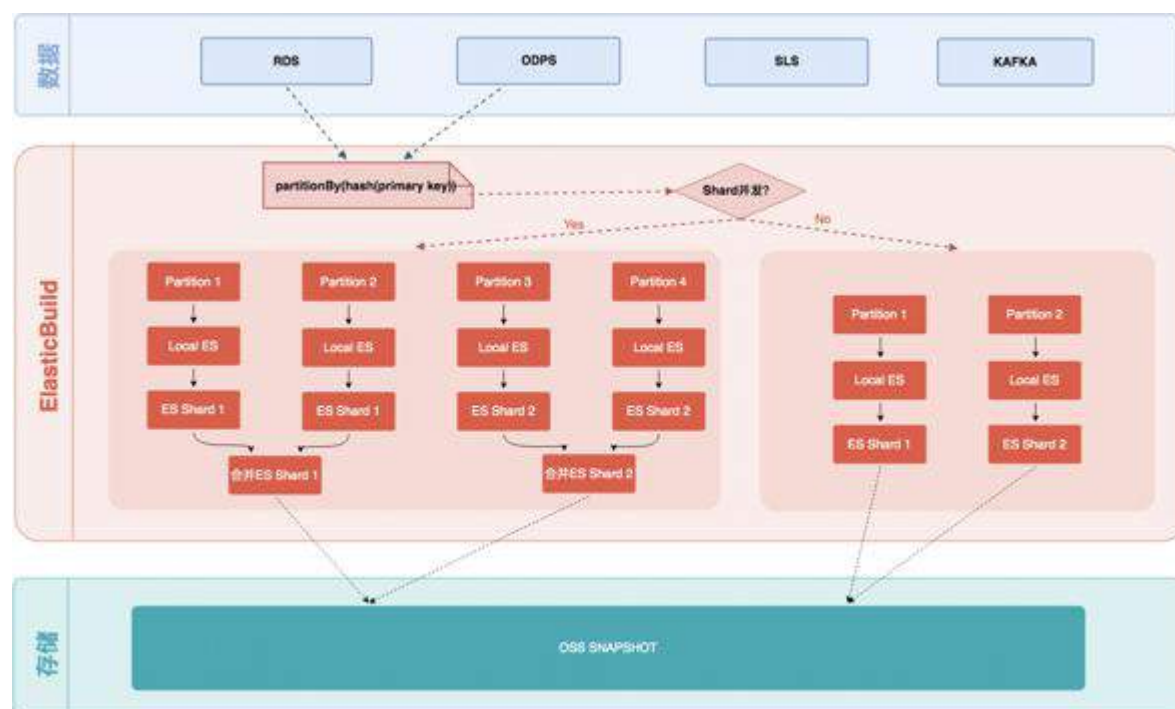
三、通用增强 - 性能优化

离线数据处理平台

为了解决全量数据写入和复杂数据处理逻辑实时写入等场景的业务痛点，阿里云ES推出了离线数据处理平台ElasticFlow。

原生Elasticsearch集群在进行数据导入，集群迁移等全量数据写入操作时，集群的读写性

能会受到较大影响，且由于全量写入数据较多处理时间较长，当面临海量数据时，耗时往往无法接受。Elastic Flow 针对于这种场景推出了全量 ElasticBuild。



当用户需要将数据从 RDS, ODPS, SLS, KAFKA, ES 等其他数据存储组件迁移到新的 Elasticsearch 集群时，可以先将数据流接入 ElasticBuild。ElasticBuild 底层使用 Blink，对导入的数据进行并发索引构建。接着将构建好的索引文件存储在 OSS 中。目标集群通过原生 Elasticsearch 的 Snapshot restore 操作即可快速地完成数据导入。

全量 Elastic Build 由于在集群外部进行，因此索引构建过程对集群读写性能无影响，且由于使用 blink 将索引构建过程并发起来，并通过内存索引合并减少磁盘IO, 从而大大降低了索引构建时间，加快索引构建速度，对于解决海量数据全量导入耗时过长问题效果显著。

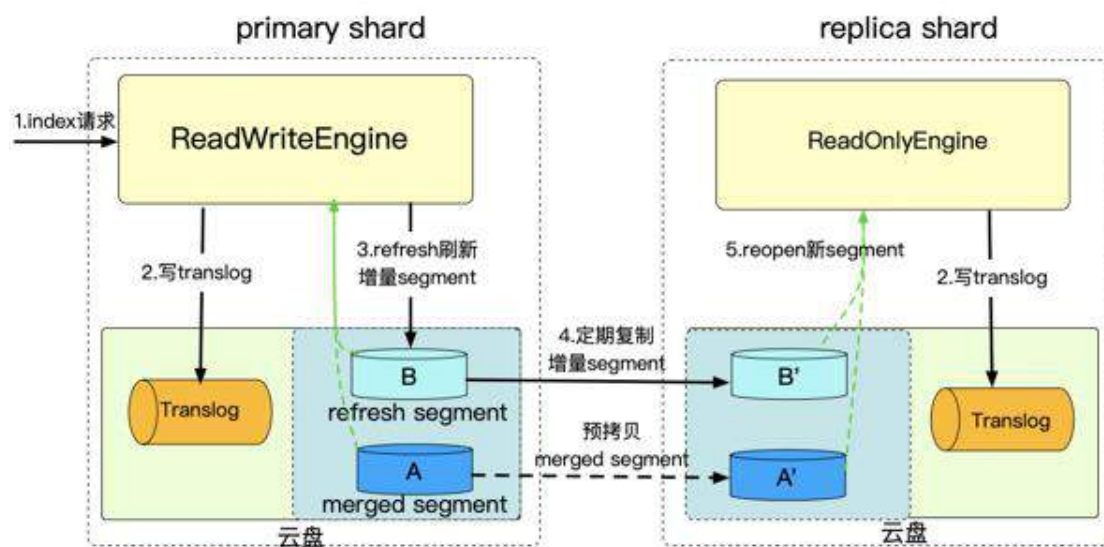
将索引构建操作移到集群外部进行，虽然解决全量数据写入的性能问题，但无法使用 Elasticsearch 原生的 translog 来解决 failover 问题。对此，ElasticFlow 使用了 Blink 的 checkpoint 来实现 failover，相对于 translog 降低了一倍的 IO 开销，并可以实现 At Least Once 数据准确和秒级 Failover。

针对于复杂处理逻辑实时写入场景，ElasticFlow 推出了增量 ElasticBuild，通过将处理逻辑移至集群外部进行，减少 client node 的计算和网络开销。

Elastic Flow 离线数据处理平台充分利用阿里云在大数据处理的优势，通过优化全量 Build 相对于在线 build 性能提升3倍，增量数据 build 相对于实时写入性能提升了30%。

通用物理复制

原生 Elasticsearch 在进行写操作时，主副分片都需要进行写 translog 和写索引操作。写 translog 是为了保证数据的高可用，主副分片都需要单独保证。但是索引构建和 segment merge 等操作不一定需要主副分片分别重复执行。基于这种思想，我们推出了物理复制功能。



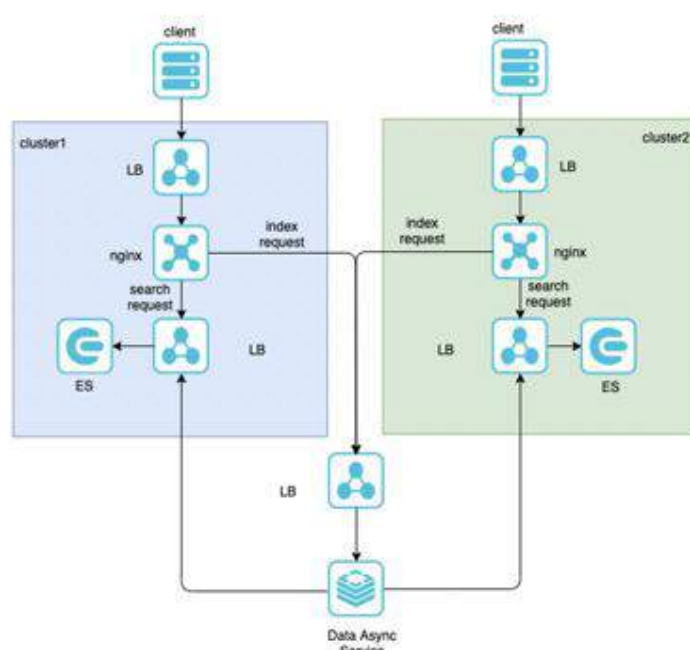
物理复制的基本原理是：Elasticsearch 主分片写入机制与原生 Elasticsearch 一样，既写索引文件也写 translog，而副本分片只写 translog 以保证数据的可靠性和一致性。主分片在每次 refresh 时，通过网络将增量的索引文件拷贝到副本分片，在主副分片分配可见性延迟略增加的情况下，大幅度提高了集群的写入性能。我们使用 esrally 自带的 nyc_taxis 数据集对 8 核 32GB 5 节点+2TB SSD 云盘集群进行测试，与原生 Elasticsearch 相比，阿里云 Elasticsearch 在开启了物理复制功能后，写入性能提升大于 45%。

四、通用增强 - 高可用方案

起新下老与读写分离

原生 Elasticsearch 在进行集群升级操作时有一些限制，如集群升级前要进行升级检查，本地升级集群中有不同版本节点容易出现兼容性问题，跨大版本升级要先升级到中间版本，不支持回滚等。为了解决平滑升级等问题，我们推出了起新下老和读写分离功能。

如下图，当需要升级集群 cluster1 时，我们先创建一个新版本的集群 cluster2，然后在 cluster1 和 cluster2 的 client 与集群之间引入 nginx 实现读写分流，client 的读请求仍然落到原集群，写入请求落入一个异步数据处理服务。该异步数据处理服务可以接收对 Elasticsearch 的写入请求，并将写入操作存储在消息队列等数据储存组件，最后消费数据双写到 cluster1 和 cluster2。对于 cluster1 存量数据可以使用 ElasticFlow 全量 build 等方式同步到 cluster2。引入该架构后，cluster1 和 cluster2 可以实现近实时数据同步，用户可以在 cluster2 上验证新版本功能，如果 cluster2 符合预期，则将 cluster1 下线，实现平滑升级，当 cluster2 不符合预期是，将 cluster2 下线，实现业务回滚。



起新下老和读写分离架构，在不影响原有集群业务的前提下，通过引入异步数据链路实现了新老集群的数据同步，对业务影响小，可以实现业务平滑升级和回滚，解决了 Elasticsearch 版本升级中面临的多种业务痛点。

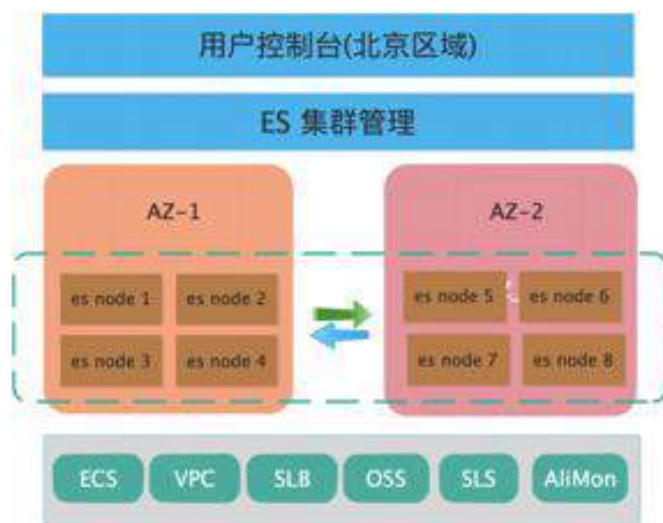
数据备份

为了保证数据的高可用，阿里云 Elasticsearch 提供了数据自动备份功能。用户可以在阿里云 Elasticsearch 官网控制台通过简单配置，开启数据自动备份功能。开启后，Elasticsearch 集群会定时将数据快照备份到 OSS 仓库中。一旦集群出现异常，用户可以从 OSS 仓库中恢复数据，快速恢复业务。

跨可用区部署

阿里云 Elasticsearch 支持用户将集群部署在多个可用区。当单个可用区由于自然灾害或其他原因导致不可用时，阿里云 Elasticsearch 会自动检测可用区异常，并快速将故障可

用区隔离。可用区故障期间，阿里云 Elasticsearch 会保障用户集群的可用性。若短期内故障可用区可以正常恢复，则集群会自动将故障节点加回集群，若故障可用区无法短期恢复，阿里云 Elasticsearch 会在其他可用区为集群创建数量及规格相同的节点恢复集群。采用跨可用区部署的集群可以实现自动的故障可用区检查，隔离及恢复，期间业务零影响，可以帮助用户在不感知无感知的情况下平稳度过可用区故障。



五、通用增强 - 稳定性保障

Master 调度优化

原生 Elasticsearch 当集群分片数过多时，集群索引创建和删除耗时会显著增加。我们实际中遇到的一个典型用户集群有 3 个专用主节点，10 个热节点，2 个冷节点，超过 5 万个 shard，索引创建和删除耗时超 1 分钟。通过对集群性能瓶颈和 ES 实现逻辑分析，我们最终发现原生 Elasticsearch 的 reroute 逻辑是性能瓶颈存在优化空间。通过引入新的数据结构，我们将 reroute 的调度算法复杂度从 $O(n^2)$ 降低到了 $O(n)$ ，万级分片集群索引创建和删除时间也降低到了 1s，当前该 PR 已经提交并作为 co-author 合并 (<https://github.com/elastic/elasticsearch/pull/48579>)。

阿里云 QOS

为了帮助用户 Elasticsearch 集群更好地应对业务流程高峰，阿里云 ES 提供了阿里云 QOS 功能。使用该功能用户可以对集群的节点和索引级别的流量进行限流配置。当业务流量突增时，可以将流量控制在设定的范围内，保证集群稳定性。用户可以对不同的索引和节点设置不同的限流配置，在流量高峰时优先保证核心业务。该功能支持动态开关调整及动态限流规则调整，可以帮助用户在业务流量高峰时及时灵活地调整限流规则。

六、通用增强 - 成本优化

数据压缩

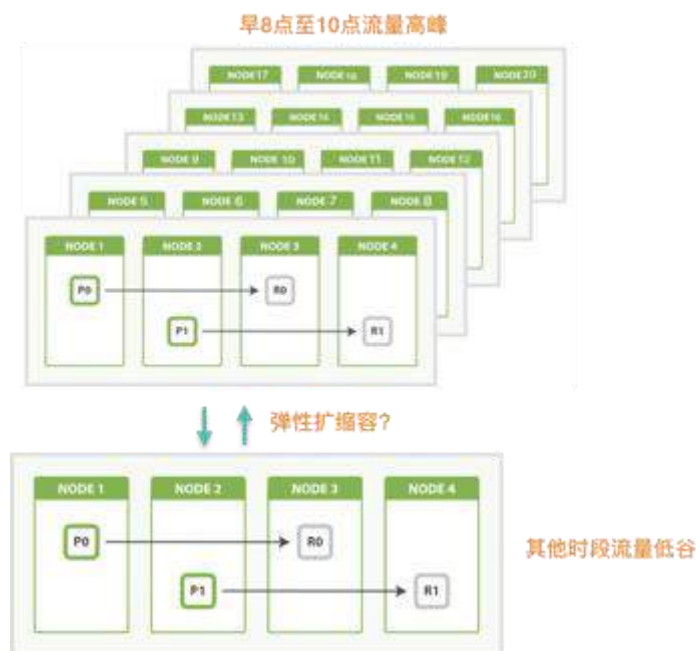
为了降低存储成本，原生 Elasticsearch 会对 store filed，doc_values 等进行数据压缩，以降低这些数据的存储空间。原生 Elasticsearch 提供的压缩算法是 LZ4 和 best-compression，其中 LZ4 有较好的压缩性能，best-compression 有较高的压缩率，用户可以根据业务的情况选择符合业务场景的压缩算法。为了进一步提高数据的压缩率，阿里云 Elasticsearch 引入了 brotli 和 zstd 两种压缩算法。这两种压缩算法在保证读写性能的基础之上进一步提高了数据的压缩率，相对于 LZ4 提供了45%，相对于 best-compression 提升了8%。

下表是使用 esrally 自带 nyc_taxi(74GB)，对 16 核 64GB 3 节点 2TBSSD 磁盘集群，使用不同压缩算法的测试结果。

	索引大小(GB)	写入TPS(doc/s)
ES默认压缩算法(lz4)	35.5	202682
best-compression(deflate)	26.4	181686
brotli	24.4	182593
zstd	24.6	181393

弹性伸缩

通过对大量 Elasticsearch 集群的分析，我们发现相当一部分集群存在明显流量的高/低峰情况。业务高峰时，集群承受较大的压力，集群的性能和稳定性面临挑战。业务低峰时，节点资源闲置，出现资源浪费。针对有明显高/低峰期规律的集群，阿里云ES近期新推出了弹性伸缩功能。该功能允许用户在创建集群时除了购买常规的集群节点，还可以额外购买一种新的节点类型——弹性节点，并可以指定弹性节点的扩容和缩容时间。通过这种方式创建出来的集群会根据用户设定的时间扩容弹性节点和缩容弹性节点。对于业务存在明显流量高峰和低峰的集群，可以使用该类型集群，在业务高峰时扩容出弹性节点，保证集群的性能和稳定性，在业务低峰时缩容弹性节点，降低集群成本，防止资源浪费。



七、场景化内核 - 场景化推荐模板

Elasticsearch 的优势之一是上手快，集群搭建简单，初学者即可通过简单的配置快速拉起一个 Elasticsearch 集群，这一优势得益于 Elasticsearch 合理的默认配置。这些默认配置不仅帮助用户简化集群搭建成本，集群在功能和性能上也都会有不错的表现。然而 Elasticsearch 的默认配置是针对所用场景通用的，在特定使用场景下并不一定是最优的。基于这种思想，阿里云 Elasticsearch 推出了场景化推荐模板功能。



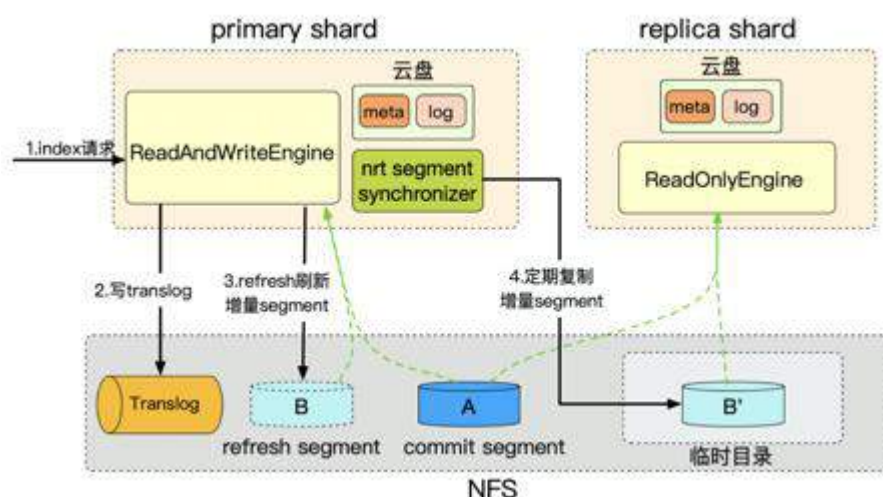
场景化推荐模板基于阿里云 Elasticsearch 丰富的 Elasticsearch 使用，运维和内核优化经验，根据用户的使用场景为用户推荐符合其使用场景的集群配置，索引模板和索引生命周期管理策略等。用户只需要在创建集群时指定其使用场景，其创建出的集群就会将上述推荐模板自动配置到集群中。用户还可以在后续使用中根据业务的实际情况对推荐模板进行调整。

八、场景化内核 - 日志场景

日志场景是典型的写多读少场景，数据量较大，对存储成本比较敏感，为此我们主要针对存储成本进行优化，包括计算存储分离和冷热分离等。

计算存储分离

原生 Elasticsearch 通过多副本的方式来保证数据高可用。然而在云计算环境下，集群节点底层存储通常使用云盘，云盘为了保证数据高可用已经对数据保存了三副本。这样对于开启一副本的索引，云计算环境下相当于保存了六份数据，这对于存储空间是一种巨大的浪费。基于这种思想，阿里云 Elasticsearch 推出了计算存储分离功能。



计算存储分离功能使用 NFS 共享存储作为节点底层存储，多个节点共享一份底层存储。主分片使用 ReadAndWriteEngine 既可进行写操作又可以进行读操作。副本分片使用 ReadOnlyEngine 仅提供读操作。集群写入文档时，只对主分片进行写操作，主分片定时将增量 segment 拷贝到副本分片的临时目录保证副本分片的实时性。

计算存储分离架构多节点底层共享一份数据文件，实现了一写多读，在将写入性能提升 100% 的情况下，成倍降低了存储成本。数据的高可用由底层 NFS 的三副本来保证。另外由于多个节点共享一份存储，扩缩新节点时不需要进行数据的拷贝操作，因此可以提供秒级弹性扩缩容能力。

冷热分离

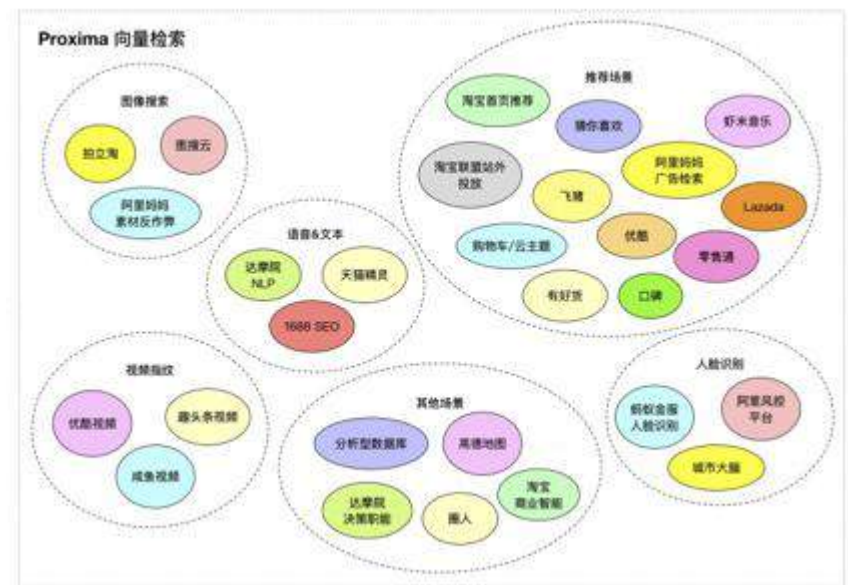
相当一部分 Elasticsearch 的集群存在明显的热点数据情况。用户的读写操作往往只集中在部分数据集上，其他数据的读写频率很低。对于日志场景，这种情况尤其明显，用户的读写操作主要集中在最近一段时间的索引，对于历史索引的读写操作很少。针对于这种情况，阿里云 Elasticsearch 推出了冷热分离功能。

使用冷热分离功能，用户可以在创建索引时除创建正常数据节点外，额外指定一定数量的冷节点。数据节点使用计算资源和磁盘性能较好的机器，如核数内存较大的SSD磁盘机器，冷节点使用计算资源和磁盘性能较差的机器，如核数内存小的SATA磁盘机器。接着用户在 Elasticsearch 集群中为索引配置索引生命周期管理策略，即可实现新创建的索引落在热节点上，当索引创建超过一定的时间，会被迁移到冷节点上。这样一方面保证了热点数据在性能较好的节点上，数据读写性能得到保障，另一方面冷数据存储性能较差的节点上，降低了集群成本。

九、场景化内核 - 搜索场景

向量检索库

阿里云基于达摩院高性能向量检索库为用户提供了向量检索功能，该功能已成熟应用于手淘猜你喜欢、拍立淘、优酷视频、指纹识别等大规模生产应用场景。该功能使用 Elasticsearch Codec 机制将高性能向量检索库与 Elasticsearch 结合，完美兼容 Elasticsearch 分布式能力。基于 Hnsw 算法，无需训练，查询速度快，召回率高。



下表是使用阿里云6.7.0版本ES对向量检索功能的测试结果：

机器配置：数据节点16c64g*2 + 100G SSD云盘

数据集：sift128维float向量

数据量：2千万

Top10召回率	98.6%
Top50召回率	97.9%
Top100召回率	97.4%
延迟(p99)	0.093s
延迟(p90)	0.018s

阿里分词

阿里云基于阿里巴巴 alinlp 分词技术推出了阿里分词功能，该功能支持多种模型和分词算法包括 CRF、结合词典的 CRF、MMSEG 等，应用于多种业务场景包括淘宝搜索、优酷、口碑等，提供近1G 的海量词库，并且支持热更新分词能力。

下图是几种分词器的分词效果对比

分词器	分词结果	查全率	查准率	查询性能
标准分词器standard	南/京/市/长/江/大/桥	高	低	低
中日韩文分词器cjkcjk	南京/京市/市长/长江/江大/大桥	高	低	高
IK中文分词器ik_max_word	南京市/南京/市长/长江大桥/长江/大桥	高	低	高
IK中文分词器ik_smart	南京市/长江大桥	低	高	高
阿里中文分词器aliws	南京/市/长江/大桥	高	高	高

在对"南京市长江大桥"进行分词时，standard 分词器直接按照单个汉字进行分词，虽然有较高的查全率，但是查准率和查询性能很差，基本没有分词效果。中日韩文分词器 cjkcjk 和 ik_max_word 虽然可以将短语切分为中文分词，但是其分词结果中出现了如"市长"等错误分词，虽然有较好的查全率和查询性能，但是查准率较差。ik_smart 分词器将短语分词为"南京市"和"长江大桥"，没有将"南京"，"长江"，"大桥"等作为单独分词，在查全率上表现不佳。而阿里分词将短语分词为"南京"，"市"，"长江"，"大桥"，在查全率，查准率，查询精度等方面均有不错表现。

aliyun-sql

阿里云 Elasticsearch 最新还推出了 aliyun-sql 功能，该功能基于 Apache Calcite 开发了部署在服务端的 SQL 解析插件，使用此插件可以像使用普通数据库一样使用 SQL 语句查询 Elasticsearch 中的数据，从而极大地降低学习和使用ES的成本。

下表是 x-pack-sql, opendistro-for-elasticsearch, aliyun-sql 的功能对比图。可以看到, X-pack-sql 不支持 join 操作, 也不支持 Case Function 和扩展 UDF。opendistro-for-elasticsearch 不支持分页查询, 也不支持 Case Function 和扩展 UDF。而 aliyun-sql 对分页查询, Join, Case Function, UDF 等均有较好支持, 功能更为全面。

SQL插件	sql解析器	分页查询	Join	Nested	常用Function	Case-Function	扩展UDF	执行计划优化
x-pack-sql (6.x版本)	antlr	支持	不支持	支持 (正常语法为a.b)	支持的Function较丰富	不支持	不支持	执行计划优化规则相对较多
opendistro-for-elasticsearch	druid	不支持 (最大查询数量受ES的 max_result_window 参数限制)	支持	支持 (nested(message.info))	支持的Function较少	不支持	不支持	有少量的执行计划优化规则
aliyun-sql	javacc	支持	支持。具有截断功能。可动态配置单表查询数量。详情请参见 语法介绍 。	支持 (正常语法为a.b)	支持的Function较丰富。详情请参见 Function和表达式 。	支持	支持。详情请参见 自定义UDF函数 。	执行计划优化规则相对较多, 并且使用 Calcode 优化执行计划

阿里云 Elasticsearch 将阿里云丰富的云计算能力与 Elastic Stack 开源生态相结合。在为用户提供原生 Elastic Stack 服务的同时, 不断地在 Elasticsearch 内核上进行优化实践。我们源于开源, 又不止于开源, 将不断地在内核方面做出更多的优化, 回馈开源, 赋能用户。

一次有趣的Elasticsearch+矩阵变换聚合实践

简介：Elasticsearch 聚合功能非常丰富，性能也相当不错，特别适合实时聚合分析场景，但在二次聚合上也有明显短板。本项目是一个基于日期维度做预处理的技术方案，以下是结合 Elasticsearch 优缺点扬长避短的一次尝试性实战，非常有趣，希望可以带来一些参考，同时欢迎各种讨论。

作者介绍：李猛，Elastic Stack 深度用户，通过 Elastic 工程师认证，2012年接触 Elasticsearch，对 Elastic Stack 技术栈开发、架构、运维等方面有深入体验，实践过多种大中型项目；为企业提供 Elastic Stack 咨询培训以及调优实施；多年实战经验，爱捣腾各种技术产品，擅长大数据，机器学习，系统架构。

背景需求

公司所属行业是物流速运，面向企业服务（简称ToB模式），提供多种物流运输方案产品，客户分布遍布全国，客户数量在百万级以上，日均产生物流运输需求在几十万票（单）以上，对于客户订单的聚合统计分析查询需求强烈，且需要一定的实时性。

同时需要满足以下用户需求：

- 1、用户需要在地图上展示客户的聚合分布；
- 2、聚合分布维度按照全国、省、市、区县、乡镇划分。



地图展示样例：非内部效果图

筛选条件

用户端基于多个筛选条件过滤聚合，选择任意条件组合，如下：

- 行政区域：

按照国家4级行政区域：省、市、区、镇等数量在5000+以上

- 企业组织架构：

企业内部多层级组织架构：大区、小区等数量超过3000+以上

- 客户企业类型：

客户企业类型划分：2B、2C等数量在10+以上

- 客户行业类型：

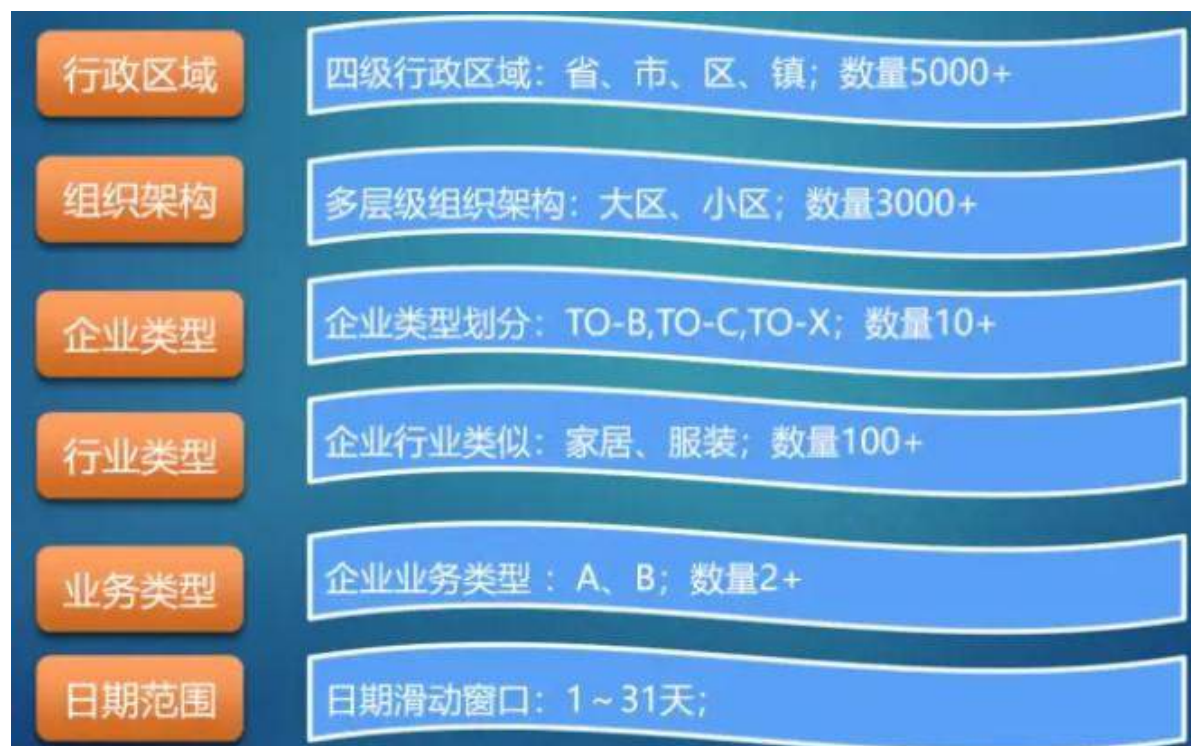
客户企业行业所属类型划分，如家具、服装、电子、3C等数量在100+以上

- 企业业务类型：

企业物流业务类型，如寄件、派件、未寄件派件等

- 日期范围：

日期范围筛选限制在1个月，即日期的滑动窗口在1~31天（这个限定范围是与业务部门多次讨论得来，否则后面实现的代价会更大，原有是多个月的窗口期）



筛选条件说明

业务模型

业务数据模型说明：

- 单个客户即使单天下单多次，单天聚合统计也只能算1个客户；
- 单个客户连续多天都有下单，多天聚合统计也只能算1个客户；
- 业务类型有寄件/派件，按照其中一种处理，逻辑比较计算。

原始业务数据模型						
下单时间	客户编号	行政区域（4级）	企业类型	行业类型	业务类型	组织架构（多级）
2019-04-10 09:09:09	A001	广东省/深圳市/宝安区/沙井镇	B2B(编号 B001)	电子(编号C001)	寄件（0/1/2）	华东/浙江/杭州/上城/xxx市场部
2019-04-10 10:10:10	A001	广东省/深圳市/宝安区/沙井镇	B2B(编号 B001)	电子(编号C001)	收件（0/1/2）	华东/浙江/杭州/上城/xxx市场部
2019-04-10 11:11:11	A001	广东省/深圳市/宝安区/沙井镇	B2B(编号 B001)	电子(编号C001)	收件（0/1/2）	华东/浙江/杭州/上城/xxx市场部

业务需求数据模型						
下单日期	客户编号	行政区域（4级）	企业类型	行业类型	业务类型	组织架构（多级）
2019-04-10	A001	广东省/深圳市/宝安区/沙井镇	B2B(编号 B001)	电子(编号C001)	寄件（0/1/2）	华东/浙江/杭州/上城/xxx市场部

样例数据模型说明

聚合数据模型

聚合数据模型说明：基于前面的业务模型数据聚合，按照区域+其它条件聚合，获取聚合后分组客户数量。

业务需求数据模型				
行政区域（四级）	企业类型	行业类型	业务类型	客户数量
广东省/深圳市/宝安区/XX镇	B2B(编号 B001)	电子(编号C001)	寄件（0/1/2）	10000
广东省/深圳市/福田区/YY镇	B2c(编号 B002)	家居(编号C001)	收件（0/1/2）	20000

聚合后的业务数据模型

技术抽象

业务需求是一个很典型的聚合统计，多数大数据产品或者传统关系数据库都支持，相反Elasticsearch 聚合支持的不怎么好，不能满足需求。

业务需求的技术本质实际上是一个去重然后分组聚合的过程：

1. **去重合并**：按照客户维度去重，合并符合过滤条件的客户数据，相同多条客户数据合并为单条数据；
2. **聚合分组**：按照聚合维度分组，并计算出分组后的客户数量。



技术抽象过程

技术尝试

在实现业务需求过程中尝试过多种技术产品，遇到不少问题：

1. **Mysql**：当数据达到一定数量级，运行超时，甚至直接运行不起来；
2. **Prestodb**：定位是秒级分析型产品，单任务启动就需要消耗好几秒的时间，且受资源限制，并发度与响应度不能满足要求，优点可与 Hive 很好结合。
3. **MPP**：Greenplum/Vertica/Infobright，与 Prestodb 其实本质差不多，都不能满足性能要求。
4. **穷举法**：探讨过将所有的组合条件全部计算存储起来，业务系统只要去定位去查询，

比如 Kylin 产品，查询复杂度确实低了，但计算量与存储量实在是太大，根本不现实；

5. Elasticsearch：虽然提供了聚合能力，但不支持在一次聚合过程中完成去重与分组统计，也就是不支持复杂的二次聚合，这是 ES 局限，也是 ES 定位。

	计算维度	计算维度数量
	行政区域	31（省/非港澳台）
	企业类型	10
	行业类型	100
	业务类型	2
	日期窗口	31阶乘
合计		$31 \times 10 \times 100 \times 2 \times 31 \text{阶乘} = \text{XXXX}$



举法计算量=愚公

矩阵转换

技术尝试过多次不同的技术产品之后得出结论，单一的数据产品已有能力是无法满足要求的，正可谓鱼与熊掌不可兼得。所以必须改变思维，设计了一种矩阵变换的算法机制，结合 Hive+ES 实现，下面介绍这种技术实现方式。

可转换性分析：分析原有业务需求，发现只有日期这个条件组合特别多，动态变化范围很

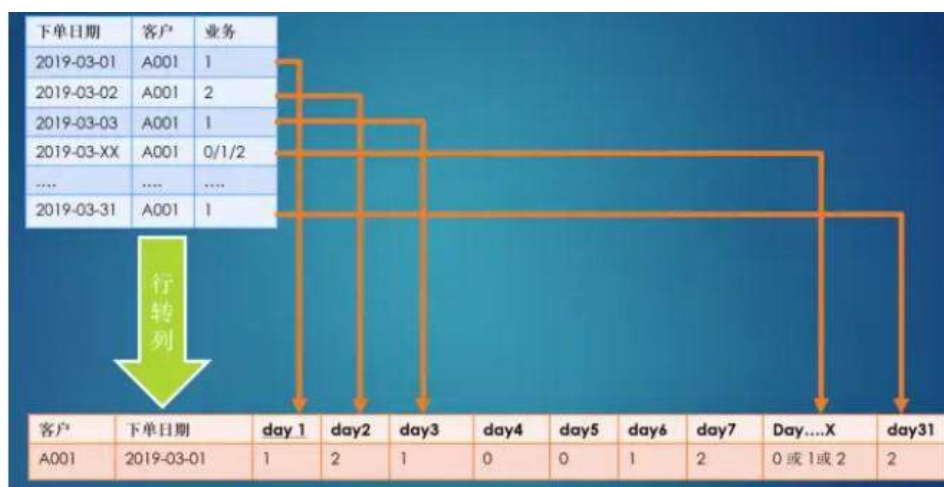
大，如果按照单月最长31天计算组合数就有31的阶乘；其余的条件变化小，也没有动态的组合条件，所以重点解决日期组合这个条件。



下单日期可变数大

数据行转列：原有业务数据是按照行存储，聚合日期最小粒度是天，单个客户下单信息除了下单日期、业务类型，其余的是相同的；将单个客户单月 31 天的下单数据 31 条转换成 1 条数据 31 列存储，31 列分表代表从下单日期往后叠加的日期，列存储的值代表当天是否有下单以及业务类型。

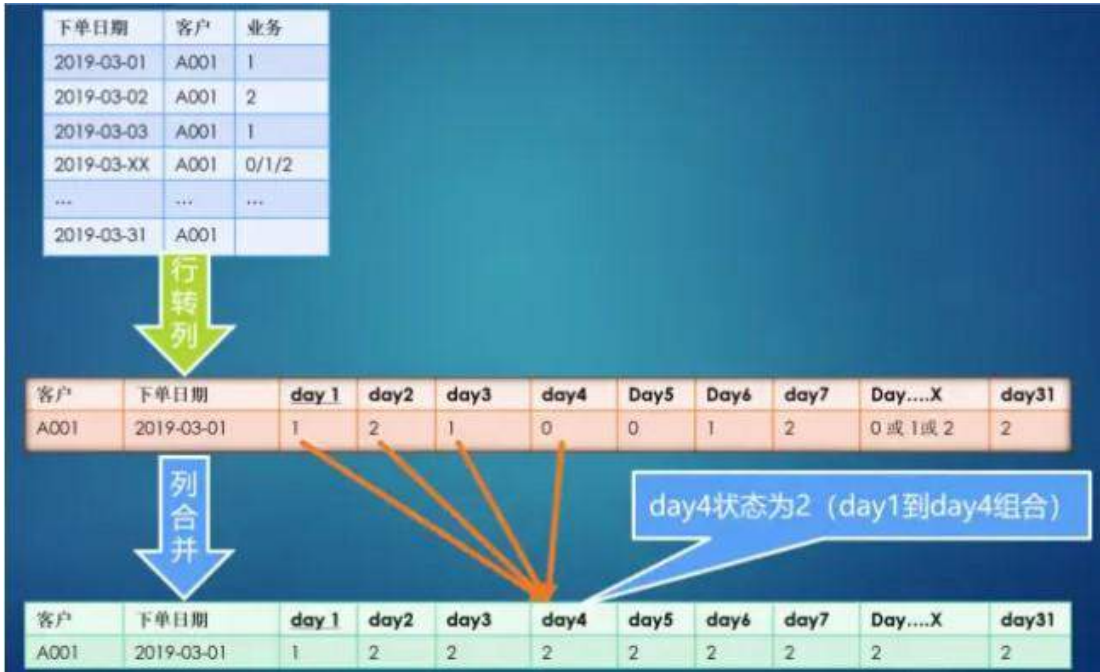
1. 本次行转列基于 Hive 实现，数仓 ODS 数据都存储在 Hive 里，方便做下一步数据清洗转换计算；
2. 首先在 Hive 上按照【客户+日期】维度将客户下单数据去重，并按照业务类型简单的逻辑计算，合并单日多次下单的业务类型；
3. 客户数据按照日期排序，从历史日期到当下昨天日期，计算任务默认 T+1；
4. 其次在 Hive 中将去重后的客户数据，按照行转列模型将 1~31 天行数据转换到 31 列的数据，并填充原始行的业务类型值。



客户端行转列示意图

列合并逻辑计算：业务需求是按照日期范围聚合，在一个日期范围内，客户订单业务类型要做一些逻辑计算（业务类型：0/1/2），按照最大，所以需要计算单个客户单条数据之后 31 天的业务类型。

- 1. 本次列合并逻辑计算基于 Hive 实现；
- 2. 合并完整的数据之后按照月的维度分开存储，当计算任务下次 T+1 运行时，只要更新最近 31 天的数据，最多跨度 2 个月。
- 3. 数据同步到 Elasticsearch 中，一个月一个索引，也只要更新最近的 2 个索引。
Elasticsearch 更新索引也很方便，采用别名切换方式，可在毫秒间完成，ES 这个优点有效的避免了业务系统查询停顿空白问题。

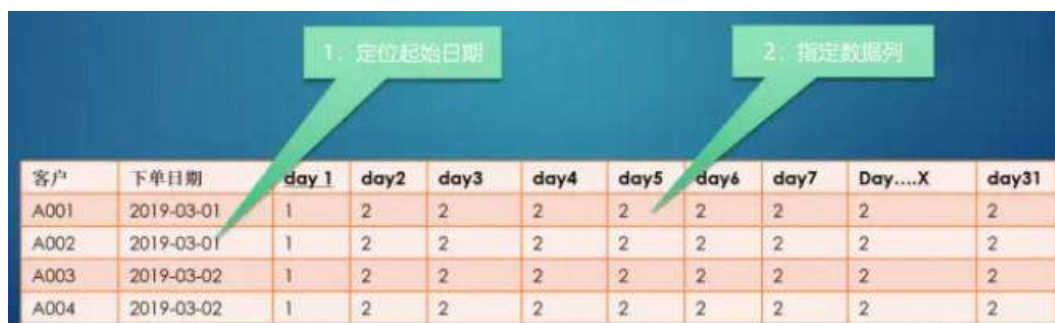


客户日期列逻辑合并

业务查询

选择 Elasticsearch 做为查询引擎是非常正确的，得益于 Elasticsearch 高效的查询机制以及高效的聚合能力。

- 1. 依据起始日期定位到该日期的月度索引，并锁定对于下单日期所有数据，Elasticsearch 支持动态索引搜索，支持高效的过滤 filter 扫描；
- 2. 依据结束日期与起始日期差值，定位到指定的数据列；
- 3. 最后只要一次聚合即可返回数据，Elasticsearch 支持高效的聚合特性 agg。



客户	下单日期	day1	day2	day3	day4	day5	day6	day7	Day....X	day31
A001	2019-03-01	1	2	2	2	2	2	2	2	2
A002	2019-03-01	1	2	2	2	2	2	2	2	2
A003	2019-03-02	1	2	2	2	2	2	2	2	2
A004	2019-03-02	1	2	2	2	2	2	2	2	2

说明案例：查询2019-03-01~2019-03-05 客户聚合数据

结语

本次需求的技术实现比较曲折，在探讨大数据分析方面做了一次很重要的探索实践，没有一种通用的数据产品即可满足性能与功能，所以在面对实际业务问题要去探讨多种技术的混合实践。本次项目中的 Hive+ES 结合就是一次很有趣的混合。

学会培养一些算法思维，用微观算法的思维分析问题解决问题。本次项目中采用矩阵转换，有效避免了诸多技术产品的不足，满足了性能与功能。

项目案例是在 2019 年 3 月完成，时任职于跨越速运大数据中心。项目方案依赖大数据平台实现大量的预计算，矩阵变换是由服务端工程师想出来的，项目完成需要前后端通力配合才能完成。



钉钉扫码加入
“Elasticsearch中文技术社区”
交流群



钉钉扫码加入
“推荐与搜索技术交流群”



微信扫码关注
“搜索与推荐技术”
公众号



阿里云开发者“藏经阁”
海量免费电子书下载