大数据项目之电商数仓（及时查询）

版本：V4.0

# 第1章 Presto

## 1.1 Presto简介

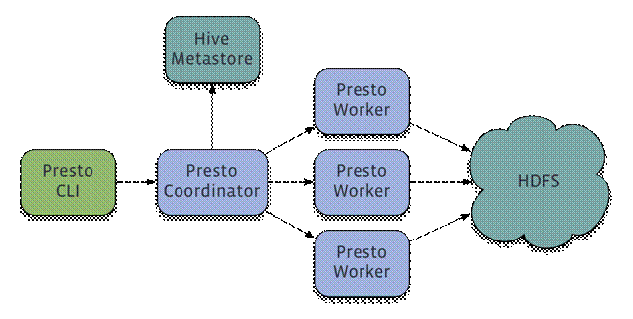
### 1.1 Presto概念

Presto是一个开源的分布式SQL查询引擎，数据量支持GB到PB字节，主要用来处理秒级查询的场景。

注意：虽然Presto可以解析SQL，但它不是一个标准的数据库。不是MySQL、Oracle的代替品，也不能用来处理在线事务（OLTP）。

### 1.2 Presto架构

Presto是一个运行在多台服务器上的分布式系统。完整安装包括一个Coordinator和多个Worker。由客户端提交查询，从Presto命令行CLI提交到Coordinator。Coordinator进行解析，分析并执行查询计划，然后分发处理队列到Worker。



Presto有两类服务器：Coordinator和Worker。

1）Coordinator

Coordinator服务器是用来解析语句，执行计划分析和管理Presto的Worker结点。Presto安装必须有一个Coordinator和多个Worker。如果用于开发环境和测试，则一个Presto实例可以同时担任这两个角色。

Coordinator跟踪每个Work的活动情况并协调查询语句的执行。Coordinator为每个查询建立模型，模型包含多个Stage，每个Stage再转为Task分发到不同的Worker上执行。

Coordinator与Worker、Client通信是通过REST API。

2）Worker

Worker是负责执行任务和处理数据。Worker从Connector获取数据。Worker之间会交换中间数据。Coordinator是负责从Worker获取结果并返回最终结果给Client。

当Worker启动时，会广播自己去发现 Coordinator，并告知 Coordinator它是可用，随时可以接受Task。

Worker与Coordinator、Worker通信是通过REST API。

3）数据源

贯穿全文，你会看到一些术语：Connector、Catelog、Schema和Table。这些是Presto特定的数据源

（1）Connector

Connector是适配器，用于Presto和数据源（如Hive、RDBMS）的连接。你可以认为类似JDBC那样，但却是Presto的SPI的实现，使用标准的API来与不同的数据源交互。

Presto有几个内建Connector：JMX的Connector、System Connector（用于访问内建的System table）、Hive的Connector、TPCH（用于TPC-H基准数据）。还有很多第三方的Connector，所以Presto可以访问不同数据源的数据。

每个Catalog都有一个特定的Connector。如果你使用catelog配置文件，你会发现每个文件都必须包含connector.name属性，用于指定catelog管理器（创建特定的Connector使用）。一个或多个catelog用同样的connector是访问同样的数据库。例如，你有两个Hive集群。你可以在一个Presto集群上配置两个catelog，两个catelog都是用Hive Connector，从而达到可以查询两个Hive集群。

（2）Catelog

一个Catelog包含Schema和Connector。例如，你配置JMX的catelog，通过JXM Connector访问JXM信息。当你执行一条SQL语句时，可以同时运行在多个catelog。

Presto处理table时，是通过表的完全限定（fully-qualified）名来找到catelog。例如，一个表的权限定名是hive.test\_data.test，则test是表名，test\_data是schema，hive是catelog。

Catelog的定义文件是在Presto的配置目录中。

（3）Schema

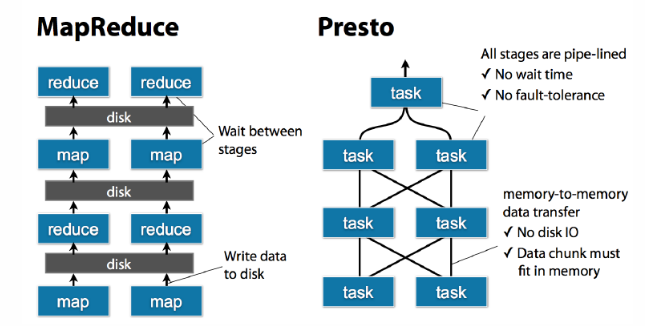
Schema是用于组织table。把catelog好schema结合在一起来包含一组的表。当通过Presto访问hive或Mysq时，一个schema会同时转为hive和mysql的同等概念。

（4）Table

Table跟关系型的表定义一样，但数据和表的映射是交给Connector。

### 1.3 Presto优缺点

Presto中SQL运行过程：MapReduce vs Presto



使用内存计算，减少与硬盘交互。

1）优点

（1）Presto与Hive对比，都能够处理PB级别的海量数据分析，但Presto是基于内存运算，减少了没必要的硬盘IO，所以更快。

（2）能够连接多个数据源，跨数据源连表查，如从Hive查询大量网站访问记录，然后从Mysql中匹配出设备信息。

（3）Presto安装也比Hive简单，因为Hive是基于HDFS的，所以需要先安装HDFS。

2）缺点

虽然能够处理PB级别的海量数据分析，但不是代表Presto把PB级别都放在内存中计算的。而是根据场景，如Count，AVG等聚合运算，是边读数据边计算，再清内存，再读数据再计算，这种耗的内存并不高。但是连表查，就可能产生大量的临时数据，因此速度会变慢，反而Hive此时会更擅长。

### 1.4 Presto、Impala性能比较

<https://blog.csdn.net/u012551524/article/details/79124532>

## 1.2 Presto安装

### 1.2.1 Presto Server安装

0）官网地址

https://prestodb.github.io/

1）下载地址

https://repo1.maven.org/maven2/com/facebook/presto/presto-server/0.196/presto-server-0.196.tar.gz

2）将presto-server-0.196.tar.gz导入hadoop102的/opt/software目录下，并解压到/opt/module目录

[atguigu@hadoop102 software]$ tar -zxvf presto-server-0.196.tar.gz -C /opt/module/

3）修改名称为presto

[atguigu@hadoop102 module]$ mv presto-server-0.196/ presto

4）进入到/opt/module/presto目录，并创建存储数据文件夹

[atguigu@hadoop102 presto]$ mkdir data

5）进入到/opt/module/presto目录，并创建存储配置文件文件夹

[atguigu@hadoop102 presto]$ mkdir etc

6）配置在/opt/module/presto/etc目录下添加jvm.config配置文件

[atguigu@hadoop102 etc]$ vim jvm.config

添加如下内容

-server

-Xmx16G

-XX:+UseG1GC

-XX:G1HeapRegionSize=32M

-XX:+UseGCOverheadLimit

-XX:+ExplicitGCInvokesConcurrent

-XX:+HeapDumpOnOutOfMemoryError

-XX:+ExitOnOutOfMemoryError

7）Presto可以支持多个数据源，在Presto里面叫catalog，这里我们配置支持Hive的数据源，配置一个Hive的catalog

[atguigu@hadoop102 etc]$ mkdir catalog

[atguigu@hadoop102 catalog]$ vim hive.properties

添加如下内容

connector.name=hive-hadoop2

hive.metastore.uri=thrift://hadoop102:9083

8）将hadoop102上的presto分发到hadoop103、hadoop104

[atguigu@hadoop102 module]$ xsync presto

9）分发之后，分别进入hadoop102、hadoop103、hadoop104三台主机的/opt/module/presto/etc的路径。配置node属性，node id每个节点都不一样。

**[atguigu@hadoop102 etc]$****vim node.properties**

node.environment=production

node.id=ffffffff-ffff-ffff-ffff-ffffffffffff

node.data-dir=/opt/module/presto/data

**[atguigu@hadoop103 etc]$vim node.properties**

node.environment=production

node.id=ffffffff-ffff-ffff-ffff-fffffffffffe

node.data-dir=/opt/module/presto/data

**[atguigu@hadoop104 etc]$vim node.properties**

node.environment=production

node.id=ffffffff-ffff-ffff-ffff-fffffffffffd

node.data-dir=/opt/module/presto/data

10）Presto是由一个coordinator节点和多个worker节点组成。在hadoop102上配置成coordinator，在hadoop103、hadoop104上配置为worker。

（1）hadoop102上配置coordinator节点

[atguigu@hadoop102 etc]$ vim config.properties

添加内容如下

coordinator=true

node-scheduler.include-coordinator=false

http-server.http.port=8881

query.max-memory=50GB

discovery-server.enabled=true

discovery.uri=http://hadoop102:8881

（2）hadoop103、hadoop104上配置worker节点

[atguigu@hadoop103 etc]$ vim config.properties

添加内容如下

coordinator=false

http-server.http.port=8881

query.max-memory=50GB

discovery.uri=http://hadoop102:8881

[atguigu@hadoop104 etc]$ vim config.properties

添加内容如下

coordinator=false

http-server.http.port=8881

query.max-memory=50GB

discovery.uri=http://hadoop102:8881

11）在/opt/module/hive目录下，启动Hive Metastore，用atguigu角色

nohup bin/hive --service metastore >/dev/null 2>&1 &

12）分别在hadoop102、hadoop103、hadoop104上启动Presto Server

（1）前台启动Presto，控制台显示日志

[atguigu@hadoop102 presto]$ bin/launcher run

[atguigu@hadoop103 presto]$ bin/launcher run

[atguigu@hadoop104 presto]$ bin/launcher run

（2）后台启动Presto

[atguigu@hadoop102 presto]$ bin/launcher start

[atguigu@hadoop103 presto]$ bin/launcher start

[atguigu@hadoop104 presto]$ bin/launcher start

13）日志查看路径/opt/module/presto/data/var/log

### 1.2.2 Presto命令行Client安装

1）下载Presto的客户端

<https://repo1.maven.org/maven2/com/facebook/presto/presto-cli/0.196/presto-cli-0.196-executable.jar>

2）将presto-cli-0.196-executable.jar上传到hadoop102的/opt/module/presto文件夹下

3）修改文件名称

[atguigu@hadoop102 presto]$ mv presto-cli-0.196-executable.jar prestocli

4）增加执行权限

[atguigu@hadoop102 presto]$ chmod +x prestocli

5）启动prestocli

[atguigu@hadoop102 presto]$ ./prestocli --server hadoop102:8881 --catalog hive --schema default

6）Presto命令行操作

Presto的命令行操作，相当于hive命令行操作。每个表必须要加上schema。

例如：

select \* from schema.table limit 100

### 1.2.3 Presto可视化Client安装

1）将yanagishima-18.0.zip上传到hadoop102的/opt/module目录

2）解压缩yanagishima

[atguigu@hadoop102 module]$ unzip yanagishima-18.0.zip

cd yanagishima-18.0

3）进入到/opt/module/yanagishima-18.0/conf文件夹，编写yanagishima.properties配置

[atguigu@hadoop102 conf]$ vim yanagishima.properties

添加如下内容

jetty.port=7080

presto.datasources=atiguigu-presto

presto.coordinator.server.atiguigu-presto=http://hadoop102:8881

catalog.atiguigu-presto=hive

schema.atiguigu-presto=default

sql.query.engines=presto

4）在/opt/module/yanagishima-18.0路径下启动yanagishima

[atguigu@hadoop102 yanagishima-18.0]$

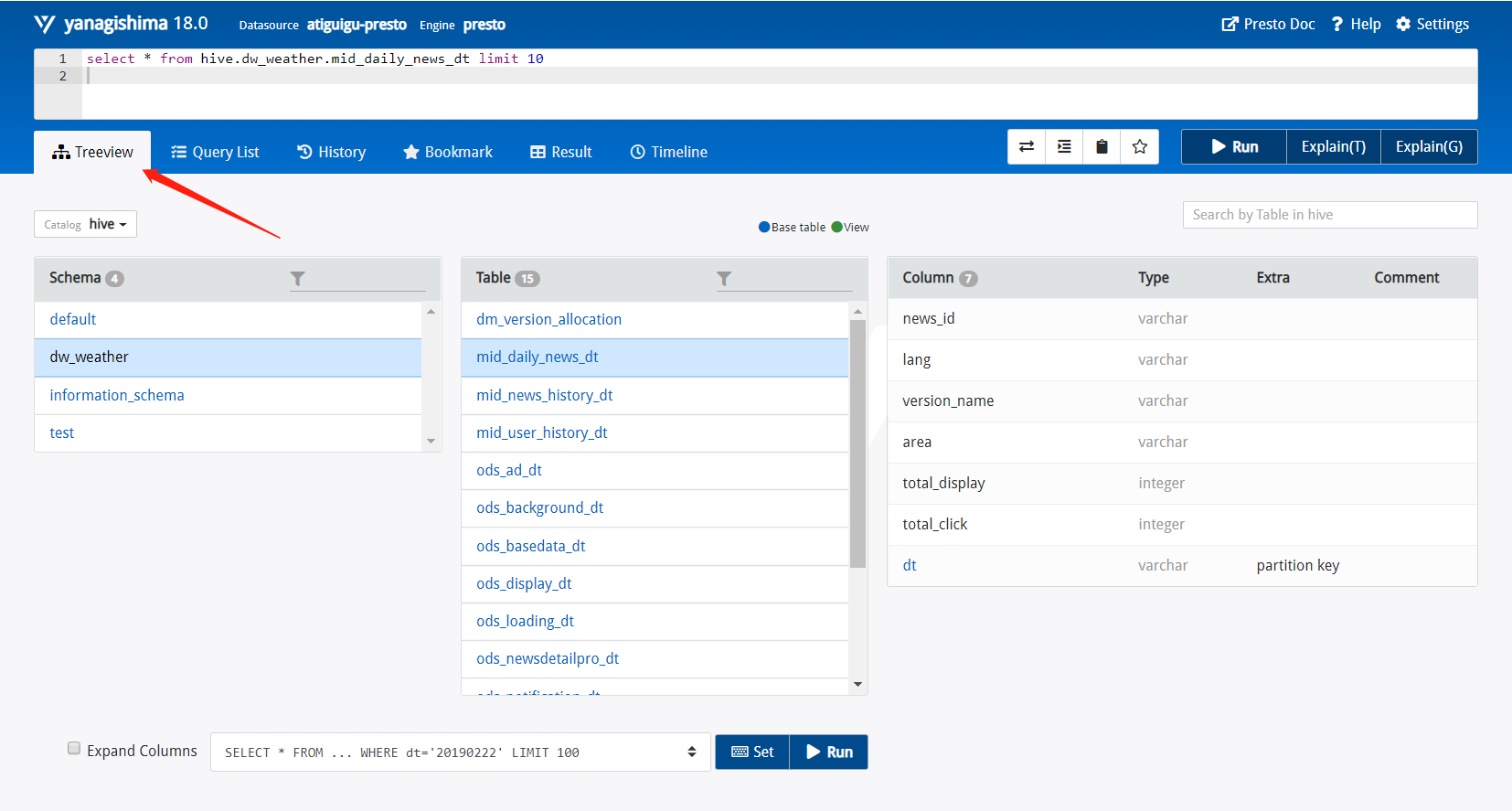
nohup bin/yanagishima-start.sh >y.log 2>&1 &

5）启动web页面

http://hadoop102:7080

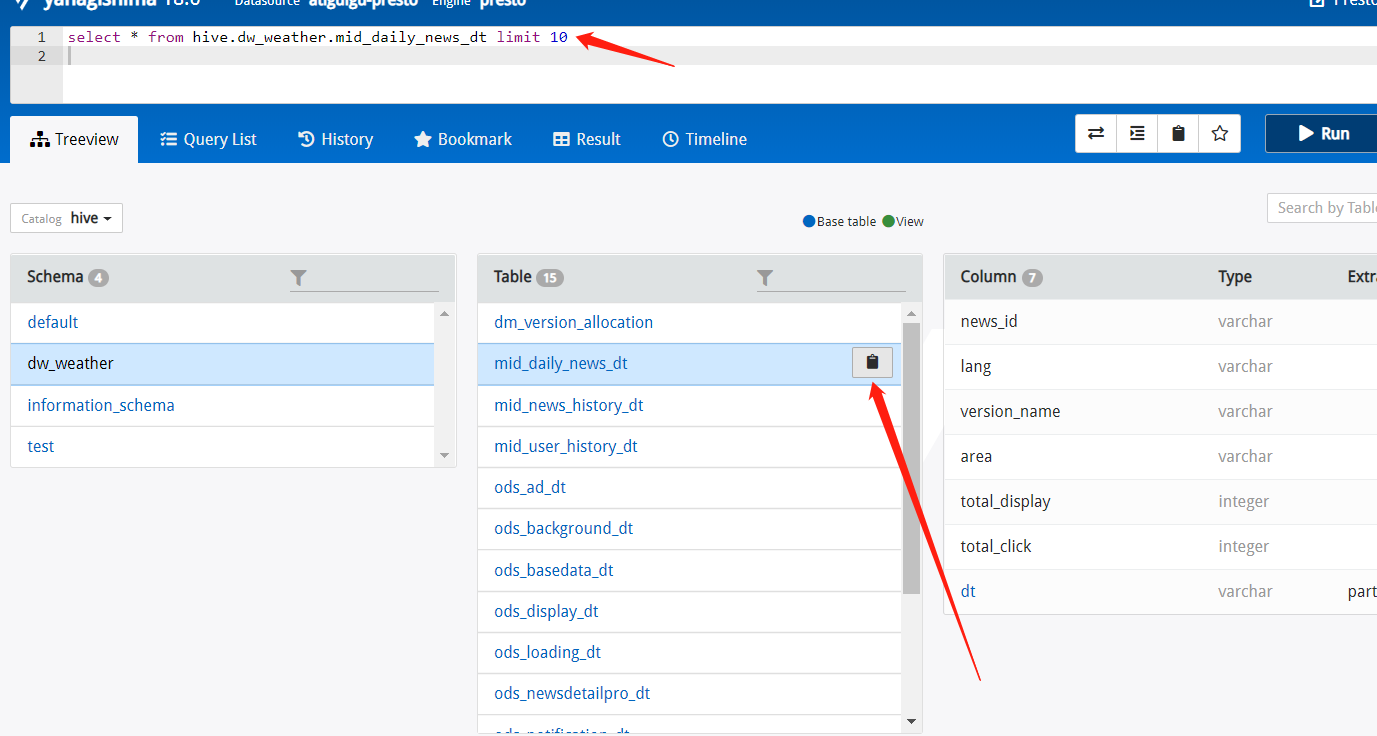
看到界面，进行查询了。

6）查看表结构

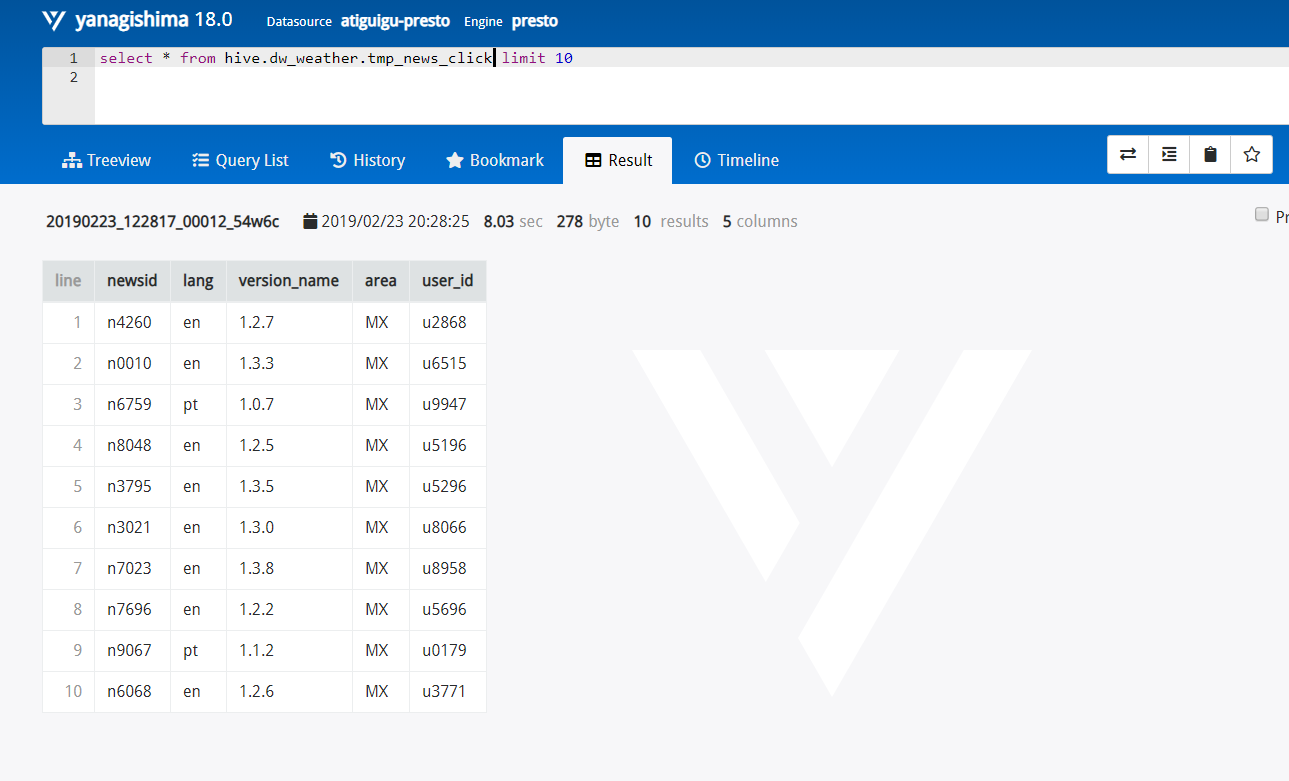


这里有个tree view，可以查看所有表的结构，包括schema、表、字段等。

比如执行select \* from hive.dw\_weather.tmp\_news\_click limit 10，这个句子里hive这个词可以删掉，是上面配置的catalog



每个表后面都有个复制键，点一下会复制完整的表名，然后再上面框里面输入sql语句，ctrl+enter键执行显示结果



## 1.3 Presto优化之数据存储

### 1.3.1 合理设置分区

与Hive类似，Presto会根据元数据信息读取分区数据，合理的分区能减少Presto数据读取量，提升查询性能。

### 1.3.2 使用列式存储

Presto对ORC文件读取做了特定优化，因此在Hive中创建Presto使用的表时，建议采用ORC格式存储。相对于Parquet，Presto对ORC支持更好。

### 1.3.3 使用压缩

数据压缩可以减少节点间数据传输对IO带宽压力，对于即席查询需要快速解压，建议采用Snappy压缩。

## 1.4 Presto优化之查询SQL

### 1.4.1 只选择使用必要的字段

由于采用列式存储，选择需要的字段可加快字段的读取、减少数据量。避免采用\*读取所有字段。

[GOOD]: SELECT time, user, host FROM tbl

[BAD]: SELECT \* FROM tbl

### 1.4.2 过滤条件必须加上分区字段

对于有分区的表，where语句中优先使用分区字段进行过滤。acct\_day是分区字段，visit\_time是具体访问时间。

[GOOD]: SELECT time, user, host FROM tbl where acct\_day=20171101

[BAD]: SELECT \* FROM tbl where visit\_time=20171101

### 1.4.3 Group By语句优化

合理安排Group by语句中字段顺序对性能有一定提升。将Group By语句中字段按照每个字段distinct数据多少进行降序排列。

[GOOD]: SELECT GROUP BY uid, gender

[BAD]: SELECT GROUP BY gender, uid

### 1.4.4 Order by时使用Limit

Order by需要扫描数据到单个worker节点进行排序，导致单个worker需要大量内存。如果是查询Top N或者Bottom N，使用limit可减少排序计算和内存压力。

[GOOD]: SELECT \* FROM tbl ORDER BY time LIMIT 100

[BAD]: SELECT \* FROM tbl ORDER BY time

### 1.4.5 使用Join语句时将大表放在左边

Presto中join的默认算法是broadcast join，即将join左边的表分割到多个worker，然后将join右边的表数据整个复制一份发送到每个worker进行计算。如果右边的表数据量太大，则可能会报内存溢出错误。

[GOOD] SELECT ... FROM large\_table l join small\_table s on l.id = s.id

[BAD] SELECT ... FROM small\_table s join large\_table l on l.id = s.id

## 1.5 注意事项

### 1.5.1 字段名引用

避免和关键字冲突：MySQL对字段加反引号**`、**Presto对字段加双引号分割

当然，如果字段名称不是关键字，可以不加这个双引号。

### 1.5.2 时间函数

对于timestamp，需要进行比较的时候，需要添加timestamp关键字，而MySQL中对timestamp可以直接进行比较。

/\*MySQL的写法\*/

SELECT t FROM a WHERE t > '2017-01-01 00:00:00';

/\*Presto中的写法\*/

SELECT t FROM a WHERE t > timestamp '2017-01-01 00:00:00';

### 1.5.3 不支持INSERT OVERWRITE语法

Presto中不支持insert overwrite语法，只能先delete，然后insert into。

### 1.5.4 PARQUET格式

Presto目前支持Parquet格式，支持查询，但不支持insert。

# 第2章 Druid

## 2.1 Druid简介

### 2.1.1 什么是Druid

Druid是一个快速的列式分布式的支持实时分析的数据存储系统。它在处理PB级数据、毫秒级查询、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。

Druid的官方网站是[http://druid.io](http://druid.io/)。

注意：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和本文讨论的Druid没有任何关系，它们解决完全不同的问题。

### 2.1.2 Druid特点

**1）列式存储格式**。Druid使用面向列的存储，它只需要加载特定查询所需要的列。查询速度迅速快。

**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）数据进行了有效的**预聚合或预计算**，查询速度快。

7）数据的结果应用了**Bitmap压缩算法**。

### 2.1.3 Druid使用场景

1）适用于清洗好的记录实时录入，但不需要更新操作

2）支持宽表，不用join的方式（换句话说就是一张单表）

3）可以总结出基础的统计指标，可以用一个字段表示

4）对时区和时间维度（year、month、week、day、hour等）要求高的（甚至到分钟级别）

5）实时性很重要

6）对数据质量的敏感度不高

7）用于定位效果分析和策略决策参考

### 2.1.4 Druid对比Presto/Spark SQL/Kylin/Elasticsearch

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 对比项目 | Druid | Kylin | Presto | Spark SQL | Elasticsearch |
| 亚秒级响应 | Y | Y | N | N | N |
| 百亿数据集 | Y | Y | Y | Y | Y |
| SQL支持 | N（开发中） | Y | Y | Y | N |
| 离线 | Y | Y | Y | Y | Y |
| 实时 | Y | N（开发中） | N | N | Y |
| 精确去重 | N | Y | Y | Y | N |
| 多表Join | N | Y | Y | Y | N |
| JDBC for BI | N | Y | Y | Y | N |

**1）Presto：**facebook开源的一个Java写的分布式数据查询框架，原生集成了Hive、Hbase和关系型数据库，Presto背后所使用的执行模式与Hive有根本的不同，它没有使用MapReduce，大部分场景下比Hive快一个数量级，其中的关键是所有的处理都在内存中完成。

**2）Druid：**是一个实时处理时序数据的OLAP数据库，因为它的索引首先按照时间分片，查询的时候也是按照时间线去路由索引。

**3）Spark SQL：**基于Spark平台上的一个OLAP框架，基本思路是增加机器来并行计算，从而提高查询速度。

**4）Kylin：**核心是Cube，Cube是一种预计算技术，基本思路是预先对数据作多维索引，查询时只扫描索引而不访问原始数据从而提速。

**5）ES：**最大的特点是使用了倒排索引解决索引问题。根据研究，ES在数据获取和聚集用的资源比在Druid高。

**6）框架选型：**

很多企业混合使用上述查询引擎，分别服务于不同的场景。

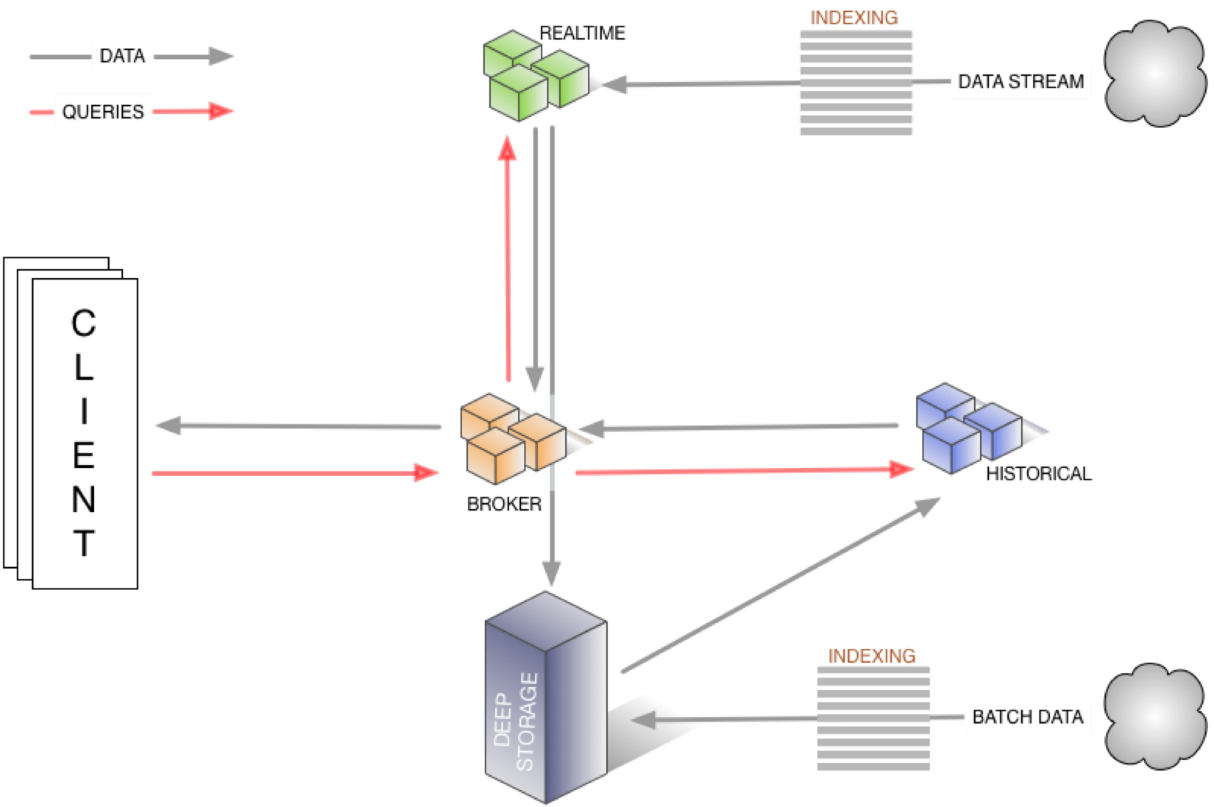
从超大数据的查询效率来看：Druid > Kylin > Presto > Spark SQL

从支持的数据源种类来讲：Presto > Spark SQL > Kylin > Druid

## 2.2 Druid框架原理

### 2.2.1 Druid集群介绍

关于Druid的架构，我们先通过其总体设计架构图来做一个概要了解，如下图所示：



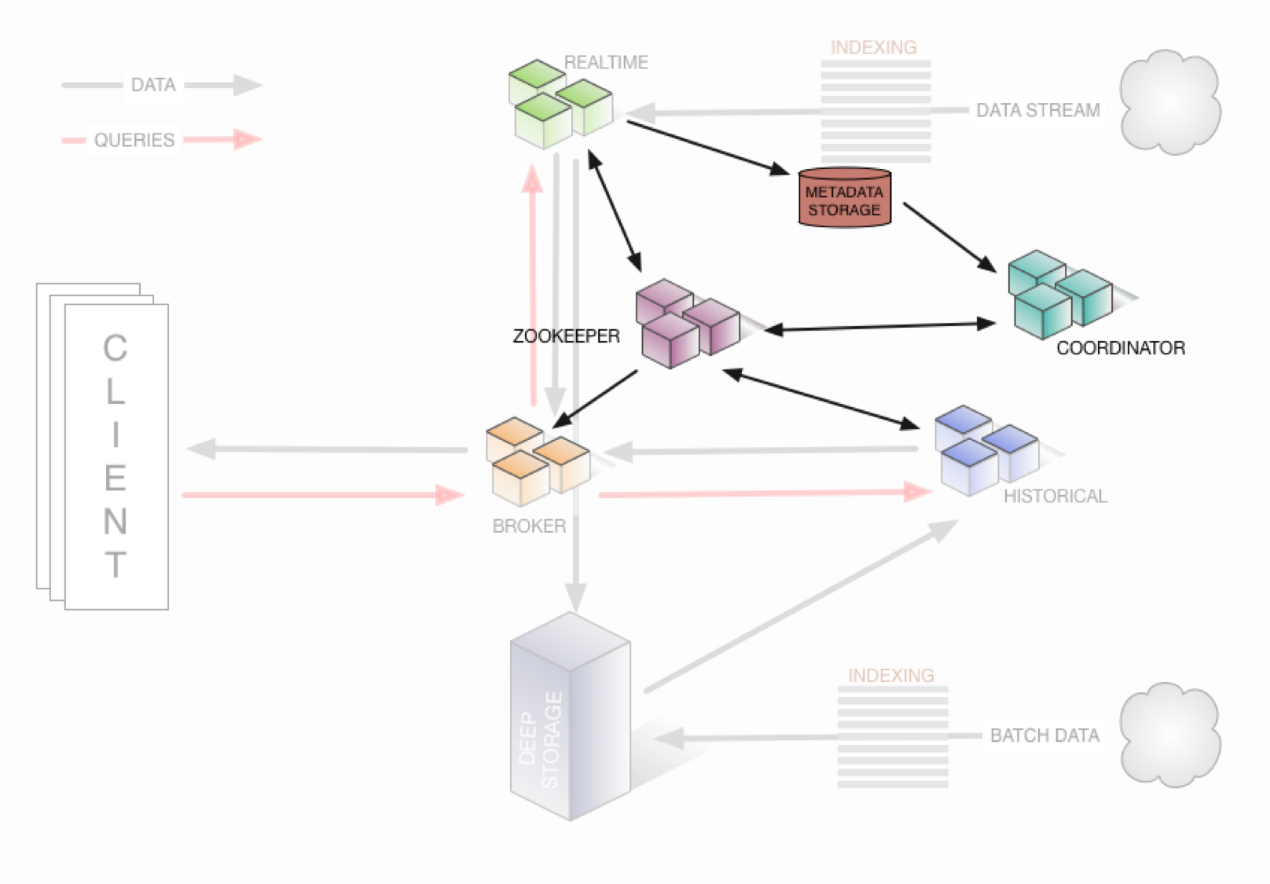
1）实时节点(RealTime Node)：即时摄入数据，以及生成Segment数据文件。

2）历史节点(Historical Node)：加载已经生成好的数据文件，以供数据查询。

3）查询节点(Broker Node)：对外提供数据查询服务，并同时从实时节点与历史节点查询数据，合并后返回给调用方。

4）协调节点(Coordinator Node)：负责历史节点数据负载均衡，以及通过规则(Rule)来管理数据的生命周期。

集群的外部依赖如下图所示：



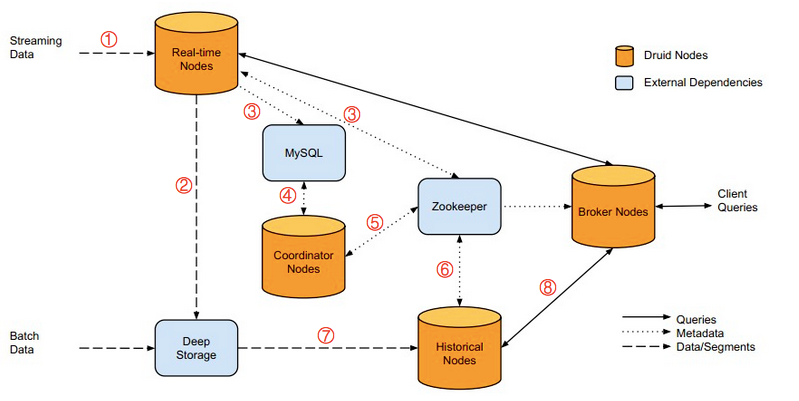
1）元数据库(Metastore)：存储Druid集群的元数据信息，比如Segment的相关信息，一般用MySQL或者PostgreSQL。

2）分布式协调服务：为Druid集群提供一致性协调服务组件，通常为Zookeeper。

3）数据文件存储(DeepStorage)：存放生成的Segment数据文件，并提供历史节点下载。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或NFS。

### 2.2.2 Druid实时节点

实时节点主要负责即时摄入数据，以及生成Segment文件，其独到的设计使其拥有超强的数据摄入速度。



Segment数据文件从生成到传播需要经历一个完整的流程，步骤如下：

1）实时节点生产出Segment数据文件，并将其上传到DeepStorage中。

2）Segment数据文件的相关元数据信息被存储到MySQL中。实时节点转存的Segment会在ZooKeeper中新增一条记录

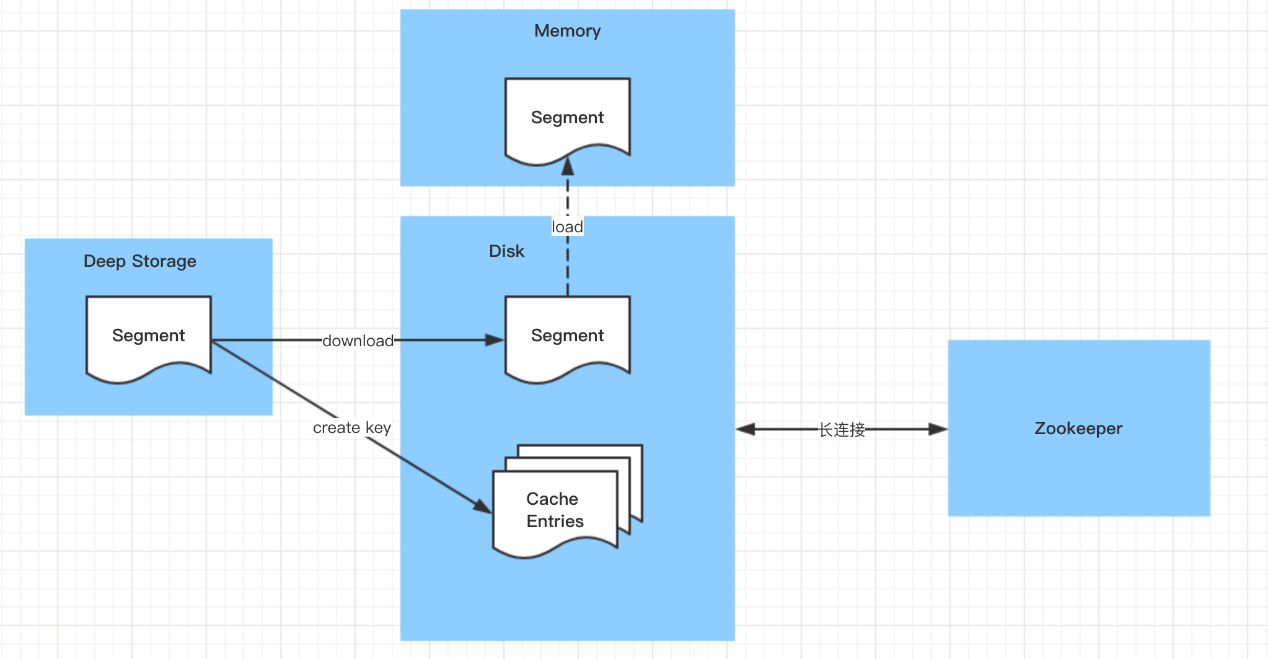
3）Master节点（即Coordinator几点）从MetaStore里得知Segment数据文件的相关元数据信息后，将其根据规则的设置分配符合条件的历史节点。

4）历史节点得到指令会主动从DeepStorage中拉取指定的Segment数据文件，并通过Zookeeper向集群申明其负责提供该Segment数据文件的查询服务。

5）实时节点丢弃该Segment数据文件，并向集群申明其不再提供该Segment数据文件的查询服务。

### 2.2.3 Druid历史节点

历史节点用于负责加载已经生成好的数据文件以提供数据查询。由于Druid的数据文件有不可更改性，因此历史节点的工作就是专注于提供数据查询。



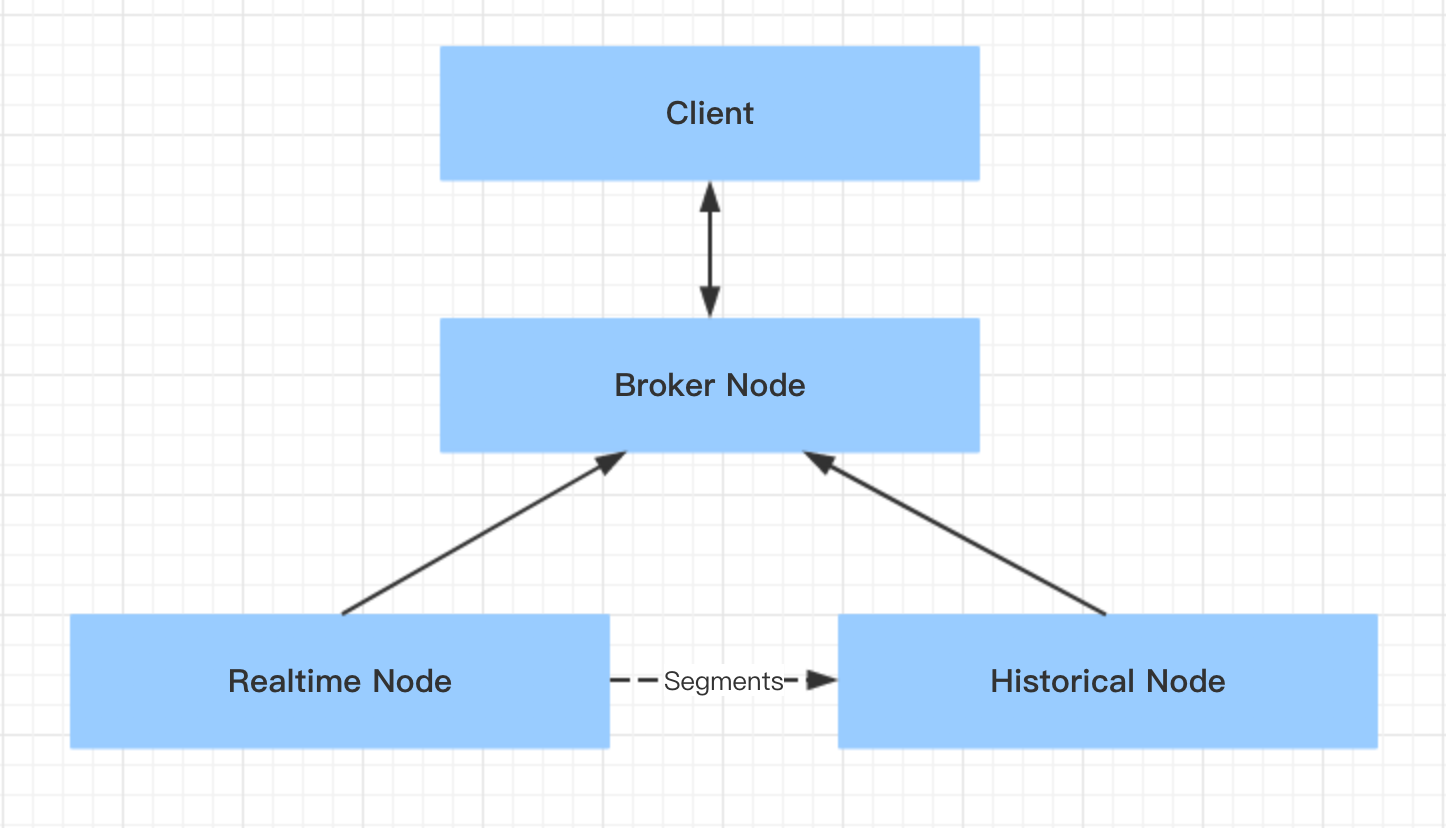
1）Coordinator在ZK与History节点相关联的加载队列路径下创建一个临时记录。

2）当历史节点发现在Zookeeper中有需要加载的新的记录。它首先检查本地磁盘目录（缓存）中关于新的Segment的信息。如果缓存中没有关于新的Segment的信息，历史节点将下载新的Segment的元数据信息并告知Zookeeper。元数据包含新的Segment在“Deep Storage”中的存储位置，怎样去解压缩和处理新的Segment的信息。

3）一旦历史节点处理完一个Segment，就公布该Segment可查询。

### 2.2.4 Druid查询节点

查询节点对外提供数据查询服务，并同时从实时节点与历史节点查询数据，合并后返回给调用方。Zookeeper维护有关历史和实时的节点信息和它们所能提供服务的Segment。当收某个数据源和时间的查询，代理节点执行查找与查询数据源时间相关的时间轴和检索包含数据查询的节点。代理节点将查询转发到所选节点。

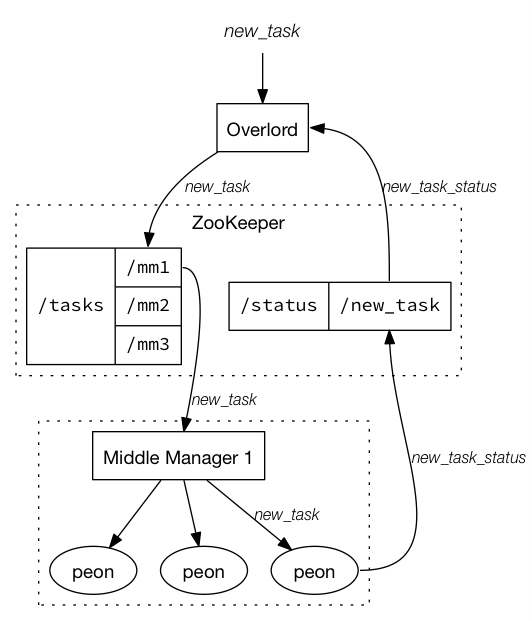


### 2.2.5 Druid协调节点

协调节点负责历史节点的数据负载均衡，以及通过规则管理数据的生命周期。Druid针对每个DataSource设置规则来加载或者丢弃具体的数据文件，以管理数据生命周期。可以对一个DataSource按照顺序添加多条规则，对于一个Segment数据文件来说，协调节点会逐条检查规则，当碰到当前Segment数据文件符合某条规则的时候，协调节点会立即命令历史节点堆该Segment数据文件执行这条规则——加载或者丢弃，并停止检查余下的规则，否则继续检查下一条规则。

### 2.2.6 Druid索引服务

除了通过实时节点产生出Segment数据文件外，Druid还提供一组名为索引服务的组件。不同于实时节点的单点模式，索引服务实际包含一组组件，并以主从结构作为其架构方式，其中统治节点（Overlord Node）为主节点，而中间管理者（Middle Manager）为从节点。索引节点服务架构示意图如下所示：

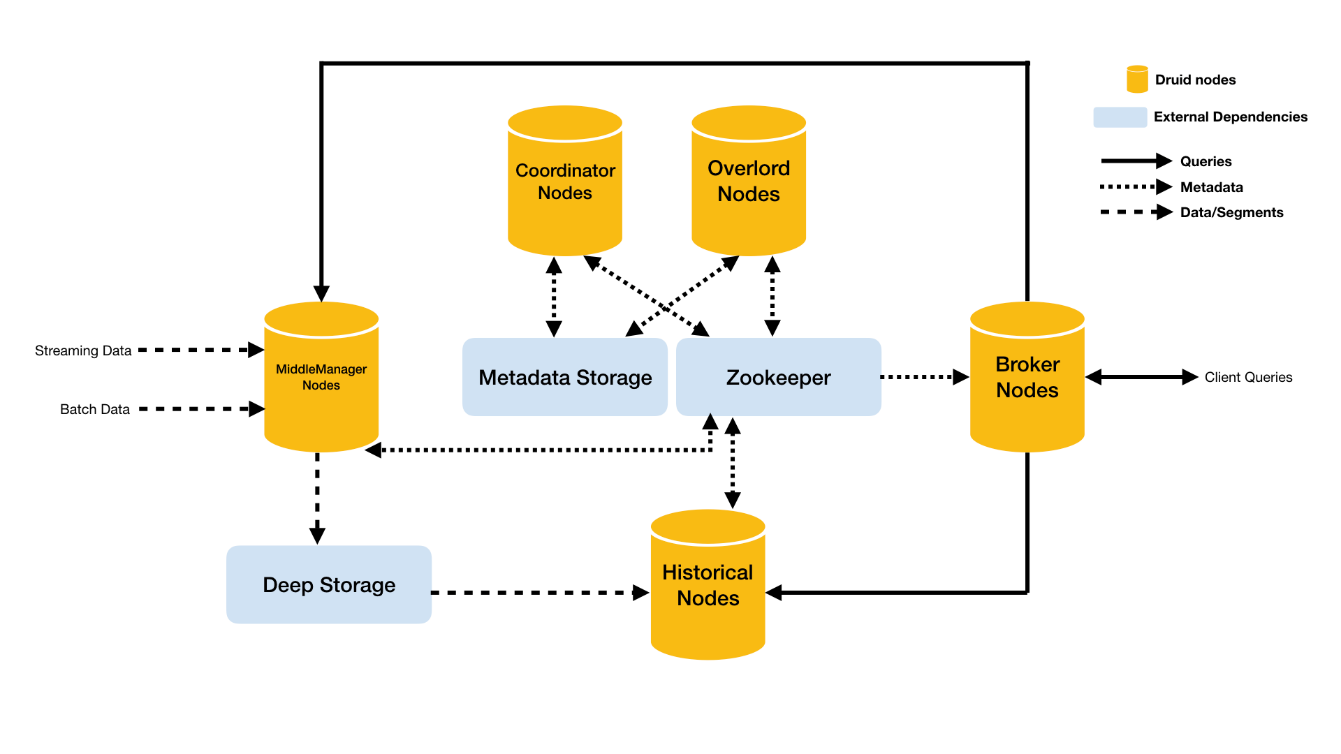


索引服务是主从结构，由三个部分组成：

1）peon组件：在一个单独的jvm中运行单个任务，通过单独的jvm对任务做资源隔离和日志隔离。

2）Middle Manager：用于创建和管理peon的中层管理组件

3）overlord组件：管理任务分配到Middle Manager



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：

6）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。

7）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。

8）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。

## 2.3 Druid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。

### 2.3.1 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。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=&url=http://album.sina.com.cn/pic/003t0w9pzy7lV2GwhUg04)﻿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。

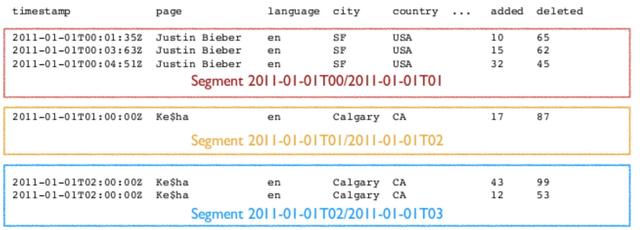
[](http://photo.blog.sina.com.cn/showpic.html#blogid=&url=http://album.sina.com.cn/pic/003t0w9pzy7lV2Rsevibc)﻿DataSource聚合后的数

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。

### 2.3.2 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。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=&url=http://album.sina.com.cn/pic/003t0w9pzy7lV2Zux2565)﻿ 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。

## 2.4 Druid安装（单机版）

### 2.4.1 安装包下载

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

### 2.4.2 安装部署

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）将imply-2.7.10.tar.gz上传到hadoop102的/opt/software目录下，并解压

[atguigu@hadoop102 software]$ tar -zxvf imply-2.7.10.tar.gz -C /opt/module

2）修改/opt/module/imply-2.7.10名称为/opt/module/imply

[atguigu@hadoop102 module]$ mv imply-2.7.10/ imply

3）修改配置文件

（1）修改Druid的ZK配置

[atguigu@hadoop102 \_common]$ vi /opt/module/imply/conf/druid/\_common/common.runtime.properties

修改如下内容

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]$ vim

/opt/module/imply/conf/supervise/quickstart.conf

修改如下内容

:verify bin/verify-java

#:verify bin/verify-default-ports

#:verify bin/verify-version-check

:kill-timeout 10

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

4）启动

（1）启动Zookeeper

[atguigu@hadoop102 imply]$ zk.sh start

（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/imply/var/sv/查看服务启动时的日志信息

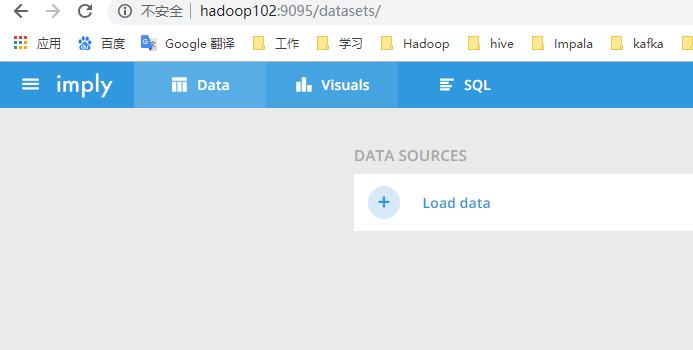
（3）启动采集Flume和Kafka（主要是为了节省内存开销，同时hadoop102内存调整为8G）

[atguigu@hadoop102 imply]$ f1.sh start

[atguigu@hadoop102 imply]$ kf.sh start

### 2.4.3 Web页面使用

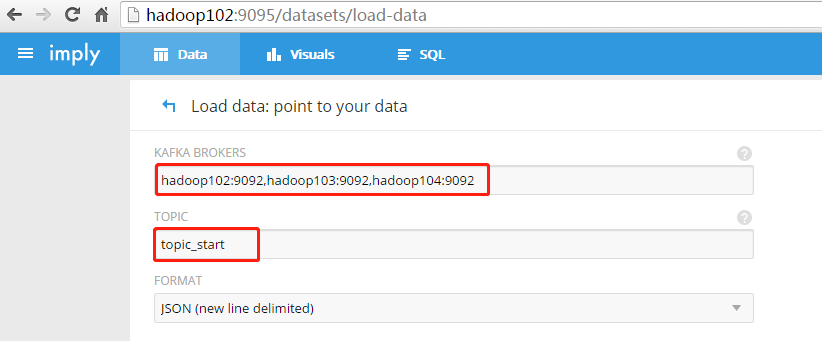
1）登录hadoop102:9095查看



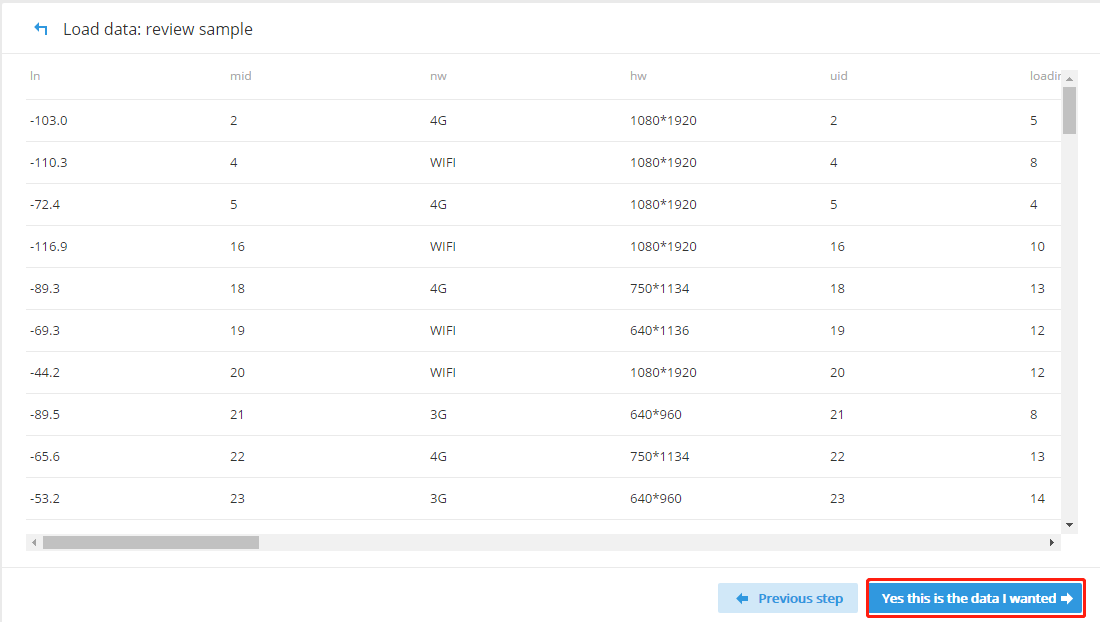
2）点击Load data->点击Apache Kafka



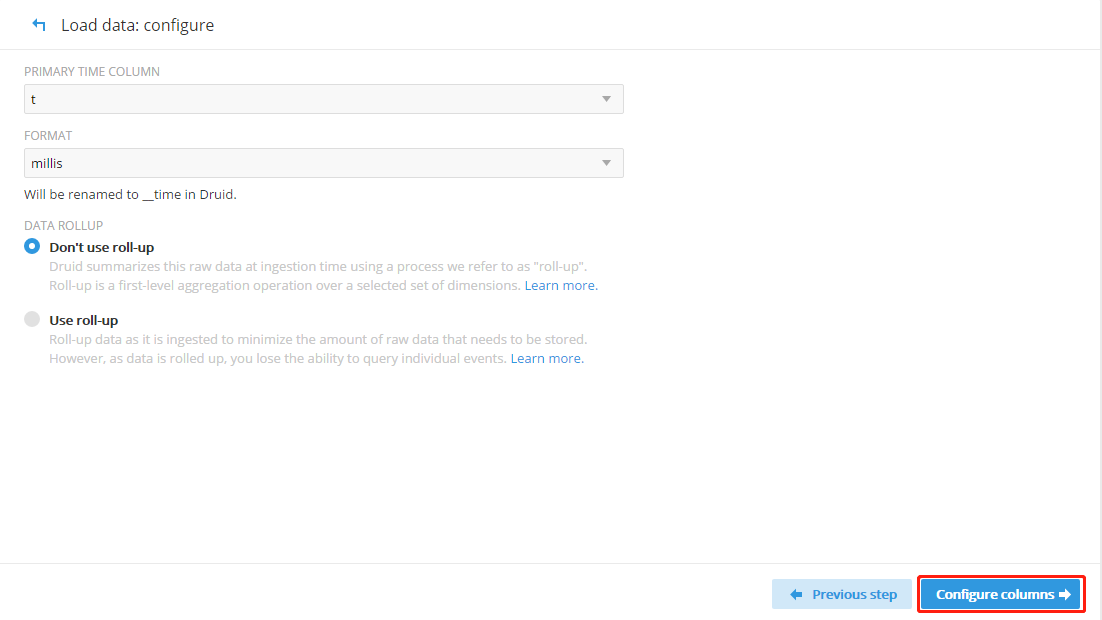
3）添加Kafka Broker和要消费的topic



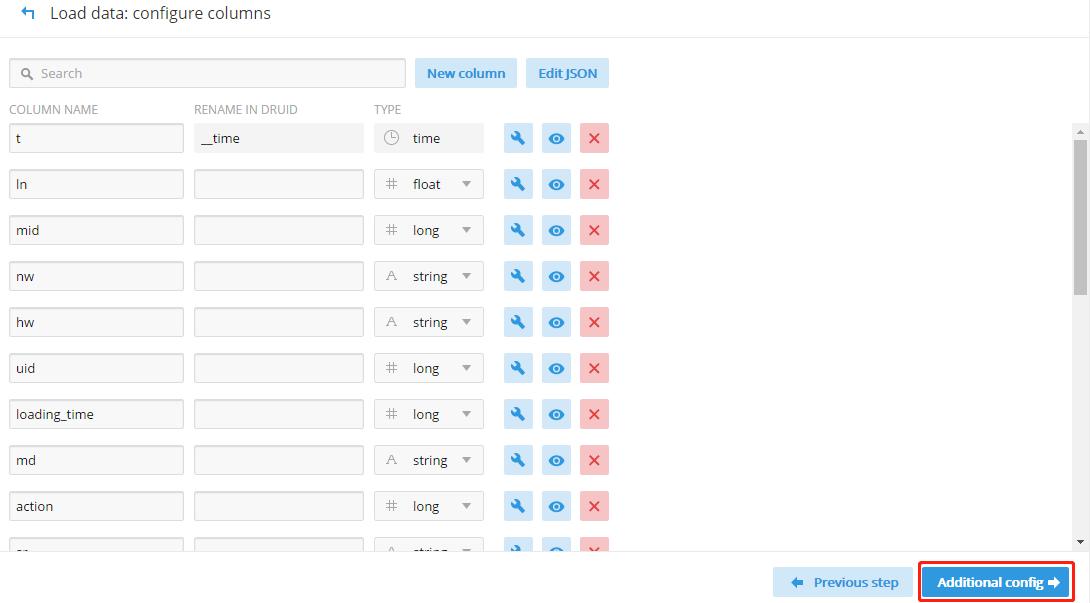
4）确认数据样本格式



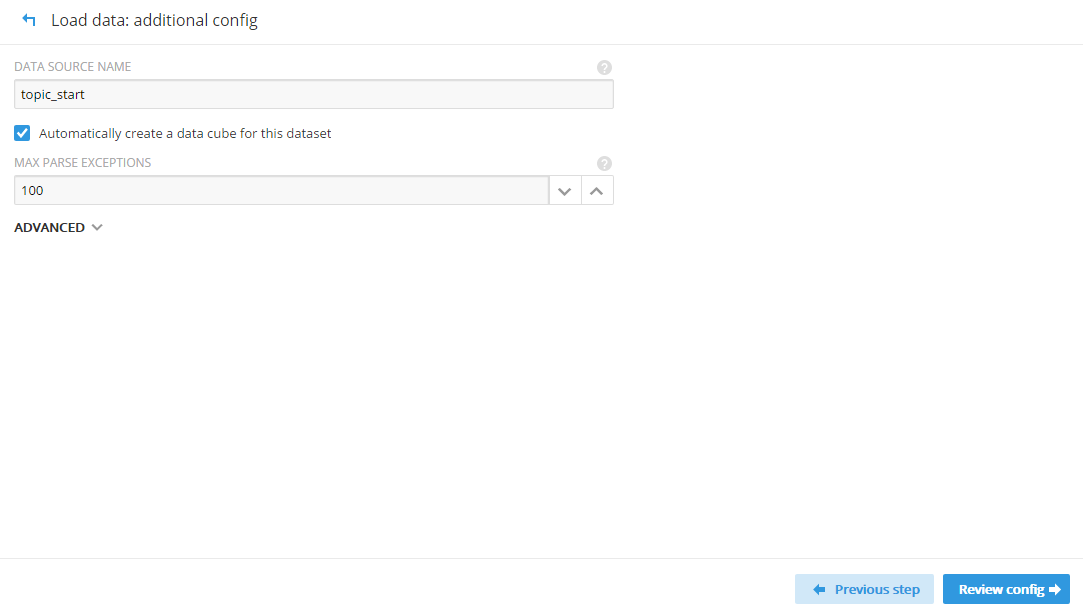
5）加载数据，必须要有时间字段



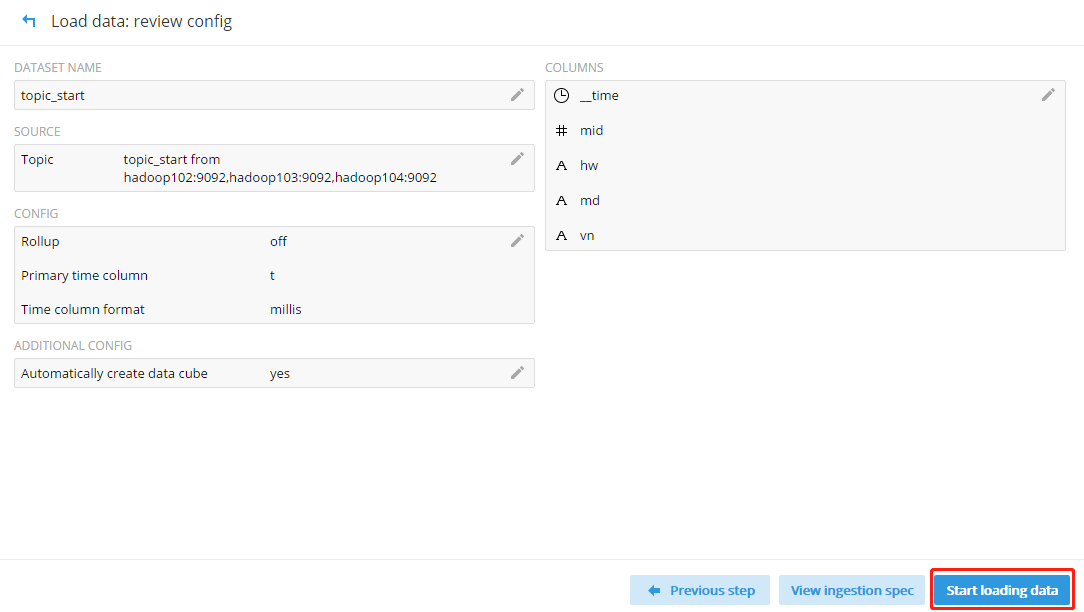
6）配置要加载哪些列



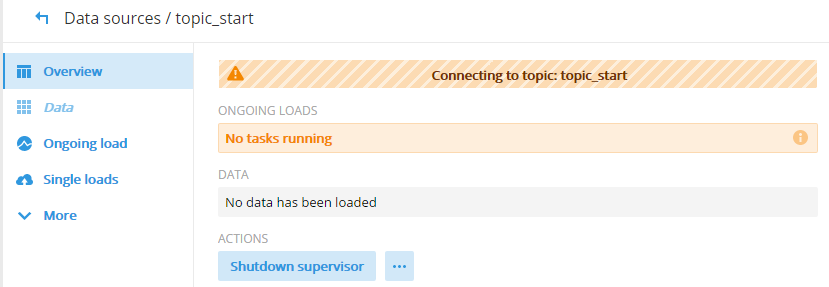
7）创建数据库表名

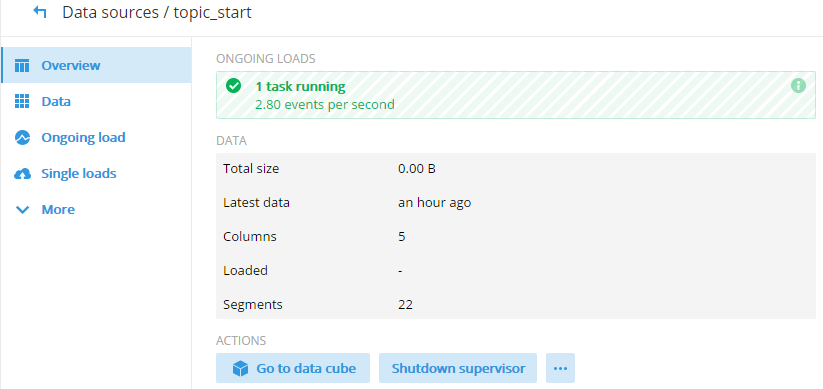


8）重新观察一下配置

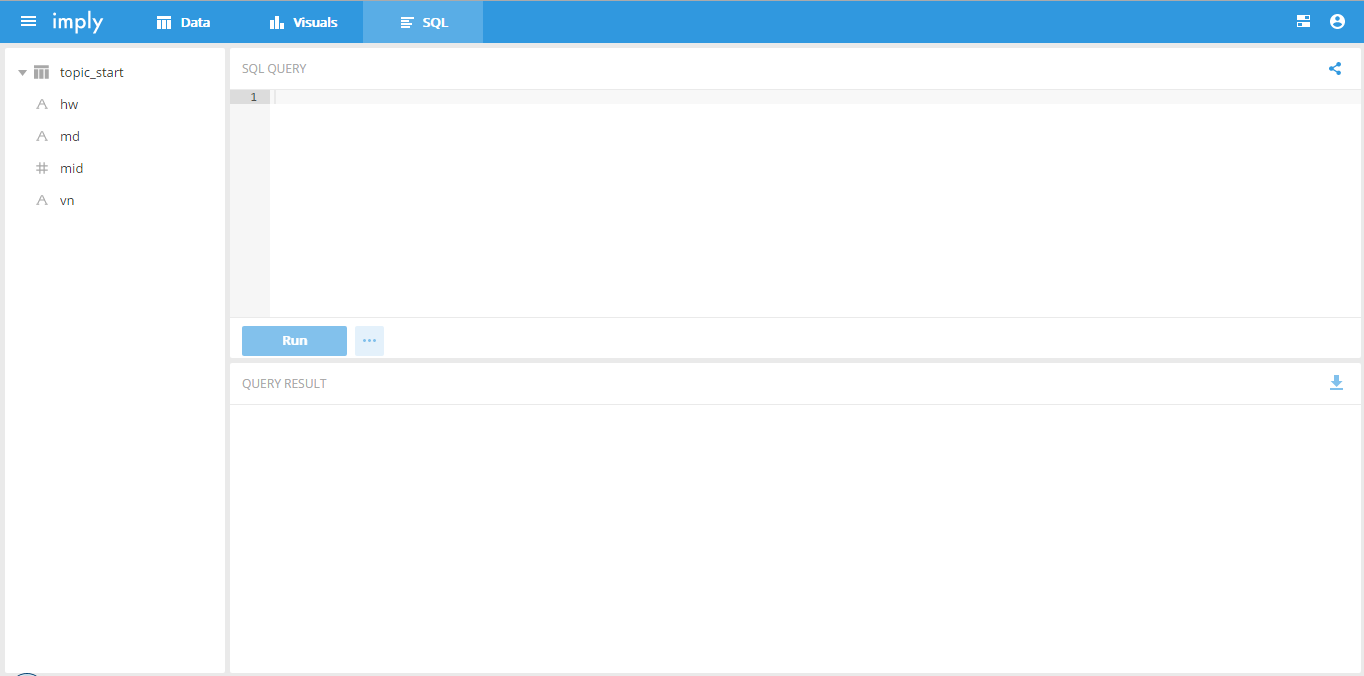


9）连接Kafka的topic\_start

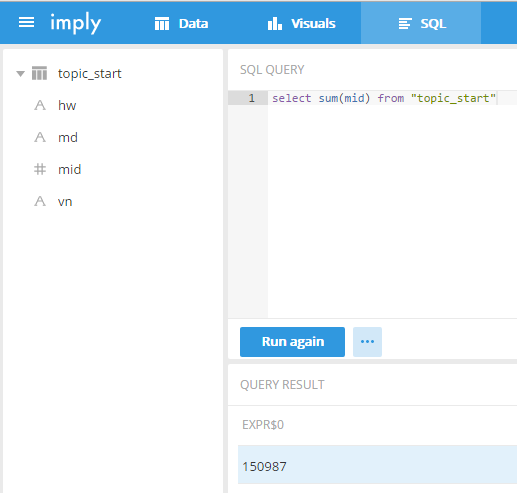




10）点击SQL



11）查询指标



### 2.4.4 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/var/目录。

# 第3章 Kylin

## 3.1 Hbase安装



## 3.2 Kylin安装



## 3.3 Kylin使用