

尚硅谷大数据技术之 Druid

(作者：尚硅谷大数据研发部)

版本：V1.0

第 1 章 初识 Druid

1.1 什么是 Druid

Druid 是一个分布式的支持实时分析的数据存储系统 (Data Store)。美国广告技术公司 MetaMarkets 于 2011 年创建了 Druid 项目，并且于 2012 年晚期开源了 Druid 项目。Druid 设计之初的想法就是为分析而生，它在处理数据的规模、数据处理的实时性方面，比传统的 OLAP 系统有了显著的性能改进，而且拥抱主流的开源生态，包括 Hadoop 等。多年以来，Druid 一直是非常活跃的开源项目。

Druid 的官方网站是 <http://druid.io>。

另外，阿里巴巴也曾创建过一个开源项目叫作 Druid (简称阿里 Druid)，它是一个数据库连接池的项目。阿里 Druid 和本问讨论的 Druid 没有任何关系，它们解决完全不同的问题。

1.2 Druid 的三个设计原则

- 快速查询 (Fast Query)：部分数据的聚合 (Partial Aggregate) + 内存化 (In-memory) + 索引 (Index)。
- 水平扩展能力 (Horizontal Scalability)：分布式数据 (Distributed Data) + 并行化查询 (Parallelizable Query)。
- 实时分析 (Realtime Analytics)：不可变的过去，只追加的未来 (Immutable Past, Append-Only Future)。

1.2.1 快速查询 (Fast Query)

对于数据分析场景，大部分情况下，我们只关心一定粒度聚合的数据，而非每一行原始数据的细节情况。因此，数据聚合粒度可以是 1 分钟、5 分钟、1 小时或 1 天等。部分数据聚合 (Partial Aggregate) 给 Druid 争取了很大的性能优化空间。

数据内存化也是提高查询速度的杀手锏。内存和硬盘的访问速度相差近百倍，但内存的

大小是非常有限的，因此在内存使用方面要精心设计，比如 Druid 里面使用了 Bitmap 和各种压缩技术。

另外，为了支持 Drill-Down 某些维度，Druid 维护了一些倒排索引。这种方式可以加快 AND 和 OR 等计算操作。

1.2.2 水平扩展能力（Horizontal Scalability）

Druid 查询性能在很大程度上依赖于内存的优化使用。数据可以分布在多个节点的内存中，因此当数据增长的时候，可以通过简单增加机器的方式进行扩容。为了保持平衡，Druid 按照时间范围把聚合数据进行分区处理。对于高基数的维度，只按照时间切分有时候是不够的（Druid 的每个 Segment 不超过 2000 万行），故 Druid 还支持对 Segment 进一步分区。历史 Segment 数据可以保存在深度存储系统中，存储系统可以是本地磁盘、HDFS 或远程的云服务。如果某些节点出现故障，则可借助 Zookeeper 协调其他节点重新构造数据。

Druid 的查询模块能够感知和处理集群的状态变化，查询总是在有效的集群架构中进行。集群上的查询可以进行灵活的水平扩展。

1.2.3 实时分析（Realtime Analytics）

Druid 提供了包含基于时间维度数据的存储服务，并且任何一行数据都是历史真实发生的事件，因此在设计之初就约定事件一旦进入系统，就不能再改变。

对于历史数据 Druid 以 Segment 数据文件的方式组织，并且将它们存储到深度存储系统中，例如文件系统或亚马逊的 S3 等。当需要查询这些数据的时候，Druid 再从深度存储系统中将它们装载到内存供查询使用。

1.3 Druid 的主要特点

1. **列式存储格式。** Druid 使用面向列的存储，这意味着，它只需要加载特定查询所需要的列。这为只差看几列的查询提供了巨大的速度提升。此外，每列都针对其特定的数据类型进行优化，支持快速扫描和聚合。
2. **可扩展的分布式系统。** Druid 通常部署在数十到数百台服务器的集群中，并且提供数百万条/秒的摄取率，保留数百万条记录，以及亚秒级到几秒钟的查询延迟。
3. **大规模的并行处理。** Druid 可以在整个集群中进行大规模的并行查询。
4. **实时或批量摄取。** Druid 可以实时摄取数据（实时获取的数据可立即用于查询）或批量处理数据。
5. **自愈，自平衡，易操作。** 集群扩展和缩小，只需添加或删除服务器，集群将在后台

自动重新平衡，无需任何停机时间。

6. **原生云、容错的架构，不会丢失数据。**一旦 Druid 吸收了您的数据，副本就安全地存储在深度存储中（通常是云存储、HDFS 或共享文件系统）。即使每个 Druid 服务器都失败，也可以从深层存储恢复数据。对于仅影响少数 Druid 服务器的更有限的故障，复制确保在系统恢复时仍然可以执行查询。
7. **用于快速过滤的索引。**Druid 使用 CONCISE 或 Roaring 压缩位图索引来创建索引，这些索引可以快速过滤和跨多个列搜索。
8. **近似算法。**Druid 包括用于近似计数、近似排序以及计算近似直方图和分位数的算法。这些算法提供了有限的内存使用，并且通常比精确计算快得多。对于准确度比速度更重要的情况，Druid 还提供精确的计数-明确和准确的排名。
9. **插入数据时自动聚合。**Druid 可选地支持摄取时的数据自动汇总。预先汇总了您的数据，并且可以导致巨大的成本节约和性能提升。

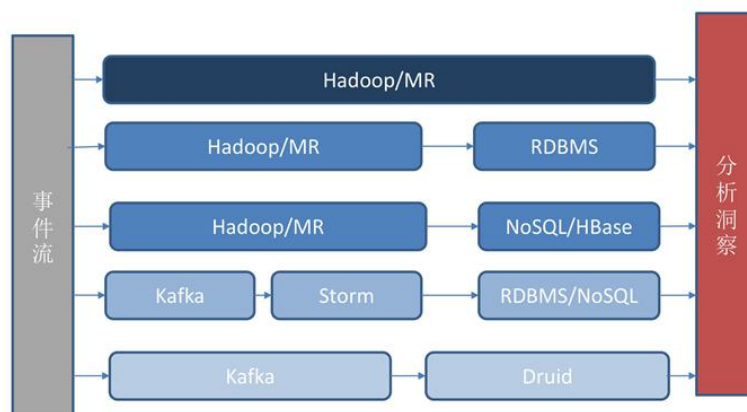
1.4 Druid 的应用场景

Druid 应用最多的是类似于广告分析创业公司 MetaMarkets 中的应用场景，如广告分析、互联网广告系统监控以及网络监控等。当业务中出现以下情况时，Druid 是一个很好的技术方案选择：

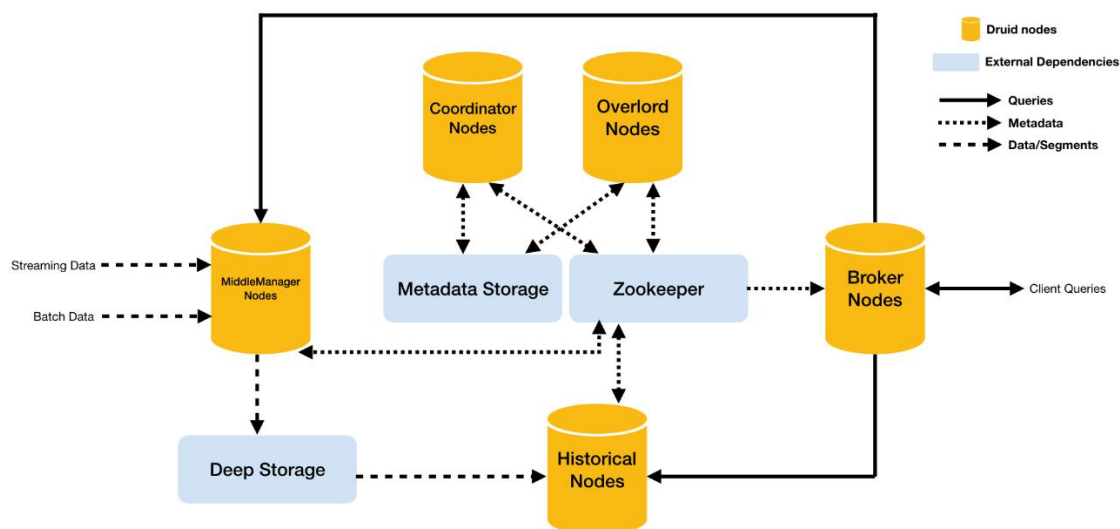
- 需要交互式聚合和快速探究大量数据时；
- 具有大量数据时，如每天数亿事件的新增、每天数 10T 数据的增加；
- 对数据尤其是大数据进行实时分析时；
- 需要一个高可用、高容错、高性能数据库时。



实时数据分析之旅



第 2 章 Druid 的架构



Druid 总体包含以下 5 类节点:

1. **中间管理节点** (middleManager node): 及时摄入实时数据, 已生成 Segment 数据文件。
2. **历史节点** (historical node): 加载已生成好的数据文件, 以供数据查询。historical 节点是整个集群查询性能的核心所在, 因为 historical 会承担绝大部分的 segment 查询。
3. **查询节点** (broker node): 接收客户端查询请求, 并将这些查询转发给 Historicals 和 MiddleManagers。当 Brokers 从这些子查询中收到结果时, 它们会合并这些结果并将它们返回给调用者。
4. **协调节点** (coordinator node): 主要负责历史节点的数据负载均衡, 以及通过规则 (Rule) 管理数据的生命周期。协调节点告诉历史节点加载新数据、卸载过期数据、复制数据、和为了负载均衡移动数据。
5. **统治者** (overlord node): 进程监视 MiddleManager 进程, 并且是数据摄入 Druid 的控制器。他们负责将提取任务分配给 MiddleManagers 并协调 Segement 发布。

同时，Druid 还包含 3 类外部依赖：

1. **数据文件存储库 (DeepStorage)**: 存放生成的 Segment 数据文件, 并供历史服务器下载, 对于单节点集群可以是本地磁盘, 而对于分布式集群一般是 HDFS。
2. **元数据库 (Metastore)**, 存储 Druid 集群的元数据信息, 比如 Segment 的相关信息, 一般用 MySQL 或 PostgreSQL。

3. **Zookeeper**: 为 Druid 集群提供以执行协调服务。如内部服务的监控, 协调和领导者选举。

第 3 章 Druid 的数据结构

与 Druid 架构相辅相成的是其基于 DataSource 与 Segment 的数据结构, 它们共同成就了 Druid 的高性能优势。

3.1. DataSource 结构

若与传统的关系型数据库管理系统 (RDBMS) 做比较, Druid 的 DataSource 可以理解为 RDBMS 中的表 (Table)。DataSource 的结构包含以下几个方面。

1. **时间列 (Timestamp)**: 表明每行数据的时间值, 默认使用 UTC 时间格式且精确到毫秒级别。这个列是数据聚合与范围查询的重要维度。
2. **维度列 (Dimension)**: 维度来自于 OLAP 的概念, 用来标识数据行的各个类别信息。
3. **指标列 (Metric)**: 指标对应于 OLAP 概念中的 Fact, 是用于聚合和计算的列。这些指标列通常是一些数字, 计算操作通常包括 Count、Sum 和 Mean 等。

timestamp	publisher	advertiser	gender	country	click	price
2011-01-01T01:01:35Z	bieberfever.com	google.com	Male	USA	0	0.65
2011-01-01T01:03:63Z	bieberfever.com	google.com	Male	USA	0	0.62
2011-01-01T01:04:51Z	bieberfever.com	google.com	Male	USA	1	0.45
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	1	1.53

时间列 维度列 指标列

DataSource 结构

无论是实时数据消费还是批量数据处理, Druid 在基于 DataSource 结构存储数据时即可选择对任意的指标列进行聚合 (RollUp) 操作。该聚合操作主要基于维度列与时间范围两方面的情况。

下图显示的是执行聚合操作后 DataSource 的数据情况。

timestamp	publisher	advertiser	gender	country	impressions	clicks	revenue
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Male	USA	1800	25	15.70
2011-01-01T01:00:00Z	bieberfever.com	google.com	Male	USA	2912	42	29.18
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Male	UK	1953	17	17.31
2011-01-01T02:00:00Z	bieberfever.com	google.com	Male	UK	3194	170	34.01

DataSource 聚合后的数

相对于其他时序数据库, Druid 在数据存储时便可对数据进行聚合操作是其一大特点, 该特点使得 Druid 不仅能够节省存储空间, 而且能够提高聚合查询的效率。

3.2. Segment 结构

DataSource 是一个逻辑概念，Segment 却是数据的实际物理存储格式，Druid 正是通过 Segment 实现了对数据的横纵向切割（Slice and Dice）操作。从数据按时间分布的角度来看，通过参数 segmentGranularity 的设置，Druid 将不同时间范围内的数据存储在不同的 Segment 数据块中，这便是所谓的数据横向切割。

这种设计为 Druid 带来一个显而易见的优点：按时间范围查询数据时，仅需要访问对应时间段内的这些 Segment 数据块，而不需要进行全表数据范围查询，这使效率得到了极大的提高。

timestamp	page	language	city	country	...	added	deleted
2011-01-01T00:01:35Z	Justin Bieber	en	SF	USA		10	65
2011-01-01T00:03:63Z	Justin Bieber	en	SF	USA		15	62
2011-01-01T00:04:51Z	Justin Bieber	en	SF	USA		32	45
Segment 2011-01-01T00/2011-01-01T01							
2011-01-01T01:00:00Z	Ke\$ha	en	Calgary	CA		17	87
Segment 2011-01-01T01/2011-01-01T02							
2011-01-01T02:00:00Z	Ke\$ha	en	Calgary	CA		43	99
2011-01-01T02:00:00Z	Ke\$ha	en	Calgary	CA		12	53
Segment 2011-01-01T02/2011-01-01T03							

通过 Segment 将数据按时间范围存储，同时，在 Segment 中也面向列进行数据压缩存储，这便是所谓的数据纵向切割。而且在 Segment 中使用了 Bitmap 等技术对数据的访问进行了优化。

第 4 章 Druid 的安装（单机版）

4.1 Jar 包下载

从 <https://imply.io/get-started> 下载最新版本安装包

4.2 Druid 的安装部署

说明：imply 集成了 Druid，提供了 Druid 从部署到配置到各种可视化工具的完整的解决方案，imply 有点类似于我们之前学过的 Cloudera Manager

1.解压

```
tar -zxvf imply-2.7.10.tar.gz -C /opt/module
```

目录说明如下：

- bin/ - run scripts for included software.

- conf/ - template configurations for a clustered setup.

- conf-quickstart/* - configurations for the single-machine quickstart.
- dist/ - all included software.
- quickstart/ - files related to the single-machine quickstart.

2.修改配置文件

1) 修改 Druid 的 ZK 配置

```
[atguigu@hadoop102 _common]$ pwd
/opt/module/implify/conf/druid/_common
[atguigu@hadoop102 _common]$ vi common.runtime.properties
druid.zk.service.host=hadoop102:2181,hadoop103:2181,hadoop104:2181
```

2) 修改启动命令参数, 使其不校验不启动内置 ZK

```
[atguigu@hadoop102 supervise]$ pwd
/opt/module/implify/conf/supervise
:verify bin/verify-java
#:verify bin/verify-default-ports
#:verify bin/verify-version-check
:kill-timeout 10

#!p10 zk bin/run-zk conf-quickstart
```

3.启动

1) 启动 zookeeper

2) 启动 imply

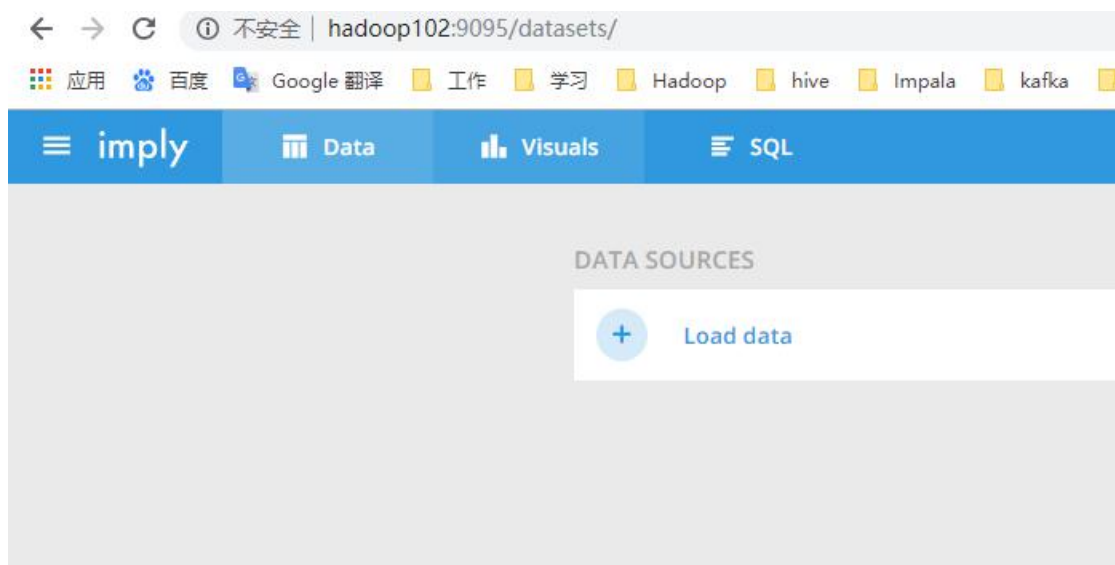
```
[atguigu@hadoop102 imply]$ bin/supervise -c
conf/supervise/quickstart.conf
```

说明: 每启动一个服务均会打印出一条日志。可以通过/opt/module/implify-2.7.10/var/sv/查看服务启动时的日志信息

3) 查看端口号 9095 的启动情况

```
[atguigu@hadoop102 ~]$ netstat -anp | grep 9095
tcp        0      0 :::9095                :::*
LISTEN     3930/implify-ui-linux
tcp        0      0 ::ffff:192.168.1.102:9095 ::ffff:192.168.1.1:52567
ESTABLISHED 3930/implify-ui-linux
tcp        0      0 ::ffff:192.168.1.102:9095 ::ffff:192.168.1.1:52568
ESTABLISHED 3930/implify-ui-linux
```

4. 登录 hadoop102:9095 查看



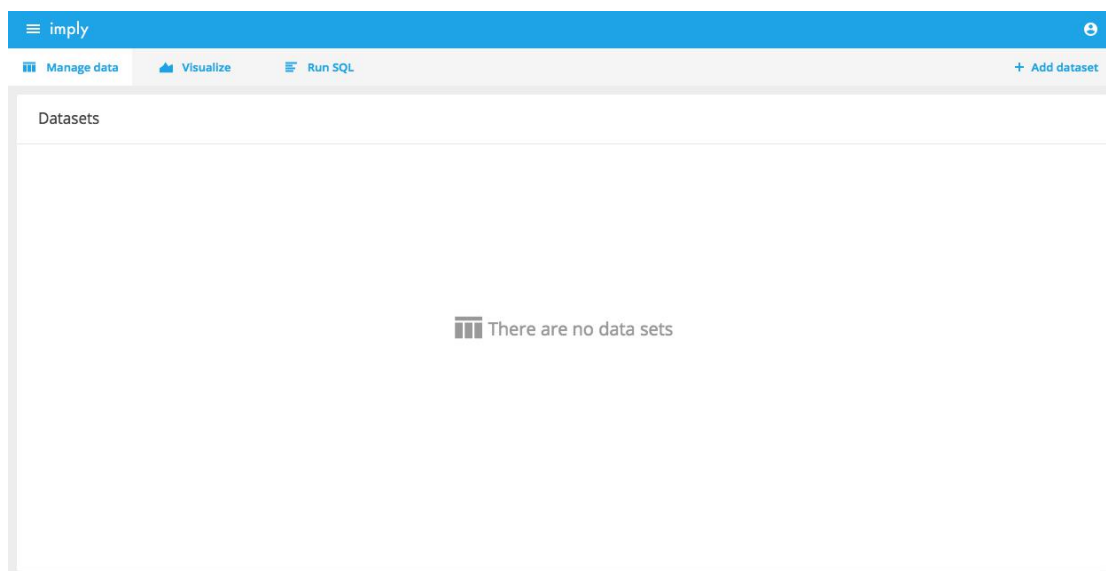
5. 停止服务

按 `Ctrl + c` 中断监督进程，如果想中断服务后进行干净的启动，请删除 `/opt/module/imply-2.7.10/var/` 目录。

第 5 章 Druid 的入门

5.1 在线加载样本数据

1. 打开 imply



2. 开始连接到实例 Wikipedia 数据集

The screenshot shows the 'Add dataset - point to your data' form in the Imply interface. The form has a blue header with the 'imply' logo and navigation tabs for 'Manage data', 'Visualize', and 'Run SQL'. The main content area is titled 'Add dataset - point to your data' and contains the following fields:

- DATASET NAME:** A text input field containing 'wikipedia'.
- URI(S):** A text input field containing 'https://static.imply.io/data/wikipedia.json.gz'.
- FORMAT:** A dropdown menu with 'JSON (new line delimited)' selected.

At the bottom right, there are two buttons: 'Back' and 'Sample and continue'.

3.加载样本数据。Wikipedia 示例使用 Http 数据加载器从 URI 路径读取数据,格式为 json。可以通过点击采样并继续,对文件前几行的数据进行采样,以确保它是可解析的数据。

The screenshot shows the 'Add dataset - rollup' form in the Imply interface. The form has a blue header with the 'imply' logo and navigation tabs for 'Manage data', 'Visualize', and 'Run SQL'. The main content area is titled 'Add dataset - rollup' and contains the following elements:

- Would you like to use rollup in your data?** A question with two radio button options:
- Don't use roll-up:** The selected option. Below it, a description states: 'Druid summarizes this raw data at ingestion time using a process we refer to as "roll-up". Roll-up is a first-level aggregation operation over a selected set of dimensions. [Read more here.](#)'
- Use roll-up:** An unselected option. Below it, a description states: 'Roll-up data as it is ingested to minimize the amount of raw data that needs to be stored. However, as data is rolled up, you lose the ability to query individual events. [Read more here.](#)'

At the bottom right, there are two buttons: 'Back' and 'Next'.

4.配置汇总

imply

Manage dataVisualizeRun SQL

← Add dataset - configure time column

PRIMARY TIME COLUMN

timestamp2016-06-27T00:00:11.080Z

Will be renamed to `__time` in Druid.

TIMESTAMP FORMAT

iso (2015-10-29T23:00:00.000Z)

SEGMENT GRANULARITY

Day

Druid partitions your data by time. This represents the granularity of these partitions.

← Back

Next →

5.配置时间戳和分区

imply

Manage dataVisualizeRun SQL

← Add dataset - configure columns

COLUMNS

Add columns

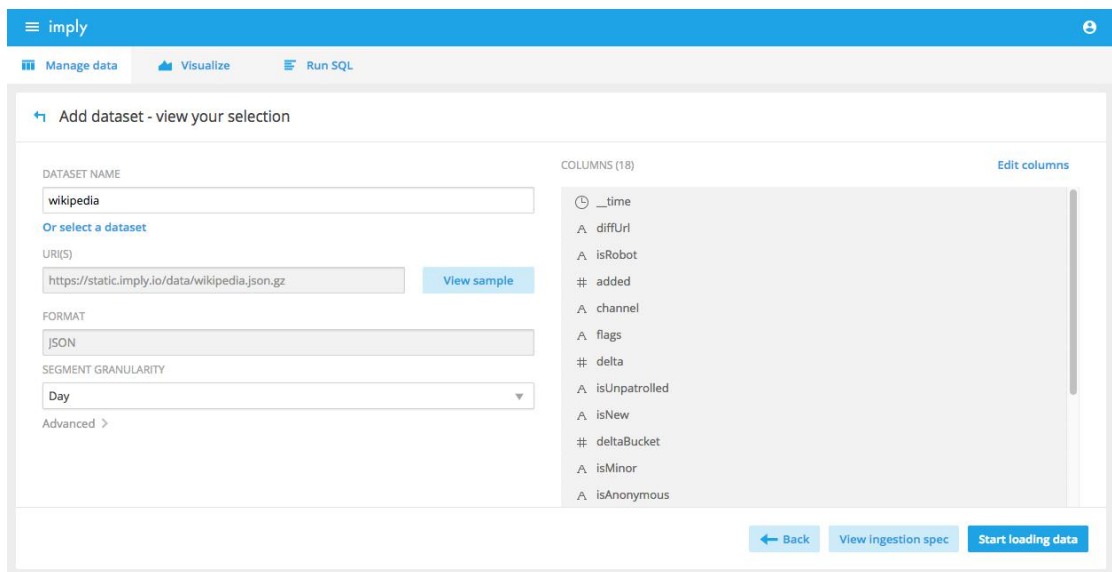
Clear all

🕒	__time (timestamp)	format: iso	✎	✕
A	diffUrl	type: string	✎	✕
A	isRobot	type: string	✎	✕
#	added	type: long	✎	✕
A	channel	type: string	✎	✕
A	flags	type: string	✎	✕
#	delta		✎	✕

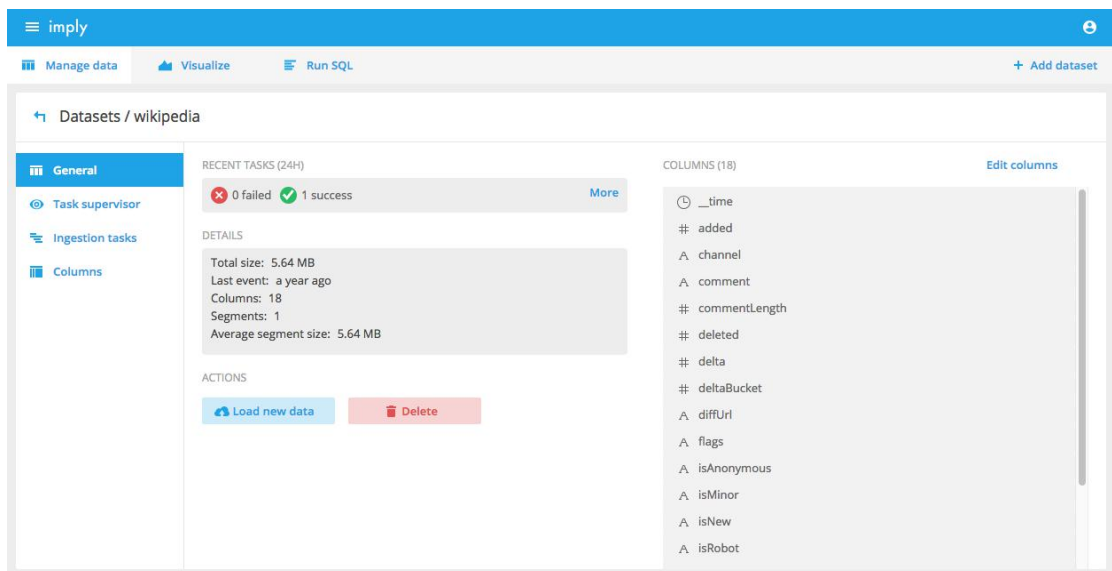
← Back

Next →

6.配置要加载的列。



7.确认并开始摄取！



一旦加载器指示数据已被索引，您就可以继续下一部分来定义数据立方体并开始可视化数据。

5.2 离线加载样本数据

如果您无法访问公共 Web 服务器，则可以从本地文件加载相同的数据集。该 quickstart 目录包括一个样本数据集和一个摄取规范来处理数据，分别命名 wikipedia-2016-06-27-sampled.json 和 wikipedia-index.json。

要为此摄取规范向 Druid 提交索引作业，请从 Imply 目录运行以下命令：

```
bin/post-index-task --file quickstart/wikipedia-index.json
```

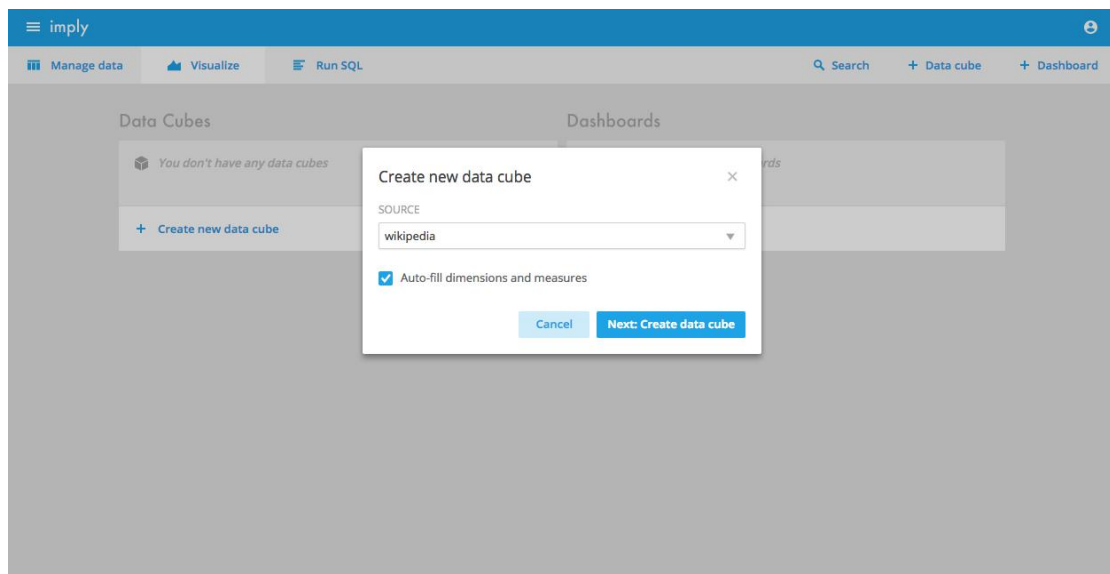
成功运行将生成类似于以下内容的日志：

```
Beginning indexing data for wikipedia
Task started: index_wikipedia_2017-12-05T03:22:28.612Z
```

```
Task                                                                    log:
http://localhost:8090/druid/indexer/v1/task/index_wikipedia_2017-
12-05T03:22:28.612Z/log
Task                                                                    status:
http://localhost:8090/druid/indexer/v1/task/index_wikipedia_2017-
12-05T03:22:28.612Z/status
Task index_wikipedia_2017-12-05T03:22:28.612Z still running...
Task index_wikipedia_2017-12-05T03:22:28.612Z still running...
Task finished with status: SUCCESS
Completed indexing data for wikipedia. Now loading indexed data onto
the cluster...
wikipedia is 0.0% finished loading...
wikipedia is 0.0% finished loading...
wikipedia is 0.0% finished loading...
wikipedia loading complete! You may now query your data
```

5.3 创建数据立方体

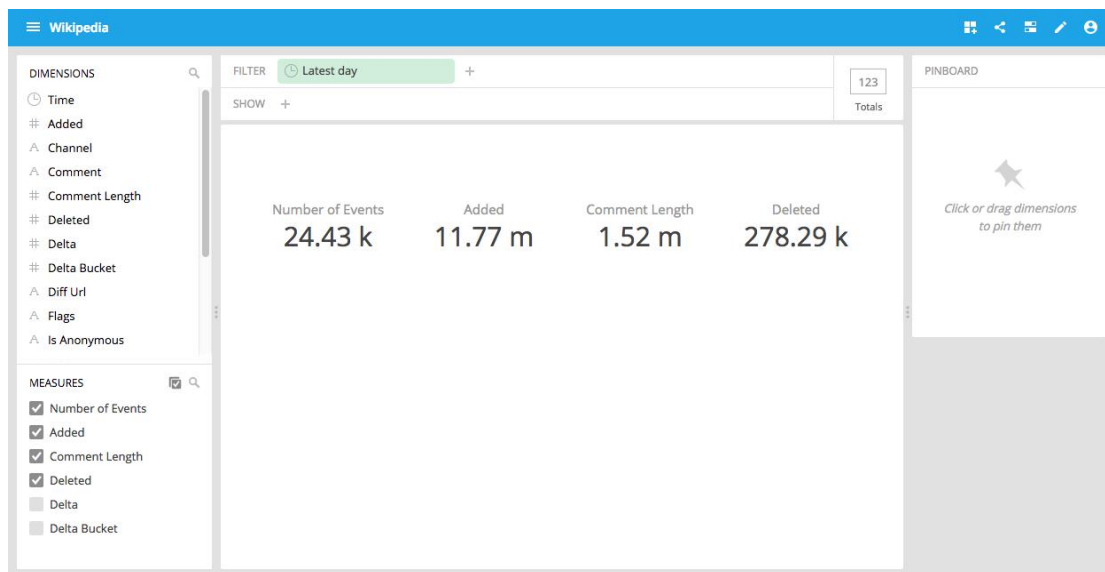
通过单击顶部栏上的相应按钮切换到 **Imply** 的“可视化”部分。从这里，您可以创建数据立方体来建模数据，浏览这些立方体，并将视图组织到仪表板中。首先单击+创建新数据多维数据集。



在出现的对话框中，确保 **wikipedia** 选中此源并选择自动填充尺寸和度量。单击下一步继续：创建数据立方体。

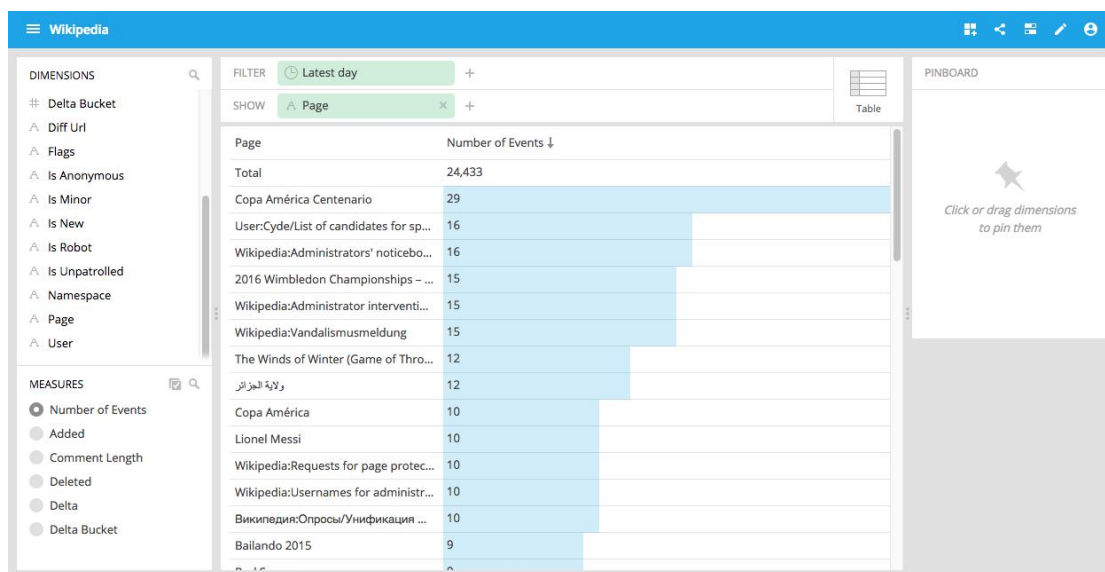
5.4 可视化数据立方体

单击“保存”后，将自动加载此新数据多维数据集的数据立方体视图。将来，还可以通过从“可视化”屏幕单击数据立方体的名称（在此示例中为“Wikipedia”）来加载此视图。

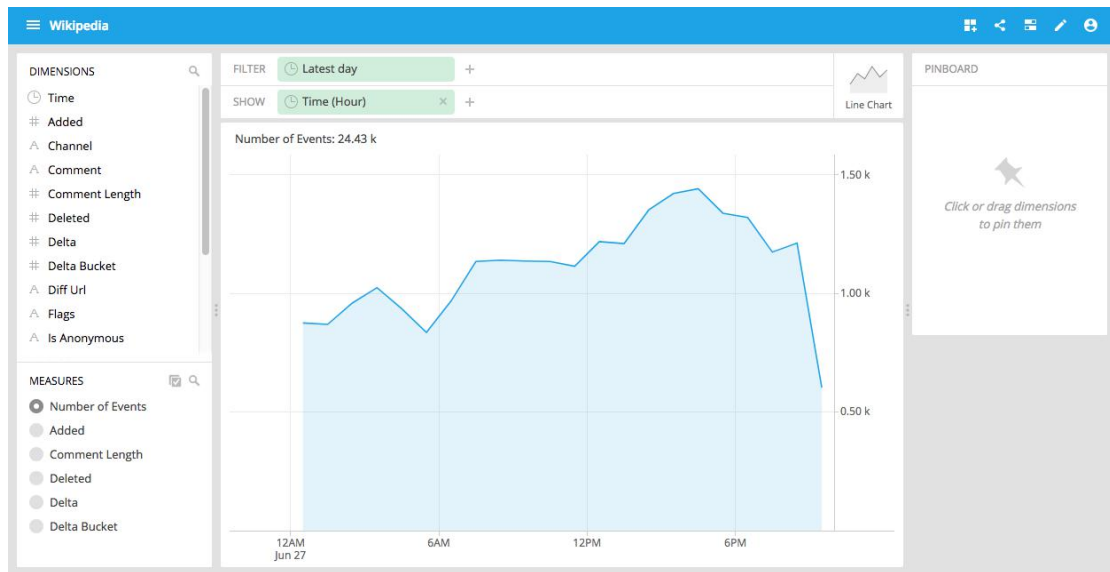


在这里，您可以通过过滤并在任何维度上拆分数据集来探索数据集。对于数据的每次过滤拆分，您将看到所选度量的总值。

例如，在维基百科数据集上，通过在 **page** 上拆分和按事件数排序查看最常编辑的 **page**。



数据立方体视图根据您的分割数据的方式建议不同的可视化。如果拆分字符串列，则数据最初将显示为表格。如果按时间拆分，数据立方体视图将推荐时间序列图，如果在数字列上拆分，则会得到条形图。



5.5 运行 SQL

访问 SQL 编辑器。

```
SELECT page, COUNT(*) AS Edits
FROM wikipedia
WHERE "__time" BETWEEN TIMESTAMP '2016-06-27 00:00:00' AND TIMESTAMP '2016-06-28 00:00:00'
GROUP BY page
ORDER BY Edits
DESC LIMIT 5
```

您应该看到如下结果：

page	Edits
Copa América Centenario	29
User:Cyde/List of candidates ...	16
Wikipedia:Administrators' no...	16
2016 Wimbledon Champions...	15
Wikipedia:Administrator inte...	15

第 6 章 数据摄入

6.1 数据格式

- 1) 摄入规范化数据：JSON、CSV、TSV

2) 自定义格式

3) 其他格式

6.2 配置

主要是摄入的规则 ingestion Spec

摄入规则主要包含 3 个部分

Field	Type	Description	Required
dataSchema	JSON Object	标识摄入数据的 schema，不同 specs 可共享	yes
ioConfig	JSON Object	标识 data 从哪来，到哪去。根据不同的 ingestion method 不同	yes
tuningConfig	JSON Object	标识如何调优不同的 ingestion parameters。根据不同的 ingestion method 不同	no

```
{
  "dataSchema" : {...},
  "ioConfig" : {...},
  "tuningConfig" : {...}
}
```

6.2.1 DataSchema

Field	Type	Description	Required
dataSource	String	要摄入的 datasource 名称，Datasources 可看做为表	yes
parser	JSON Object	ingested data 如何解析	yes
metricsSpec	JSON Object array	aggregators 器列表	yes
granularitySpec	JSON Object	指定 segment 的存储粒度和查询粒度	yes

1、parser

(1) string parser

Field	Type	Description	Required
type	String	一般为 string，或在 Hadoop indexing job 中使用 hadoopString	no
parseSpec	JSON Object	标识格式 format 和、timestamp、dimensions	yes

parseSpec 两个功能:

➤ String Parser 用 parseSpec 判定将要处理 rows 的数据格式 (JSON, CSV, TSV)

- 所有的 Parsers 用 parseSpec 判定将要处理 rows 的 timestamp 和 dimensionsAll format 字段默认为 tsv 格式

JSON ParseSpec

Field	Type	Description	Required
format	String	json	no
timestampSpec	JSON Object	指明时间戳列名和格式	yes
dimensionsSpec	JSON Object	指明维度的设置	yes
flattenSpec	JSON Object	若 json 有嵌套层级，则需要指定	no

CSV ParseSpec

Field	Type	Description	Required
format	String	csv.	yes
timestampSpec	JSON Object	指明时间戳列名和格式	yes
dimensionsSpec	JSON Object	指明维度的设置	yes
listDelimiter	String	多值 dimensions 的分割符	no (default == ctrl+A)
columns	JSON array	csv 的数据列名	yes

TSV ParseSpec

Field	Type	Description	Required
format	String	tsv.	yes
timestampSpec	JSON Object	指明时间戳列名和格式	yes
dimensionsSpec	JSON Object	指明维度的设置	yes
listDelimiter	String	多值 dimensions 的分割符	no (default == ctrl+A)
columns	JSON array	tsv 的数据列名	yes
delimiter	String	数据之间的分隔符，默认是\t	no

TimestampSpec

Field	Type	Description	Required
column	String	timestamp 的列	yes
format	String	iso, millis, posix, auto or Joda time, 时间戳格式	no (default == 'auto')

DimensionsSpec

Field	Type	Description	Required
-------	------	-------------	----------

dimensions	JSON array	dimension schema 对象或 dimension names, 标识维度列, 否则将 timestamp 列外的所以 string 列作为维度列	yes
dimensionExclusions	JSON String array	剔除的维度名列表	no (default == [])
spatialDimensions	JSON Object array	空间维度名列表, 主要用于地理几何运算	no (default == [])

2、metricsSpec

Field	Type	Description	Required
dimensions	string	count, longSum 等聚合函数类型	yes
fieldName	string	聚合函数运用的列名	no
name	string	聚合后指标的列名	yes

一些简单的聚合函数:

count、longSum、longMin、longMax、doubleSum、doubleMin、doubleMax

3、GranularitySpec

Field	Type	Description	Required
type	string	uniform	yes
segmentGranularity	string	segment 的存储粒度, HOUR DAY 等	yes
queryGranularity	string	最小查询粒度 MINUTE HOUR	yes
intervals	JSON Object array	输入数据的时间段, 可选, 对于流式数据 pull 方式而言可以忽略	no

```

"dataSchema" : {
  "dataSource" : "wikipedia",
  "parser" : {
    "type" : "string",
    "parseSpec" : {
      "format" : "json",
      "dimensionsSpec" : {
        "dimensions" : [
          "channel",
          "cityName",
          "comment",
          "countryIsoCode",

```

```

        "countryName",
        "isAnonymous",
        "isMinor",
        "isNew",
        "isRobot",
        "isUnpatrolled",
        "metroCode",
        "namespace",
        "page",
        "regionIsoCode",
        "regionName",
        "user",
        { "name" : "commentLength", "type" : "long" },
        { "name" : "deltaBucket", "type" : "long" },
        "flags",
        "diffUrl",
        { "name": "added", "type": "long" },
        { "name": "deleted", "type": "long" },
        { "name": "delta", "type": "long" }
    ]
},
"timestampSpec": {
    "column": "timestamp",
    "format": "iso"
}
}
},
"metricsSpec" : [],
"granularitySpec" : {
    "type" : "uniform",
    "segmentGranularity" : "day",
    "queryGranularity" : "none",
    "intervals" : ["2016-06-27/2016-06-28"],
    "rollup" : false
}
}
}

```

6.2.2 ioConfig

ioConfig 指明了真正具体的数据源

Field	Type	Description	Required
type	String	always be 'realtime'.	yes
firehose	JSON Object	指明数据源，例如本地文件 kafka	yes
plumber	JSON Object	Where the data is going.	yes

不同的 firehose 的格式不太一致，以 kafka 为例

```

{
    firehose : {
        consumerProps : {
            auto.commit.enable : false
            auto.offset.reset : largest
            fetch.message.max.bytes : 1048586
            group.id : druid-example
            zookeeper.connect : localhost:2181

```

```

        zookeeper.connect.timeout.ms : 15000
        zookeeper.session.timeout.ms : 15000
        zookeeper.sync.time.ms : 5000
    },
    feed : wikipedia
    type : kafka-0.8
}
}

```

ioConfig 的案例:

```

"ioConfig" : {
  "type" : "index",
  "firehose" : {
    "type" : "local",
    "baseDir" : "quickstart/",
    "filter" : "wikipedia-2016-06-27-sampled.json"
  },
  "appendToExisting" : false
}

```

6.2.3 tuningConfig

tuningConfig 这部分的配置是优化数据输入的过程

Field	Type	Description	Required
type	String	realtime	no
maxRowsInMemory	Integer	在存盘之前内存中最大的存储行数, 指的是聚合后的行数 indexing 所需 Maximum heap memory= maxRowsInMemory * (2 + maxPendingPersists).	no (default == 75000)
windowPeriod	ISO 8601 Period String	默认 10 分钟, 最大可容忍时间窗口, 超过窗口, 数据丢弃	no (default == PT10m)
intermediatePersistPeriod	ISO8601 Period String	多长时间数据临时存盘一次	no (default == PT10m)
basePersistDirectory	String	临时存盘目录	no (default == java tmp dir)
versioningPolicy	Object	如何为 segment 设置版本号	no (default == based on segment start time)
rejectionPolicy	Object	数据丢弃策略	no (default == 'serverTime')

maxPendingPersists	Integer	最大同时存盘请求数，达到上限，输入将会暂停	no (default == 0)
shardSpec	Object	分片设置	no (default == 'NoneShardSpec')
buildV9Directly	Boolean	是否直接构建 V9 版本的索引	no (default == true)
persistThreadPriority	int	存盘线程优先级	no (default == 0)
mergeThreadPriority	int	存盘归并线程优先级	no (default == 0)
reportParseExceptions	Boolean	是否汇报数据解析错误	no (default == false)

```
"tuningConfig" : {
  "type" : "index",
  "targetPartitionSize" : 5000000,
  "maxRowsInMemory" : 25000,
  "forceExtendableShardSpecs" : true
}
```

6.3 从 hadoop 加载数据

6.3.1 加载数据

批量摄取维基百科样本数据，文件位于 `quickstart/wikipedia-2016-06-27-sampled.json`。使用 `quickstart/wikipedia-index-hadoop.json` 摄取任务文件。

```
bin/post-index-task --file quickstart/wikipedia-index-hadoop.json
```

此命令将启动 Druid Hadoop 摄取任务。

摄取任务完成后，数据将由历史节点加载，并可在一两分钟内进行查询。

6.3.2 查询数据

The screenshot shows the Imply SQL interface. The top navigation bar has tabs for 'imply', 'Data', 'Visuals', and 'SQL'. The 'SQL' tab is selected, and the query editor shows the query: `select * from wikipedia`. Below the query editor is a 'Run again' button. The 'QUERY RESULT' section displays a table with the following columns: `__time`, `added`, `channel`, `cityName`, `comment`, and `commentLength`. The table contains 10 rows of data, including comments from various channels like `#sv.wikipedia`, `#ja.wikipedia`, `#en.wikipedia`, `#sh.wikipedia`, and `#pl.wikipedia`.

__time	added	channel	cityName	comment	commentLength
2016-06-27T00:00:11.080Z	31	#sv.wikipedia		Botskapande Indonesien om...	35
2016-06-27T00:00:17.457Z	125	#ja.wikipedia		70年代	4
2016-06-27T00:00:34.959Z	2	#en.wikipedia	Buenos Aires	/* Scores */	12
2016-06-27T00:00:36.027Z	0	#en.wikipedia		standard term is [[title chara...	36
2016-06-27T00:00:46.874Z	0	#sh.wikipedia		Bot: Automatska zamjena te...	118
2016-06-27T00:00:56.913Z	76	#en.wikipedia		linking to other wikipedia pa...	32
2016-06-27T00:00:58.599Z	270	#pl.wikipedia		utworzenie kategorii	20
2016-06-27T00:01:01.364Z	0	#sh.wikipedia		Bot: Automatska zamjena te...	118
2016-06-27T00:01:03.685Z	0	#sh.wikipedia		Bot: Automatska zamjena te...	118

6.4 从 kafka 加载数据

6.4.1 准备 kafka

1.启动 kafka

```
[atguigu@hadoop102 kafka]$ bin/kafka-server-start.sh config/server.properties
[atguigu@hadoop103 kafka]$ bin/kafka-server-start.sh config/server.properties
[atguigu@hadoop104 kafka]$ bin/kafka-server-start.sh config/server.properties
```

2.创建 wikipedia 主题

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102:2181 --topic
wikipedia --partitions 1 --replication-factor 1 --create
Created topic "wikipedia".
```

3.查看主题是否创建成功

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102:2181 --list
__consumer_offsets
first
wikipedia
```




6.4.2 启动索引服务

我们将使用 Druid 的 Kafka 索引服务从我们新创建的维基百科主题中提取消息。要启动该服务，我们需要通过从 `Impley` 目录运行以下命令向 Druid 的 `overlord` 提交 supervisor spec

```
[atguigu@hadoop102 impley-2.7.10]$ curl -XPOST -H'Content-Type: application/json' -d
@quickstart/wikipedia-kafka-supervisor.json
http://hadoop102:8090/druid/indexer/v1/supervisor
```

说明：

`curl` 是一个利用 URL 规则在命令行下工作的文件传输工具。它支持文件的上传和下载，所以是综合传输工具。

-  `-X` 为 HTTP 数据包指定一个方法，比如 PUT、DELETE。默认的方法是 GET 6.4.3
-  `-H` 为 HTTP 数据包指定 Header 字段内容
-  `-d` 为 POST 数据包指定要向 HTTP 服务器发送的数据并发送出去，如果<data>的内容以符号 @ 开头，其后的字符串将被解析为文件名，`curl` 命令会从这个文件中读取数据发送。

6.4.3 加载历史数据

启动 kafka 生产者生产数据

```
[atguigu@hadoop102 kafka]$ bin/kafka-console-producer.sh --broker-list hadoop102:9092 -
-topic wikipedia < /opt/module/impley-2.7.10/quickstart/wikipedia-2016-06-27-sampled.json
```

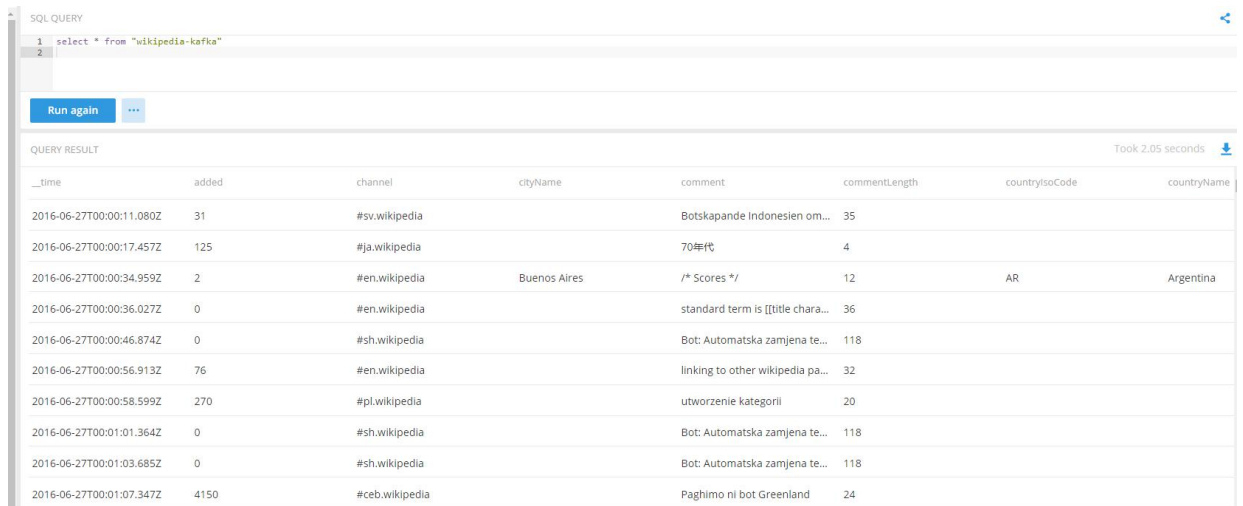
说明：

< 将文件作为命令输入

可在 kafka 本地看到相应的数据生成

```
[atguigu@hadoop103 logs]$ ll
drwxrwxr-x. 2 atguigu atguigu 4096 3月 30 11:16 wikipedia-0
[atguigu@hadoop103 logs]$ pwd
/opt/module/kafka/logs
```

将样本事件发布到 Kafka 的 wikipedia 主题，然后由 Kafka 索引服务将其提取到 Druid 中。
你现在准备运行一些查询了！



SQL QUERY

```
1 select * from "wikipedia-kafka"
```

Run again

QUERY RESULT

Took 2.05 seconds

_time	added	channel	cityName	comment	commentLength	countryIsoCode	countryName
2016-06-27T00:00:11.080Z	31	#sv.wikipedia		Botskapande Indonesien om...	35		
2016-06-27T00:00:17.457Z	125	#ja.wikipedia		70年代	4		
2016-06-27T00:00:34.959Z	2	#en.wikipedia	Buenos Aires	/[* Scores */	12	AR	Argentina
2016-06-27T00:00:36.027Z	0	#en.wikipedia		standard term is [[title chara...	36		
2016-06-27T00:00:46.874Z	0	#sh.wikipedia		Bot: Automatska zamjena te...	118		
2016-06-27T00:00:56.913Z	76	#en.wikipedia		linking to other wikipedia pa...	32		
2016-06-27T00:00:58.599Z	270	#pl.wikipedia		utworzenie kategorii	20		
2016-06-27T00:01:01.364Z	0	#sh.wikipedia		Bot: Automatska zamjena te...	118		
2016-06-27T00:01:03.685Z	0	#sh.wikipedia		Bot: Automatska zamjena te...	118		
2016-06-27T00:01:07.347Z	4150	#ceb.wikipedia		Paghimo ni bot Greenland	24		

6.4.4 加载实时数据

下载一个帮助应用程序，该应用程序将解析维基媒体的 IRC 提要中的 event，并将这些 event 发布到我们之前设置的 Kafka 的 wikipedia 主题中。

```
[atguigu@hadoop102 imply-2.7.10]$ curl -O
https://static.imply.io/quickstart/wikiticker-0.4.tar.gz
```

说明：

-O 在本地保存获取的数据时，使用她们在远程服务器上的文件名进行保存。

```
[atguigu@hadoop102 imply-2.7.10]$ tar -zxvf wikiticker-0.4.tar.gz
[atguigu@hadoop102 imply-2.7.10]$ cd wikiticker-0.4
```

现在运行带有参数的 wikiticker，指示它将输出写入我们的 Kafka 主题：

```
[atguigu@hadoop102 wikiticker-0.4]$ bin/wikiticker -J-Dfile.encoding=UTF-8 -out kafka -
topic Wikipedia
```

查询多次，对比结果的变化

SQL QUERY	
1	SELECT count(*) from "wikipedia-kafka";
Run ...	
QUERY RESULT	
Day	Edits
2019-03-30T00:00:00.000Z	1241

6.4.5 加载自定义 kafka 主题数据

可以通过编写自定义 supervisor spec 来加载自己的数据集。要自定义受监督的 Kafka 索引服务提取,您可以将包含的 quickstart/wikipedia-kafka-supervisor.json 规范复制到自己的文件中,根据需要进行编辑,并根据需要创建或关闭管理程序。没有必要自己重启 Imply 或 Druid 服务。

第 7 章 Druid 数据摄入之 Tranquility

在上面的内容中,我们学习了如何通过索引服务器摄取数据,然而索引服务器的 API 太过底层,运用起来比较麻烦。Tranquility 对索引服务的 API 进行了封装,可以方便创建任务,处理分片,复制、服务器发现以及无缝的数据结构调整。

向 Druid 发送数据,需将 Tranquility 库依赖到程序中。

地址: <https://github.com/druid-io/tranquility>

Tranquility 提供了两种 Core API:

- Tranquilizer, 一个高级 API, 发送单个 message。
- Beam, 一种低级 API, 发送批次数据。

7.1 高级 API

7.1.1 环境准备

依赖

```
<dependencies>
  <dependency>
    <groupId>io.druid</groupId>
    <artifactId>tranquility-core_2.11</artifactId>
```

```

        <version>0.8.2</version>
    </dependency>
    <dependency>
        <groupId>com.metamx</groupId>
        <artifactId>java-util</artifactId>
        <version>1.3.2</version>
    </dependency>
</dependencies>

```

数据格式的配置:

```

{
  "dataSources": [
    {
      "spec": {
        "dataSchema": {
          "dataSource": "wikipedia02",
          "parser": {
            "type": "string",
            "parseSpec": {
              "format": "json",
              "timestampSpec": {
                "column": "timestamp",
                "format": "auto"
              },
            },
            "dimensionsSpec": {
              "dimensions": [
                "page",
                "language",
                "user",
                "unpatrolled",
                "newPage",
                "robot",
                "anonymous",
                "namespace",
                "continent",
                "country",
                "region",
                "city"
              ],
              "dimensionExclusions": [],
              "spatialDimensions": []
            }
          }
        },
        "metricsSpec": [
          {
            "type": "count",
            "name": "count"
          },
          {
            "type": "doubleSum",
            "name": "added",
            "fieldName": "added"
          },
          {
            "type": "doubleSum",
            "name": "deleted",
            "fieldName": "deleted"
          },
          {
            "type": "doubleSum",
            "name": "delta",
            "fieldName": "delta"
          }
        ]
      }
    }
  ]
}

```



```

    }
  ],
  "granularitySpec": {
    "type": "uniform",
    "segmentGranularity": "DAY",
    "queryGranularity": "NONE"
  },
  "tuningConfig": {
    "type": "realtime",
    "maxRowsInMemory": 100000,
    "intermediatePersistPeriod": "PT10m",
    "windowPeriod": "PT10M"
  }
}
],
"properties": {
  "zookeeper.connect": "hadoop102:2181",
  "druid.selectors.indexing.serviceName": "druid/overlord",
  "druid.discovery.curator.path": "/druid/discovery",
  "druidBeam.taskLocator": "overlord",
  "druidBeam.overlordPollPeriod": "PT5S"
}
}

```

7.1.2 Java 代码编写

```

package com.atguigu.tranquility;
import com.google.common.collect.ImmutableMap;
import com.metamx.common.logger.Logger;
import com.metamx.tranquility.config.DataSourceConfig;
import com.metamx.tranquility.config.PropertiesBasedConfig;
import com.metamx.tranquility.config.TranquilityConfig;
import com.metamx.tranquility.druid.DruidBeams;
import com.metamx.tranquility.tranquilizer.MessageDroppedException;
import com.metamx.tranquility.tranquilizer.Tranquilizer;
import com.twitter.util.FutureEventListener;
import org.joda.time.DateTime;
import scala.runtime.BoxedUnit;

import java.io.InputStream;
import java.util.Map;

public class JavaExample {
    private static final Logger log = new Logger(JavaExample.class);

    public static void main(String[] args)
    {
        // Read config from "example.json" on the classpath.
        final InputStream configStream =
JavaExample.class.getClassLoader().getResourceAsStream("example.json");
        final TranquilityConfig<PropertiesBasedConfig> config =
TranquilityConfig.read(configStream);
        final DataSourceConfig<PropertiesBasedConfig> wikipediaConfig =
config.getDataSource("wikipedia02");
        final Tranquilizer<Map<String, Object>> sender =
DruidBeams.fromConfig(wikipediaConfig)
            .buildTranquilizer(wikipediaConfig.tranquilizerBuilder());

        sender.start();
    }
}

```

```

    try {
        // Send 10000 objects

        for (int i = 0; i < 10; i++) {
            // Build a sample event to send; make sure we use a current date
            final Map<String, Object> obj = ImmutableMap.<String, Object>of(
                "timestamp", new DateTime().toString(),
                "page", "foo",
                "added", i
            );

            // Asynchronously send event to Druid:
            sender.send(obj).addEventListener(
                new FutureEventListener<BoxedUnit>()
                {
                    @Override
                    public void onSuccess(BoxedUnit value)
                    {
                        log.info("Sent message: %s", obj);
                    }

                    @Override
                    public void onFailure(Throwable e)
                    {
                        if (e instanceof MessageDroppedException) {
                            log.warn(e, "Dropped message: %s", obj);
                        } else {
                            log.error(e, "Failed to send message: %s", obj);
                        }
                    }
                }
            );
        }
    } finally {
        sender.flush();
        sender.stop();
    }
}
}

```

7.1.3 scala 代码编写

```

package com.atguigu.tranquility

import com.metamx.common.scala.Logging
import com.metamx.tranquility.config.DataSourceConfig
import com.metamx.tranquility.config.PropertiesBasedConfig
import com.metamx.tranquility.config.TranquilityConfig
import com.metamx.tranquility.druid.DruidBeams
import com.metamx.tranquility.tranquilizer.MessageDroppedException
import com.metamx.tranquility.tranquilizer.Tranquilizer
import com.twitter.util.Return
import com.twitter.util.Throw
import org.joda.time.DateTime
import scala.collection.JavaConverters._
object ScalaExample extends Logging
{
    def main(args: Array[String]) {
        // Read config from "example.json" on the classpath.
        val configStream = getClass.getClassLoader.getResourceAsStream("example.json")
        val config: TranquilityConfig[PropertiesBasedConfig] =
TranquilityConfig.read(configStream)
    }
}

```

```

    val      wikipediaConfig:      DataSourceConfig[PropertiesBasedConfig] =
config.getDataSource("wikipedia03")
    val sender: Tranquilizer[java.util.Map[String, AnyRef]] = DruidBeams
      .fromConfig(config.getDataSource("wikipedia03"))
      .buildTranquilizer(wikipediaConfig.tranquilizerBuilder())

    sender.start()

    try {
      // Send 10000 objects.

      for (i <- 0 until 10) {
        val obj = Map[String, AnyRef](
          "timestamp" -> new DateTime().toString,
          "page" -> "foo",
          "added" -> Int.box(i)
        )

        // Asynchronously send event to Druid:
        sender.send(obj.asJava) respond {
          case Return(_) =>
            log.info("Sent message: %s", obj)

          case Throw(e: MessageDroppedException) =>
            log.warn(e, "Dropped message: %s", obj)

          case Throw(e) =>
            log.error(e, "Failed to send message: %s", obj)
        }
      }
    } finally {
      sender.flush()
      sender.stop()
    }
  }
}

```

7.2 低级 API

Druid 整合 SparkStreaming

7.2.1 环境准备

```

<dependencies>
  <dependency>
    <groupId>io.druid</groupId>
    <artifactId>tranquility-core_2.11</artifactId>
    <version>0.8.2</version>
  </dependency>
  <dependency>
    <groupId>com.metamx</groupId>
    <artifactId>java-util</artifactId>
    <version>1.3.2</version>
  </dependency>
  <dependency>
    <groupId>io.druid</groupId>
    <artifactId>tranquility-spark_2.11</artifactId>
    <version>0.8.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>

```

```

    <artifactId>spark-streaming_2.11</artifactId>
    <version>2.1.1</version>
  </dependency>
</dependencies>

```

7.2.2 代码编写

```

class SimpleEventBeamFactory extends BeamFactory[SimpleEvent]
{
  // Return a singleton, so the same connection is shared across all tasks in the same JVM.
  def makeBeam: Beam[SimpleEvent] = SimpleEventBeamFactory.BeamInstance
}

object SimpleEventBeamFactory
{
  val BeamInstance: Beam[SimpleEvent] = {
    // Tranquility uses ZooKeeper (through Curator framework) for coordination.
    val curator = CuratorFrameworkFactory.newClient(
      "localhost:2181",
      new BoundedExponentialBackoffRetry(100, 3000, 5)
    )
    curator.start()

    val indexService = "druid/overlord" // Your overlord's druid.service, with slashes
replaced by colons.
    val discoveryPath = "/druid/discovery" // Your overlord's
druid.discovery.curator.path
    val dataSource = "foo"
    val dimensions = IndexedSeq("bar")
    val aggregators = Seq(new LongSumAggregatorFactory("baz", "baz"))
    val isRollup = true

    // Expects simpleEvent.timestamp to return a Joda DateTime object.
    DruidBeams
      .builder((simpleEvent: SimpleEvent) => simpleEvent.timestamp)
      .curator(curator)
      .discoveryPath(discoveryPath)
      .location(DruidLocation(indexService, dataSource))
      .rollup(DruidRollup(SpecificDruidDimensions(dimensions), aggregators,
QueryGranularities.MINUTE, isRollup))
      .tuning(
        ClusteredBeamTuning(
          segmentGranularity = Granularity.HOUR,
          windowPeriod = new Period("PT10M"),
          partitions = 1,
          replicants = 1
        )
      )
      .buildBeam()
  }
}

// Add this import to your Spark job to be able to propagate events from any RDD to Druid
import com.metamx.tranquility.spark.BeamRDD._

// Now given a Spark DStream, you can send events to Druid.
dstream.foreachRDD(rdd => rdd.propagate(new SimpleEventBeamFactory))

```