

数据治理技术预研

数据目录，它是数据湖中所有[大数据的目录](#)。通过将元数据应用于数据湖中的所有内容，数据发现 and 治理变得更加容易。通过将元数据和分层逻辑应用于传入数据，数据集接收必须有的上下文和可跟踪的血缘关系，以便在工作流中有效使用。数据源像一条条小溪，最终将数据倾倒入数据湖中，如果缺乏数据目录的组织级别，最终数据湖会变成数据沼泽。

[Datahub VS Atlas VS Amundsen](#)

搜索	推荐	表描述	数据预览	列统计	占用指标	权限	排名	数据血统	改变通知	开源	文档	支持数据源	
Amundsen (Lyft)	✓	✓	✓	✓	✓		✓	✓	Todo	✓	✓	✓	Hive, Redshift, Druid, RDBMS, Presto, Snowflake, etc.
Datahub (LinkedIn)	✓		✓				✓	✓	✓	✓	✓	✓	Hive, Kafka, RDBMS
Metacat (Netflix)	✓		✓		✓	✓		Todo		Todo	✓		Hive, RDS, Teradata, Redshift, S3, Cassandra
Atlas (Apache)	✓		✓						✓	✓	✓	✓	HBase, Hive, Sqoop, Kafka, Storm
Marquez (Wework)	✓		✓						✓		✓		S3, Kafka
Databook (Uber)	✓		✓	✓	✓				✓				Hive, Vertica, MySQL, Postgress, Cassandra
Dataportal (Airbnb)	✓		✓		✓		✓	✓					Unknown
Data Access Layer (Twitter)	✓		✓						✓				HDFS, Vertica, MySQL
Lexikon (Spotify)	✓	✓	✓				✓	✓					Unknown

# DataHub

---

[官网入门](#)

<https://github.com/linkedin/datahub> ☆ 4.7k

DataHub 是一个现代数据目录，旨在支持端到端数据发现、数据可观察性和数据治理。这个可扩展的元数据平台是为开发人员构建的，让他们可以适应快速变革的数据生态系统的复杂性，并让数据从业者在其组织内利用数据的全部价值。

## 安装

python3

```
# ubuntu中已经预装python和python3
# 执行python3 -m pip 提示 pip模块找不到
sudo apt-get install python3-pip
```

docker-compose

参考:<https://blog.csdn.net/pushiqiang/article/details/78682323>

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.1/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

安装DataHub CLI

```
python3 -m pip install --upgrade pip wheel setuptools
python3 -m pip uninstall datahub acryl-datahub || true # sanity check - ok if
it fails
python3 -m pip install --upgrade acryl-datahub
datahub version # 或 python3 -m datahub version
```

执行quick-start

```
datahub docker quickstart
```

@TODO 启动失败

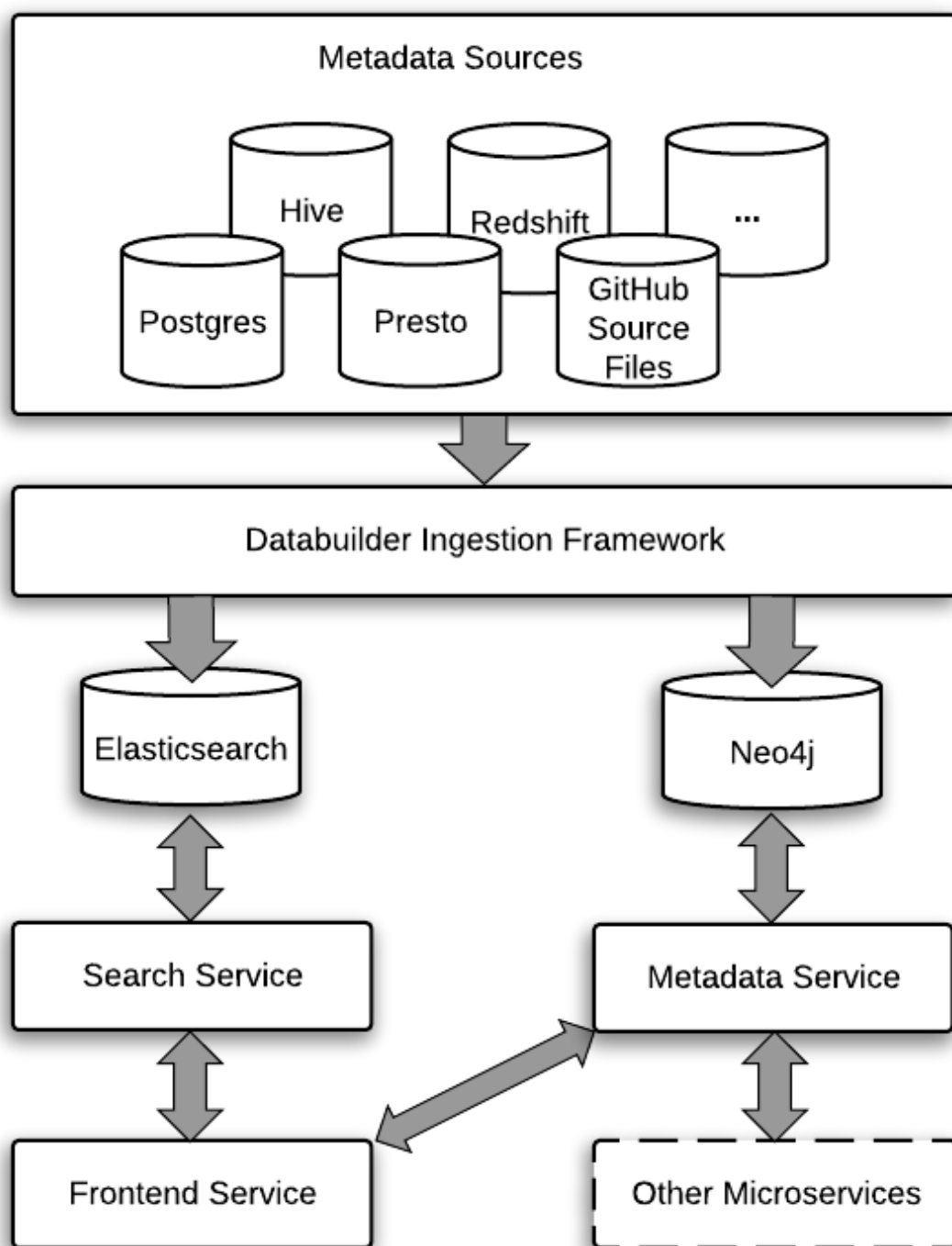
## Amundsen

---

<https://www.amundsen.io/amundsen/>

<https://github.com/amundsen-io/amundsen> ☆3k

Amundsen 是一个数据发现和元数据引擎，用于提高数据分析师、数据科学家和工程师在与数据交互时的工作效率。



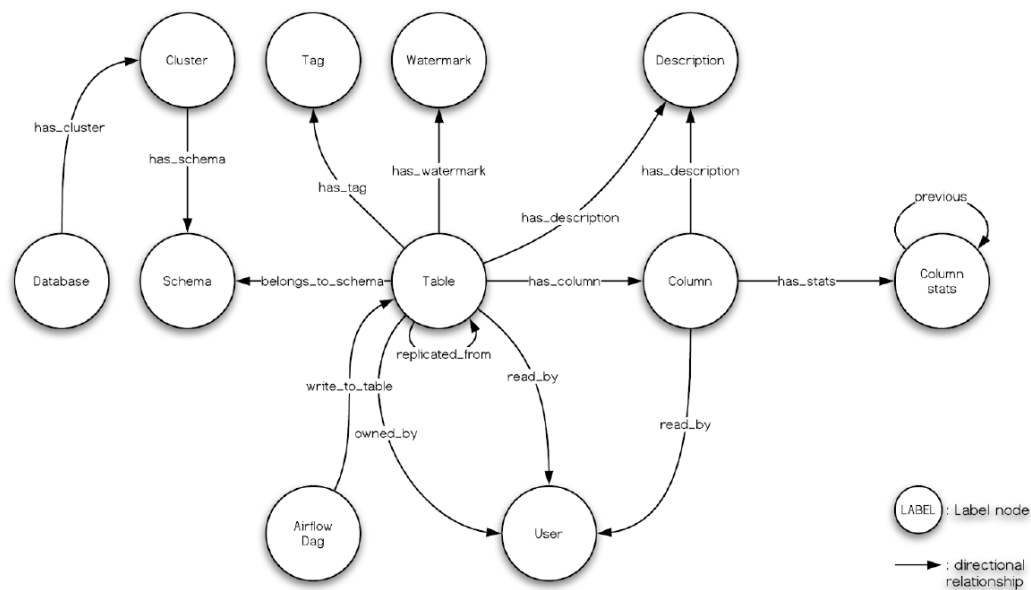
Amundsen的架构：

前端：基于Flask的web应用侧，展示层基于React with Redux, Bootstrap, Webpack, and Babel.

搜索：基于ElasticSearch并提供RESTful API进行搜索服务。搜索索引基于Databuilder elasticsearch publisher.

元数据：使用Neo4j代理与Neo4j图形数据库交互，提供元数据服务。Amundsen中的元数据模型如下图：

DataBuilder: 提供数据摄取库构建元数据。



Quick Start:

准备阶段，安装python>=3.7

```
# python > 3.7
sudo apt update
sudo apt-get install zlib1g-dev libbz2-dev libssl-dev libncurses5-dev libsqlite3-
dev libreadline-dev tk-dev libgdbm-dev libdb-dev libpcap-dev xz-utils libexpat1-
dev liblzma-dev libffi-dev libc6-dev
# download https://www.python.org/ftp/python/3.9.10/
sudo mkdir -p /usr/local/python3 #建立安装目录

##编译安装
#后面加上 --enable-optimizations 会自动安装pip3及优化配置
./configure --prefix=/usr/local/python3 --enable-optimizations
make
sudo make install

sudo rm -rf /usr/bin/python3
sudo rm -rf /usr/bin/pip3

#添加python3的软链接
sudo ln -s /usr/local/python3/bin/python3.9 /usr/bin/python3
#添加 pip3 的软链接
sudo ln -s /usr/local/python3/bin/pip3.9 /usr/bin/pip3

# 检查版本
python3 -V
pip3 -V
```

运行官方示例

```
$ git clone --recursive https://github.com/amundsen-io/amundsen.git
# For Neo4j Backend
$ docker-compose -f docker-amundsen.yml up
```

```
# For Atlas
$ docker-compose -f docker-amundsen-atlas.yml up

# 导入样本数据到Neo4j
cd databuilder

apt-get install python3-venv
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple/ --upgrade setuptools

python3 -m venv venv
source venv/bin/activate
pip3 install --upgrade pip
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple/ -r requirements.txt

# maybe you need upgrade setuptools
pip3.4 install --upgrade setuptools

python3 setup.py install
python3 example/scripts/sample_data_loader.py
```

最终通过在docker中运行成功：

```
docker run -it -v /home/anxin/amundsen-main:/data python:3.7.12-bullseye
/bin/bash
```

The screenshot shows the Amundsen web interface. The main content area displays details for a dataset named 'test\_schema.test\_table2'. On the left, there is a sidebar with metadata including 'Description' (2nd test table), 'Last Updated' (Dec 01, 2003 6am CST), 'Date Range' (From: Jan 01, 2018, To: Oct 01, 2019), 'Tags' (cheap, recommended), 'S3 Crawler', 'Size: 1T', and 'Monthly Cost: \$50'. The main panel shows a table of columns with 4 columns: NAME, TYPE, and two unlabeled columns. The columns are: col1 (string), col2 (string), col3 (string), and col4 (int). Below the column list, there is a 'Column Statistics' section showing data collected between Aug 15, 2018 and Oct 29, 2019. The statistics table has columns: average, \*400\*, median, and \*250\*.

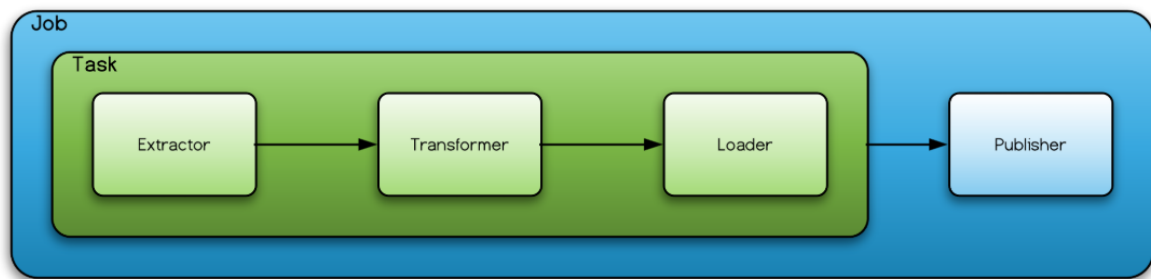
NAME	TYPE
col1 col1 description	string
col2 col2 description	string
col3 col3 description	string
col4 col4 description	int

Column Statistics			
Stats reflect data collected between Aug 15, 2018 and Oct 29, 2019.			
average	*400*	median	*250*

## Databuilder

客户端通过Job启动ETL任务，包括task和publisher。

其中task控制Extractor、Transformer、Loader组件，实现Amundsen的ETL流程。



## Extractor

An extractor extracts records from the source. This does not necessarily mean that it only supports [pull pattern](#) in ETL. For example, extracting records from messaging bus makes it a push pattern in ETL.

导出器，不仅仅包含拉取模式，例如消息总线中推送过来的消息。

内置了哪些导出器：

DBAPIExtractor 数据库导出

Generic 通用导出

HiveTableLastUpdated 导出最近更新的Hive metastore

HiveTableMetadata 导出Hive metadata

Cassandra 导出apache cassandra数据库metastore

Glue 导出 AWS Glue metastore

Delta-Lake-MetadataExtractor 运行在spark上面，使用Spark sql导出delta-lake 元数据

PostgresMetadataExtractor 导出Postgres或Redshift数据库的表列元数据，包括数据库、架构、表名、表描述、列名、列描述。

RestAPIExtractor 通过构造RESTAPI查询导出数据 [RestAPIQuery](#)。

ElasticsearchMetadataExtractor 导出ES索引的元数据

其他：Neo4j/Altas/BigQuery/Snowflake/Databricks/Superset

## Transformer

A transformer takes a record from either an extractor or from other transformers (via ChainedTransformer) to transform the record.

转换器，处理从Extractor或另一个Transformer中过来的消息。

## Loader

A loader takes a record from a transformer or from an extractor directly and loads it to a sink, or a staging area. As the loading operates at a record-level, it's not capable of supporting atomicity.

加载到Sink、或者一个输出区域\*(staging area).

## Task

A task orchestrates an extractor, a transformer, and a loader to perform a record-level operation.

Task调度ETL组件，执行记录级别操作

## Record

A record is represented by one of [models](#).

代表一条模型

## Publisher

A publisher is an optional component. Its common usage is to support atomicity in job level and/or to easily support bulk load into the sink.

publisher是可选组件，支持job级别的自动化或者简单支持批量加载到Sink。

## Job

A job is the highest level component in Databuilder, and it orchestrates a task and, if any, a publisher.

Job是Databuilder中层级最高的组件，包括task或publisher。

## Model

Models are abstractions representing the domain.

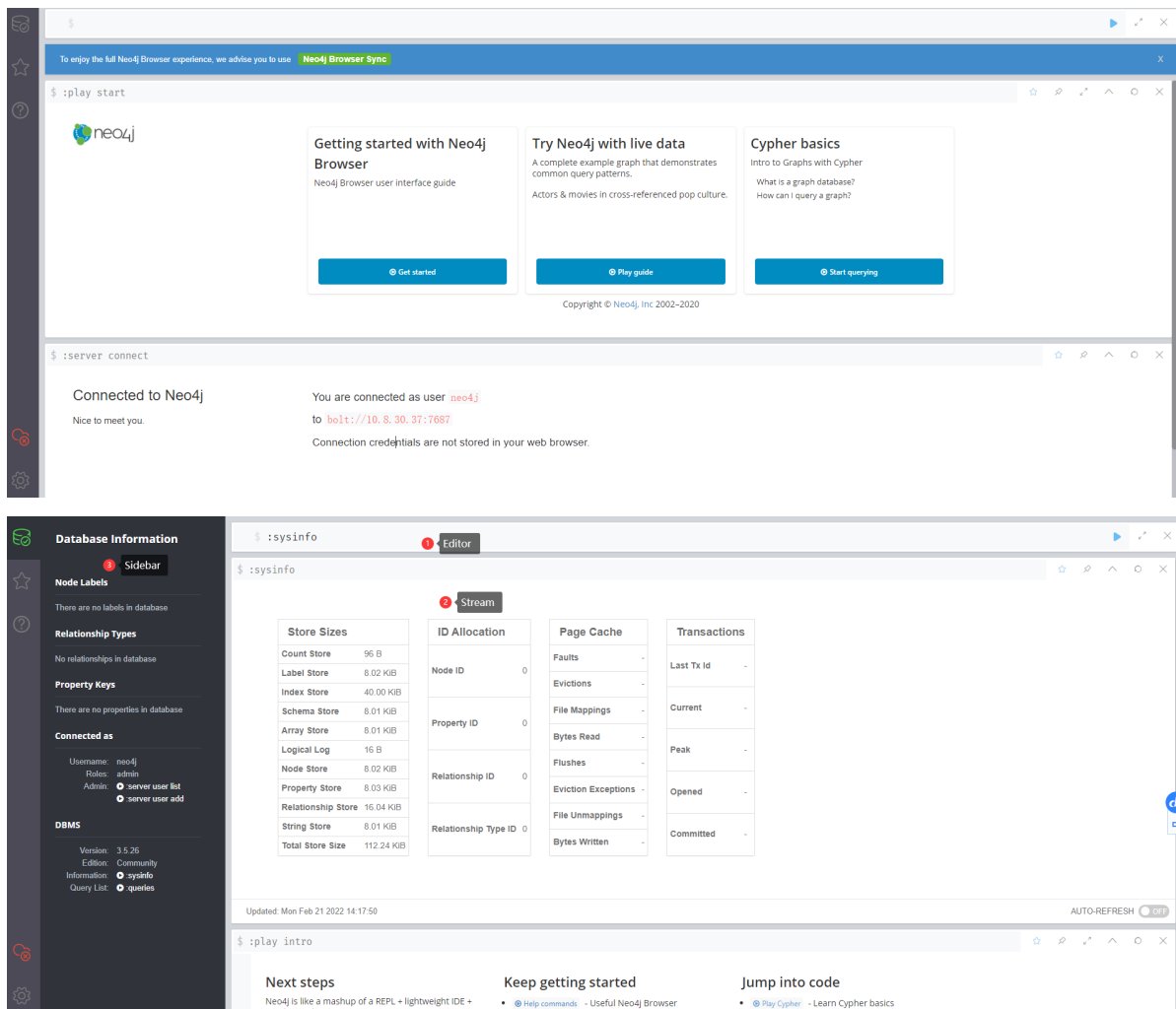
Model（模型）是领域代表的抽象

## Neo4j

[Neo4j](#)是一个高性能的,NOSQL图形数据库。

在docker-compose中启动：

```
neo4j:
  image: neo4j:3.5.26
  container_name: neo4j_amundsen
  environment:
    - NEO4J_AUTH=neo4j/test
  ulimits:
    nofile:
      # 限制打开的文件数
      soft: 40000
      hard: 40000
  ports:
    - 7474:7474
    - 7687:7687
  volumes:
    - ./example/docker/neo4j/conf:/var/lib/neo4j/conf
    - ./example/docker/neo4j/plugins:/var/lib/neo4j/plugins
    - ./example/backup:/backup
    - neo4j_data:/data
  networks:
    - amundsennet
```



## Atlas

<https://atlas.apache.org/>

<https://github.com/apache/atlas> ☆ 1.2k

Atlas是一组可扩展的核心基础治理服务，为组织提供开放的元数据管理和治理功能，以构建其数据资产的目录，对这些资产进行分类和治理，并为数据科学家、分析师和数据治理团队提供围绕这些数据资产的协作能力。

- 能够动态创建分类 - 如 PII、EXPIRES\_ON、DATA\_QUALITY、SENSITIVE
- 集成多重 Hadoop 和非 Hadoop 元数据的预定义类型
- 血统：直观的 UI，可在数据通过各种流程时查看数据沿袭
- 搜索发现：可按类型、分类、属性值或自由文本搜索实体，并支持DSL查询
- 安全：与Apache Ranger集成，支持按分类对数据访问进行授权/数据屏蔽

## Ranger

[https://ranger.apache.org/quick\\_start\\_guide.html](https://ranger.apache.org/quick_start_guide.html)

<https://github.com/apache/ranger> ☆ 582

Apache Ranger™ is a framework to enable, monitor and manage comprehensive data security across the Hadoop platform.



# DataX

DataX是阿里云DataWorks数据集成的开源版本。离线数据同步工具/平台。

链接:

阿里云大数据开发治理平台 [DataWorks](#)

[DataX3.0介绍](#)

架构: Job > 切分Task> 划分TaskGroup > 执行Task (Read+Channel+Writer)

快速使用:

下载: <http://datax-opensource.oss-cn-hangzhou.aliyuncs.com/datax.tar.gz>

运行前:

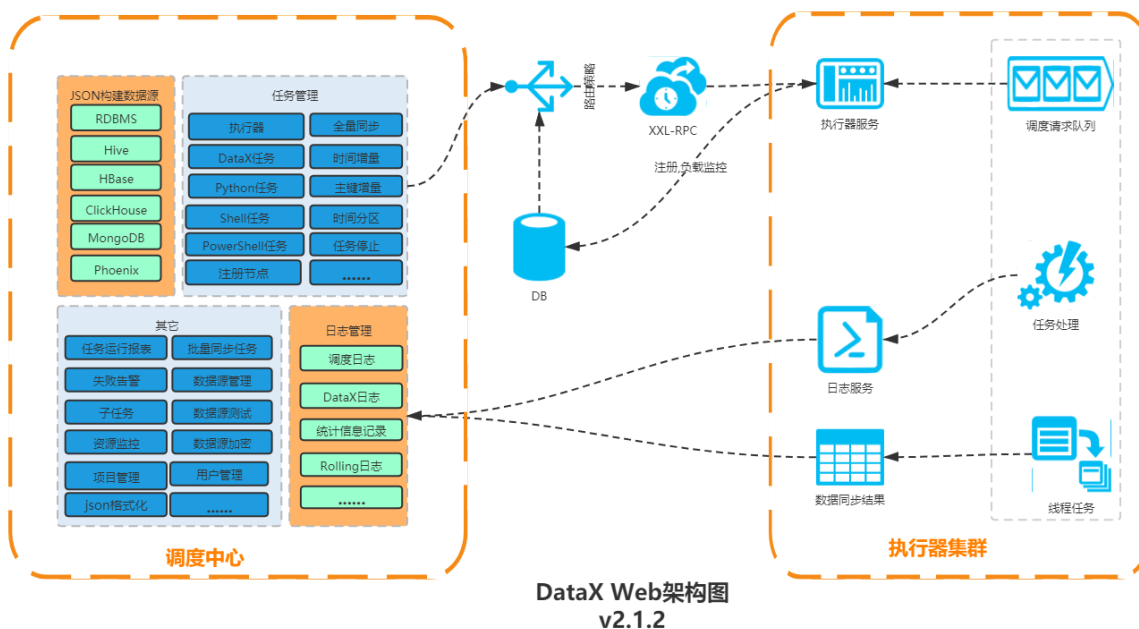
- 删除plugin目录下 (包含reader, writer目录下) 所有 . 开头的文件
- 要在python3环境运行, 需要用datax-web项目doc下的py文件拷贝到当前bin下

配置:

```
# 查看对应模板
$ python datax.py -r streamreader -w streamwriter
# 保存模板并修改, 滞后执行
$ python datax.py ./stream2stream.json
```

## [DataX WEB UI](#)

任务管理和调度、WEB构建DataX json、WEB实时日志、运行参数配置等



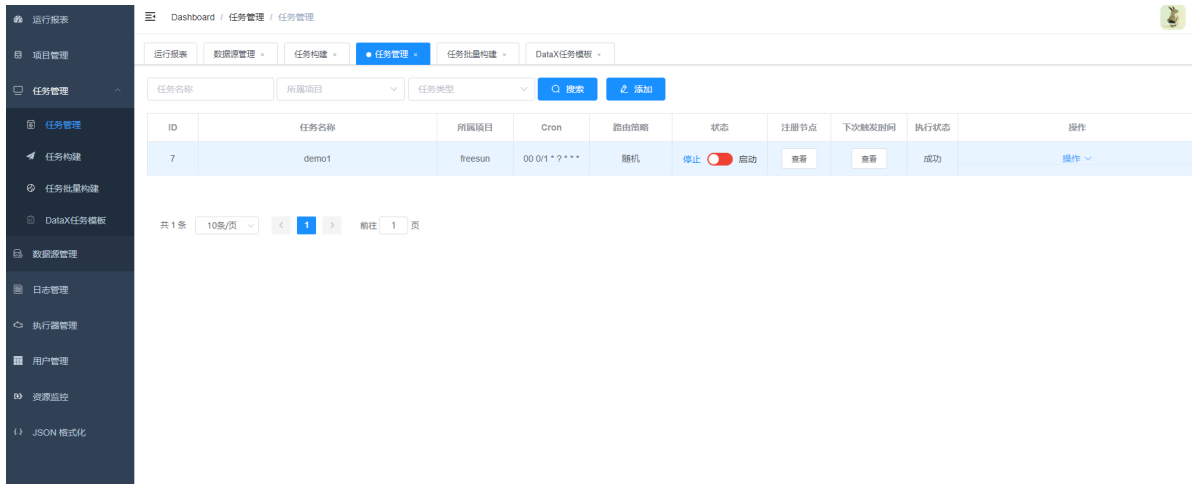
## [DataX Web部署](#)

datax\_admin 控制台: 需指定数据库地址 (bin/sql) / 邮件发送账户、日志、job线程池等

datax\_executor 执行器 (通过python\_path指定datax engine的运行目录)

- datax.job.admin admin地址
- executor.port 执行器端口

- jsonpath datax json临时文件保存路径
- pypath datax启动脚本 `xxx/datax/bin/datax.py`



The screenshot shows a web interface for managing DataX tasks. On the left is a dark sidebar with navigation links: 运行列表, 项目管理, 任务管理, 任务构建, 任务批量构建, DataX任务模板, 数据源管理, 日志管理, 执行器管理, 用户管理, 资源监控, and JSON 格式化. The main content area is titled 'Dashboard / 任务管理 / 任务管理'. It features a breadcrumb trail: 运行列表 > 数据源管理 > 任务管理 > 任务构建 > 任务批量构建 > DataX任务模板. Below this are filters for '任务名称', '所属项目', and '任务类型', along with '搜索' and '添加' buttons. A table lists tasks with columns: ID, 任务名称, 所属项目, Cron, 路由策略, 状态, 注册节点, 下次触发时间, 执行状态, and 操作. One task is visible: ID 7, 任务名称 demo1, 所属项目 freesun, Cron 00 0/1 \* \* \* \*, 路由策略 随机, 状态 停止 (with a red stop icon and '自动' label), 注册节点 查看, 下次触发时间 查看, 执行状态 成功, and 操作 操作. At the bottom, it shows '共 1 条' results, '10条/页', and pagination controls for page 1 of 1.

ID	任务名称	所属项目	Cron	路由策略	状态	注册节点	下次触发时间	执行状态	操作
7	demo1	freesun	00 0/1 * * * *	随机	停止 <span>自动</span>	查看	查看	成功	操作

可以在日志管理中查看任务执行情况和日志输出。

## FlinkX

[DTStack/chunjun](#)

DATAX的升级版，基于flink的分布式数据同步工具。

通过json配置实现批量数据迁移，与Flink-CDC相比，非官方，但有更多的连接方式，但是不支持流式数据处理。

未进一步研究。

## Flink CDC Connectors

[Debezium 简介](#)

Debezium ([DOC](#)) 是一个分布式平台，可将您现有的数据库转换为事件流（CDC）服务。

[官方文档](#)

<https://github.com/ververica/flink-cdc-connectors>

是一个集成Debezium CDC的Flink Connector组件。

第一个例子：

#准备flink环境，这里下载1.13.5版本到node38节点，执行 bin/start-cluster.sh启动standalone集群，通过38:8081访问flink控制台UI

#下载cdc项目git代码

```
mvn spotless:apply
```

```
mvn clean install -DskipTests
```

#完后，对应的cdc jar包在.m2目录 F:\Program Files\apache-maven-3.2.3\mavenRepository\.m2\repository\com\ververica

#这里用到

flink-connector-mysql-cdc-2.2-SNAPSHOT.jar

#拷贝相关cdc的jar包到 FLINK\_HOME/lib/.

最终client lib下的jar包形态:

Quick connect...		
/home/anxin/flink-1.13.6/lib/		
Name	Size (KB)	Last r
..		
log4j-slf4j-impl-2.17.1.jar	23	2022
log4j-core-2.17.1.jar	1 748	2022
log4j-api-2.17.1.jar	294	2022
log4j-1.2-api-2.17.1.jar	203	2022
jackson-databind-2.13.1.jar	1 498	2022
jackson-core-2.13.1.jar	365	2022
jackson-annotations-2.13.1.jar	73	2022
flink-table_2.11-1.13.6.jar	35 600	2022
flink-table-blink_2.11-1.13.6.jar	40 114	2022
flink-sql-connector-postgres-cdc-2.1.1.jar	14 824	2022
flink-sql-connector-mysql-cdc-2.1.1.jar	19 187	2022
flink-sql-connector-kafka_2.11-1.13.5.jar	3 588	2022
flink-sql-connector-elasticsearch7_2.11-1.13.6.jar	27 113	2022
flink-shaded-zookeeper-3.4.14.jar	7 529	2021
flink-json-1.13.6.jar	144	2022
flink-dist_2.11-1.13.6.jar	112 720	2022
flink-csv-1.13.6.jar	90	2022

修改: flink-sql-connector-elasticsearch7\_2.11-1.13.6.jar --> flink-sql-connector-elasticsearch6\_2.11-1.13.5.jar

注意: elasticsearch6连接器参数中需要指定document-type。

Postgresql通过逻辑复制(>10.0 logical replication)实际就是一个CDC应用，等价于消费预写式日志 WAL。

在sql-client CLI中执行:

[例子来源](#)

-- 创建MySQL Product对应表，同步数据库表的数据

```
CREATE TABLE products (  
  id INT,  
  name STRING,  
  description STRING,  
  PRIMARY KEY (id) NOT ENFORCED  
) WITH (  
  'connector' = 'mysql-cdc',  
  'hostname' = '10.8.30.37',  
  'port' = '3306',  
  'username' = 'root',  
  'password' = '123456',  
  'database-name' = 'mydb',  
  'table-name' = 'products'  
);
```

-- 创建订单表映射

```
CREATE TABLE orders (  
  order_id INT,  
  order_date TIMESTAMP(0),  
  customer_name STRING,  
  price DECIMAL(10, 5),  
  product_id INT,  
  order_status BOOLEAN,  
  PRIMARY KEY (order_id) NOT ENFORCED  
) WITH (  
  'connector' = 'mysql-cdc',  
  'hostname' = '10.8.30.37',  
  'port' = '3306',  
  'username' = 'root',  
  'password' = '123456',  
  'database-name' = 'mydb',  
  'table-name' = 'orders'  
);
```

-- 创建快递表 (PostgreSQL)

```
CREATE TABLE shipments (  
  shipment_id INT,  
  order_id INT,  
  origin STRING,  
  destination STRING,  
  is_arrived BOOLEAN,  
  PRIMARY KEY (shipment_id) NOT ENFORCED  
) WITH (  
  'connector' = 'postgres-cdc',  
  'hostname' = '10.8.30.39',  
  'port' = '5432',  
  'username' = 'postgres',  
  'password' = 'postgres',  
  'database-name' = 'trest',  
  'schema-name' = 'public',  
  'table-name' = 'shipments',  
  'debezium.plugin.name' = 'wal2json'  
);
```

-- 目标ES数据表

```
CREATE TABLE enriched_orders (  
  order_id INT,  
  order_date TIMESTAMP(0),  
  customer_name STRING,
```

```

price DECIMAL(10, 5),
product_id INT,
order_status BOOLEAN,
product_name STRING,
product_description STRING,
shipment_id INT,
origin STRING,
destination STRING,
is_arrived BOOLEAN,
PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
    'connector' = 'elasticsearch-6',
    'hosts' = 'http://10.8.30.155:9200',
    'index' = 'enriched_orders',
    'document-type'='_doc'
);

-- 关联查询形成宽表, 插入到ES中
INSERT INTO enriched_orders
SELECT o.*,p.name, p.description, s.shipment_id, s.origin, s.destination,
s.is_arrived
FROM orders AS o
LEFT JOIN products AS p ON o.product_id=p.id
LEFT JOIN shipments AS s ON o.order_id=s.order_id;

```

## MySQL启动CDC

```

#/etc/mysql/mysql.conf.d

[mysqld]
log-bin=mysql-bin
server-id=1
binlog_format=ROW

show variables like 'log_%'; # 显示ON表示打开

```

## Postgres启动CDC [参考](#)

9.6需要使用插件 [wal2json](#)。10+版本不需要使用插件。

decoderbufs  
需要安装一堆插件,在centos上非常难于安装,官方给出的例子也是基于debian操作系统

wal2json  
debezium对wal2json的支持不好,他自己官网说的

pgoutput  
postgres-10 纳入了内核 以下为pgoutput解析出来的逻辑数据

```

#安装wal2json
git clone https://github.com/eulerto/wal2json -b master --single-branch \

```

```

&& cd wal2json \
&& git checkout d2b7fef021c46e0d429f2c1768de361069e58696 \
&& make && make install \
&& cd .. \
&& rm -rf wal2json

#postgres.conf
# MODULES
shared_preload_libraries = 'wal2json'

# REPLICATION
wal_level = logical
max_wal_senders = 4
max_replication_slots = 4

# 数据库用户权限,至少具有REPLICATION和LOGIN权限 (postgres用户默认有权限) (C)
CREATE USER user WITH PASSWORD 'pwd';
ALTER ROLE user replication;
grant CONNECT ON DATABASE test to user;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO user;

# 数据库访问权限 (pg_hba.conf)
local    replication    postgres                                trust
host     replication    postgres          0.0.0.0/0                md5
host     replication    FashionAdmin      0.0.0.0/0                md5
host     replication    FashionAdmin      ::1/128                  md5

# 重启pg服务
service postgresql restart

# 测试wal2json是否启用
pg_recvlogical -d trest --slot test_slot --create-slot -P wal2json

pg_recvlogical -d trest --slot test_slot --start -o pretty-print=1 -f -

# 删除槽
pg_recvlogical -d trest --slot test_slot --drop-slot

```

## 报错记录

1. java.io.StreamCorruptedException: unexpected block data

类加载顺序问题, flink默认是child-first, 在flink的flink-conf.yaml文件中添加  
`classloader.resolve-order: parent-first` 改成parent-first, 重启集群即可。

## Zeppelin Flink解释器

数据科学笔记本。

开发人员将Jupyter描述为“多语言交互式计算环境”。Jupyter Notebook 是一个基于 Web 的交互式计算平台, 结合了实时代码、方程式、叙述性文本、可视化、交互式仪表板和其他媒体。 流行 Python

另一方面，**Apache Zeppelin**被详细描述为“支持交互式数据分析的基于 Web 的笔记本”。支持交互式数据分析的基于 Web 的笔记本。您可以使用 SQL、Scala 等工具制作精美的数据驱动、交互式和协作文档。 [大数据](#) [Python](#) [Scala](#) [SQL](#)

Jupyter 和 Apache Zeppelin 主要可以归类为“**数据科学笔记本**”工具。

与jupyter[详细比较](#)。

[官方](#)文档翻译：

在 Zeppelin 0.9 中，我们重构了 Zeppelin 中的 Flink 解释器以支持最新版本的 Flink。**仅支持 Flink 1.10+，旧版本的 flink 将无法使用。** Zeppelin 支持 Apache Flink，Flink 解释器组由下面列出的五个解释器(interpreter)组成。

姓名	班级	描述
%flink	Flink 解释器	创建 ExecutionEnvironment/StreamExecutionEnvironment/BatchTableEnvironment/StreamTableEnvironment 并提供 <b>Scala</b> 环境
%flink.pyflink	PyFlink 解释器	提供python环境
%flink.ipynb	IPyFlink解释器	提供 ipython 环境
%flink.ssql	FlinkStreamSqlInterpreter	提供流式sql环境
%flink.bsql	FlinkBatchSqlInterpreter	提供批处理sql环境

## 特点

特征	描述
支持多个版本的 Flink	您可以在一个 Zeppelin 实例中运行不同版本的 Flink
支持多个版本的 Scala	您可以在 Zeppelin 实例上运行不同的 Scala 版本的 Flink
支持多种语言	支持Scala、Python、SQL，此外你还可以跨语言协作，例如你可以编写Scala UDF 并在PyFlink中使用它
支持多种执行模式	Local   Remote   Yarn   Yarn Application
支持Hive	支持 Hive 目录
交互式开发	交互式开发用户体验提高您的生产力
Flink SQL 的增强	<ul style="list-style-type: none"><li>* 一个笔记本同时支持流式和批处理sql</li><li>* 支持sql注释（单行注释/多行注释）</li><li>* 支持高级配置（jobName，并行度）</li><li>* 支持多条insert语句</li></ul>
多租户	多个用户可以在一个 Zeppelin 实例中工作而不会相互影响。
休息 API 支持	你不仅可以通过 Zeppelin notebook UI 提交 Flink 作业，还可以通过它的 rest api 来提交（你可以使用 Zeppelin 作为 Flink 作业服务器）。

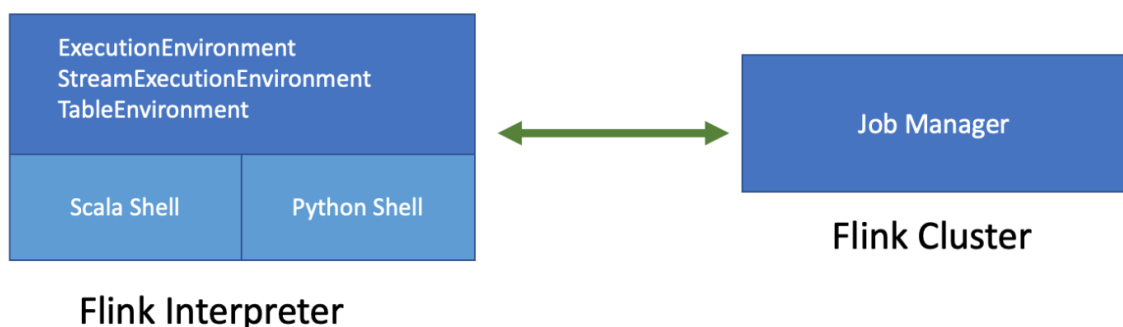
## 快速启动

(dockers, 映射本地flink二进制程序到容器中, 验证本地模式) (8081端口: flink UI)

```
docker run -u $(id -u) -p 8080:8080 -p 8081:8081 --rm -v /mnt/disk1/flink-1.12.2:/opt/flink -e FLINK_HOME=/opt/flink --name zeppelin apache/zeppelin:0.10.0

# run on node37
docker run -u $(id -u) -p 8080:8080 --rm -v /home/anxin/flink-1.12.1:/opt/flink -e FLINK_HOME=/opt/flink -e flink.execution.mode=remote -e flink.execution.remote.host=10.8.30.37 -e flink.execution.remote.port=6123 --name zeppelin apache/zeppelin:0.10.0
```

## 架构



上图是 Flink on Zeppelin 的架构。左侧的 Flink 解释器实际上是一个 Flink 客户端，负责编译和管理 Flink 作业生命周期，例如提交、取消作业、监控作业进度等。右侧的 Flink 集群是执行 Flink 作业的地方。它可以是 MiniCluster（本地模式）、Standalone 集群（远程模式）、Yarn 会话集群（yarn 模式）或 Yarn 应用会话集群（yarn-application 模式）

Flink 解释器中有两个重要的组件：Scala shell & Python shell

- Scala shell 是 Flink 解释器的入口点，它会创建 Flink 程序的所有入口点，例如 **ExecutionEnvironment**，**StreamExecutionEnvironment** 和 **TableEnvironment**。Scala shell 负责编译和运行 Scala 代码和 sql。
- Python shell 是 PyFlink 的入口，它负责编译和运行 Python 代码。

## 配置

连接到Flink集群：



财产	默认	描述
<code>FLINK_HOME</code>		Flink 安装位置。必须指定，否则不能在 Zeppelin 中使用 Flink
<code>HADOOP_CONF_DIR</code>		hadoop conf的位置，如果在yarn模式下运行必须设置
<code>HIVE_CONF_DIR</code>		hive conf 的位置，如果要连接到 hive 元存储，必须设置此项
<code>flink.execution.mode</code>	local	Execution mode of Flink, e.g. local   remote   yarn   yarn-application
<code>flink.execution.remote.host</code>		运行 JobManager 的主机名。仅用于远程模式
<code>flink.execution.remote.port</code>		运行 JobManager 的端口。仅用于远程模式

## 第三方依赖

通过[内联配置](#)实现具体note下的依赖包：

1. `flink.execution.packages`。类似pom中添加依赖

```
%flink.conf

flink.execution.packages  org.apache.flink:flink-connector-
kafka_2.11:1.10.0,org.apache.flink:flink-connector-kafka-
base_2.11:1.10.0,org.apache.flink:flink-json:1.10.0
```

2. `flink.execution.jars`。配置外部依赖的jar包文件路径

```
%flink.conf

flink.execution.jars  /usr/lib/flink-kafka/target/flink-kafka-1.0-
SNAPSHOT.jar
```

## Flink UDF

在 Zeppelin 中有 4 种方法可以定义 UDF。

- 编写 Scala UDF

```
%flink

class ScalaUpper extends ScalarFunction {
  def eval(str: String) = str.toUpperCase
}

btenv.registerFunction("scala_upper", new ScalaUpper())
```

- 编写 PyFlink UDF
- 通过 SQL 创建 UDF

通过IDE编写复杂的UDF库，然后通过SQL语句注册到环境。一般需配合通过flink.udf.jar方式引入库。

```
%flink.sql
```

```
CREATE FUNCTION myupper AS 'org.apache.zeppelin.flink.udf.JavaUpper';
```

- 通过 flink.udf.jar 配置 udf jar

## Hive

配置：

zeppelin.flink.enableHive=true

HIVE\_CONF\_DIR

复制：

- flink-connector-hive\_2.11-\*.jar
- flink-hadoop-compatibility\_2.11-\*.jar
- hive-exec-2.x.jar

本地属性：（解释器上面的参数）

Property	Default	Description
type		Used in %flink.sql to 指定流式数据的显示类型 (single, update, append)
refreshInterval	3000	Used in %flink.sql 指定流式数据刷新周期.
template	{0}	Used in %flink.sql to specify html template for single type of streaming data visualization, And you can use {i} as placeholder for the {i}th column of the result.
parallelism		Used in %flink.sql & %flink.bsql to specify the flink sql job parallelism
maxParallelism		Used in %flink.sql & %flink.bsql to specify the flink sql job max parallelism in case you want to change parallelism later. For more details, refer this <a href="#">link</a>
savepointDir		If you specify it, then when you cancel your flink job in Zeppelin, it would also do savepoint and store state in this directory. And when you resume your job, it would resume from this savepoint.
execution.savepoint.path		When you resume your job, it would resume from this savepoint path.
resumeFromSavepoint		Resume flink job from savepoint if you specify savepointDir.
resumeFromLatestCheckpoint		Resume flink job from latest checkpoint if you enable checkpoint.
runAsOne	false	All the insert into sql will run in a single flink job if this is true. (让多条SQL insert语句在同一个job中运行)

## 附录：相关阅读

以下信息摘抄自 Oracle Big Data Blog

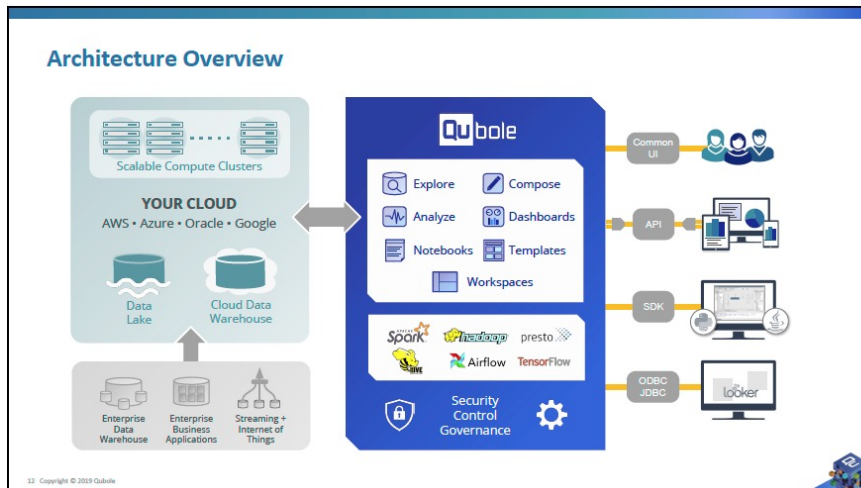
### 数据湖安全

1. 建立治理：为所有的数据构建数据湖。作为原始和非结构化数据的存储库，它可以从任何来源提取几乎任何东西。要了解如何管理、处理和使用这些数据，避免形成数据沼泽。通过治理，可以识别所有权、敏感数据的安全规则、数据历史、数据血缘
2. 访问：建立不同级别的用户访问权限
3. 使用机器学习：一些数据湖平台带有内置的机器学习 (ML) 功能。机器学习的使用可以通过加速原始数据处理和分类来显著降低安全风险

4. 分区和层次结构：当数据被摄取到数据湖中时，将其存储在适当的分区中很重要。在《数据湖最佳实践》中包括原始区、生产区、工作区和敏感区。这里从不同维度划分：时间、原始（原始，包含对敏感的加密后）、受信任（便于用户-科学家、分析师等轻松访问）、精炼（最终输出）
5. 数据生命周期：淘汰陈旧数据，甚至通过ML实现自动实现数据的冷热切换，最大化资源利用率
6. 数据加密：加密对数据安全至关重要，尤其敏感数据

## Qubole

用于分析，人工智能，机器学习的云原生平台。



集成的组件：Spark/Presto/Hive/Quantum/Airflow(工作流ETL)

任何数据湖设计都应包含元数据存储策略，以使业务用户能够搜索、定位和了解湖中可用的数据集。

- 可以强制创建元数据
- 自动创建元数据
- 允许搜索元数据的数据目录
- 访问和挖掘（Access and Mining）:读取模式
  - Schema on Write -> Schema on Read
- 数据处理
  - 组合连接不同的数据集
  - Denormalization 反范式化

题外：例如将用户和用户地址表join'存储就属于反范式化设计；范式化虽然节省存储空间、简化更新，但反范式化方便查询。提升数据库性能的手段就可以包括：索引、物化视图、缓存、反范式化

- Cleansing, deduplication, householding。清洗、去重、家计
- 派生计算数据字段

**数据湖平台应提供：**

- 多种数据处理引擎选项，例如 Spark、Hadoop/Hive、Presto 等。这对于支持广泛的用例至关重要。
- 一个基于开放标准的 Metastore，例如 Hive，然后可以从 Hive、Presto 和 Spark SQL 中使用
- 与 AWS Glue 的目录集成。
- 支持可用于从元数据中获取有用信息的 AIR（警报、见解和建议）
- 支持 Kafka Schema 注册表（用于流数据集）。
- 数据仓库解决方案的连接器，例如 Snowflake、Redshift、BigQuery、Azure SQL 数据库等。

- MySQL、Oracle、MongoDB、Vertica、SQL Server 等流行商业数据库的连接。
- 无服务器计算选项（例如 Presto）可以经济高效地满足交互式查询要求。
- 基于浏览器的统一 UI 供分析师运行查询。
- 用于从 BI 工具（如 Tableau、Looker、Click View、SuperSet、Redash 等）进行查询的 JDBC/ODBC 驱动程序。
- 适用于数据科学家和分析师的 Jupyter/Zepplin notebook。
- 适用于 Python 和 R 的基于 UI 的数据科学包管理。

## AirFlow

工作流管理平台 (Python)

[官网](#)、[入门概念](#)

实操Shell (docker-compose quick-start)

```
curl -Lfo 'https://airflow.apache.org/docs/apache-airflow/2.2.3/docker-
compose.yaml'

# 设置文件夹权限: docker-compose mount dirs
mkdir -p ./dags ./logs ./plugins
echo -e "AIRFLOW_UID=$(id -u)" > .env

# 初始化数据库
docker-compose up airflow-init

# 清理环境: 清理目录
# docker-compose down --volumes --remove-orphans

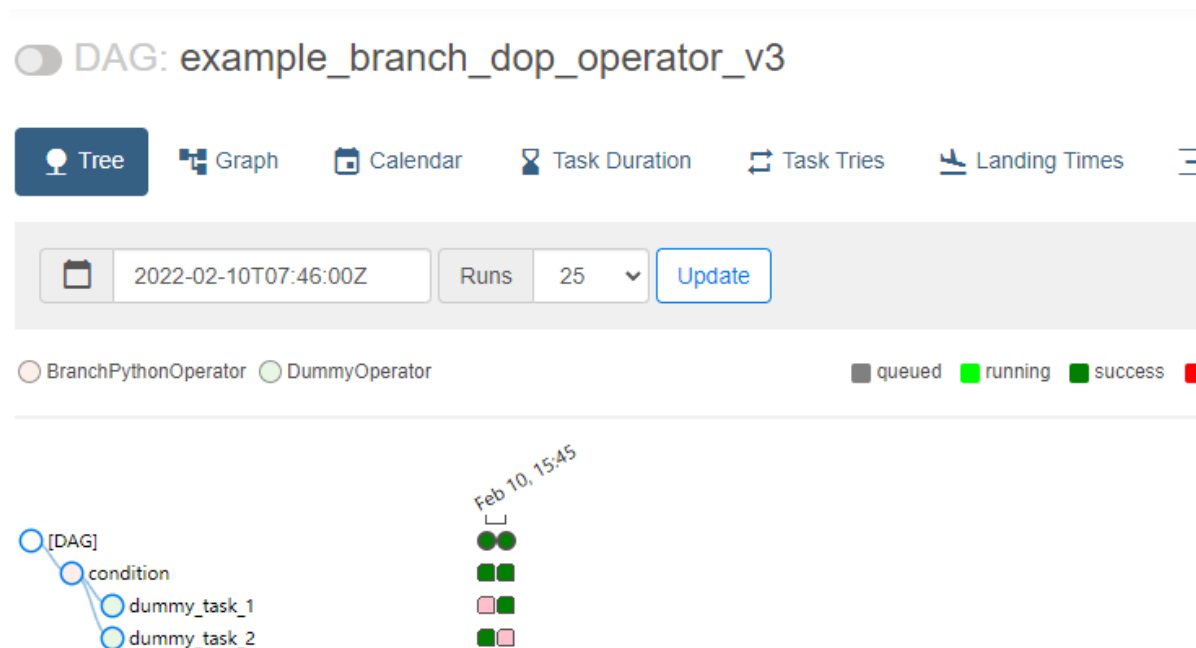
# 启动AirFlow
docker-compose up
```

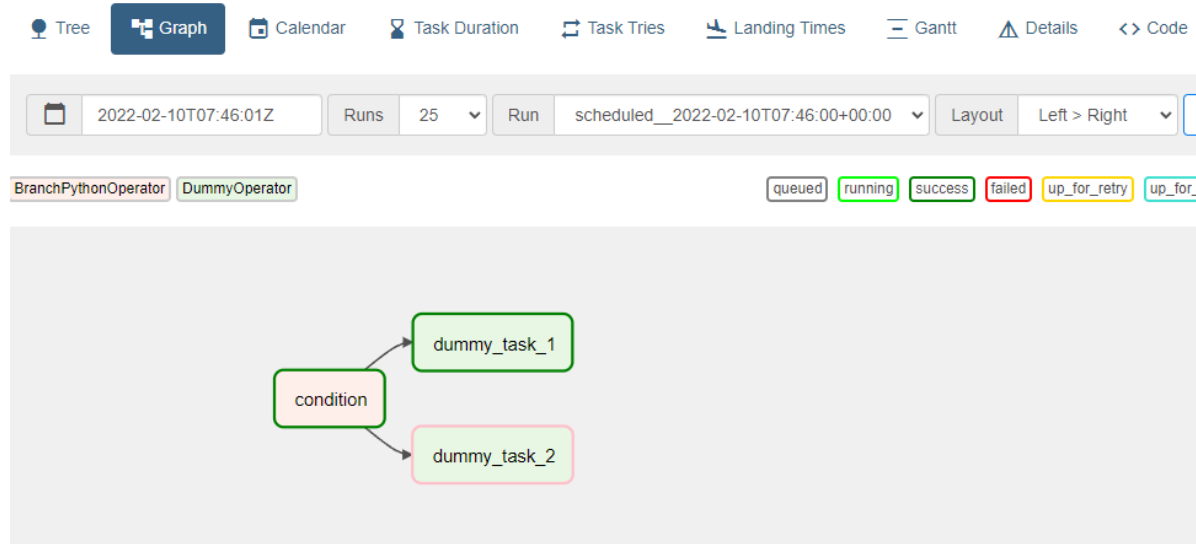
quick-start中的模块:

- `airflow-scheduler` - The [scheduler](#) monitors all tasks and DAGs, then triggers the task instances once their dependencies are complete.
- `airflow-webserver` - The webserver is available at `http://localhost:8080`.
- `airflow-worker` - The worker that executes the tasks given by the scheduler.
- `airflow-init` - The initialization service.
- `flower` - [The flower app](#) for monitoring the environment. It is available at `http://localhost:5555`.
- `postgres` - The database.
- `redis` - [The redis](#) - broker that forwards messages from scheduler to worker.

通过CLI访问:

```
docker-compose run airflow-worker airflow info
# 或者
curl -Lfo 'https://airflow.apache.org/docs/apache-airflow/2.2.3/airflow.sh'
chmod +x airflow.sh
./airflow.sh info # 显示airflow服务信息
./airflow.sh bash # 进入容器bash
./airflow.sh python # 进入python控制台
```





## 大数据处理中的Lambda架构和Kappa架构

- 数据采集

将应用程序产生的数据和日志等同步到大数据系统中，由于数据源不同，这里的数据同步系统实际上是多个相关系统的组合。数据库同步通常用 Sqoop（目前项目退休，使用spark或flink替代），日志同步可以选择 Flume，打点采集的数据经过格式化转换后通过 Kafka 等消息队列进行传递。

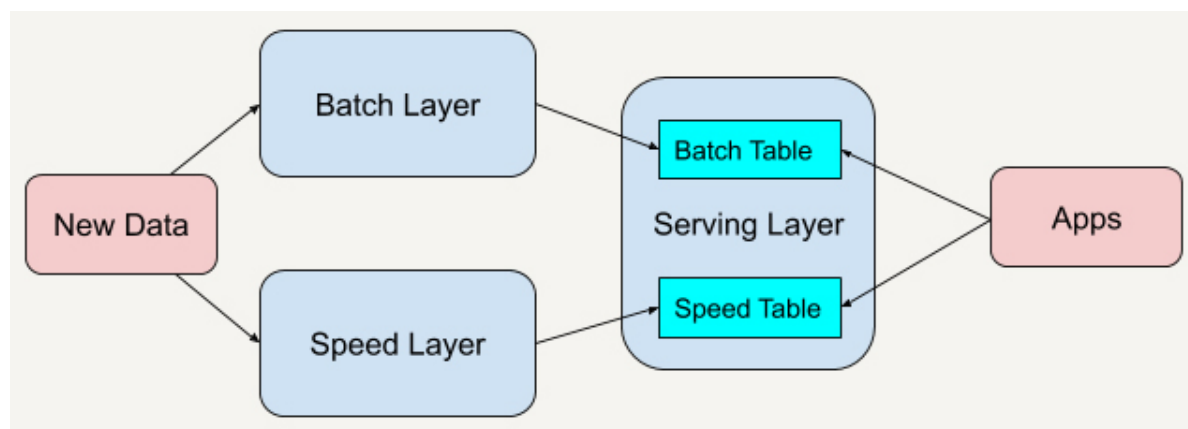
- 数据处理

这部分是大数据存储与计算的核心，数据同步系统导入的数据存储在 HDFS。MapReduce、Hive、Spark 等计算任务读取 HDFS 上的数据进行计算，再将计算结果写入 HDFS。

- 数据输出与展示

大数据计算产生的数据还是写入到 HDFS 中，但应用程序不可能到 HDFS 中读取数据，所以必须要将 HDFS 中的数据导出到数据库中。

Lambda架构：

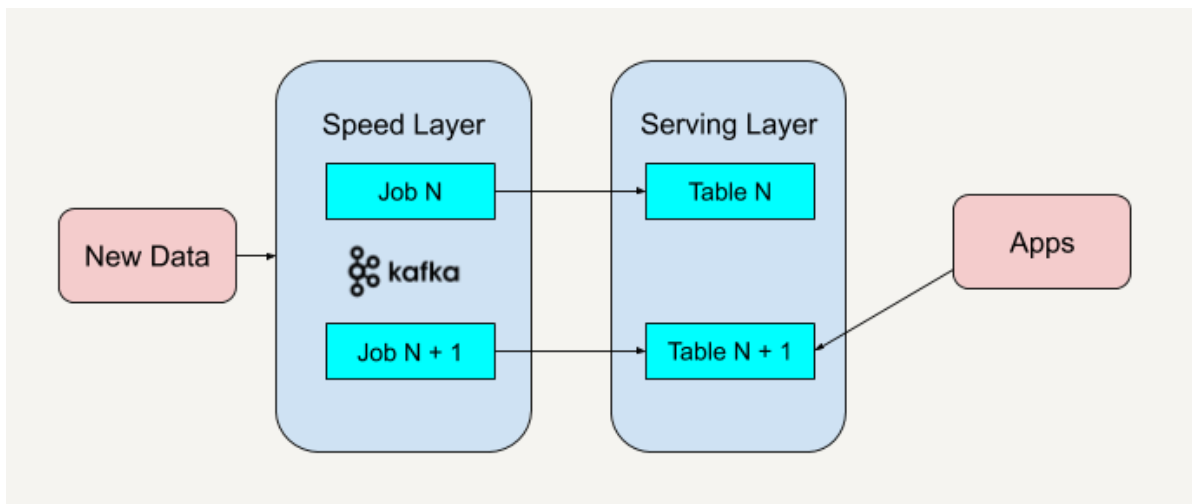


Lambda架构(Lambda Architecture),总共由三层系统组成：

批处理层（Batch Layer），速度处理层（Speed Layer）以及响应查询的服务层（Serving Layer）。

Kappa架构：

以kafka为例说明：



去掉了批处理层，仅保留了速度层。实现流批一体

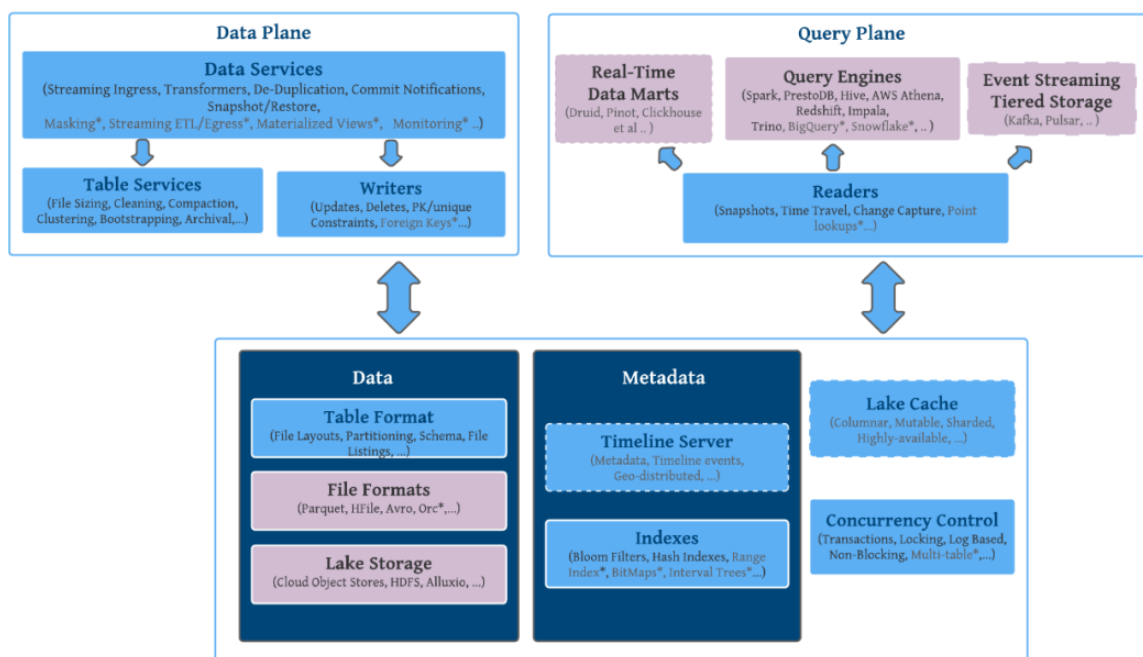
以下为其他内容补充

## Hudi Pk Iceberg

[终于有人将数据湖讲明白了](#)

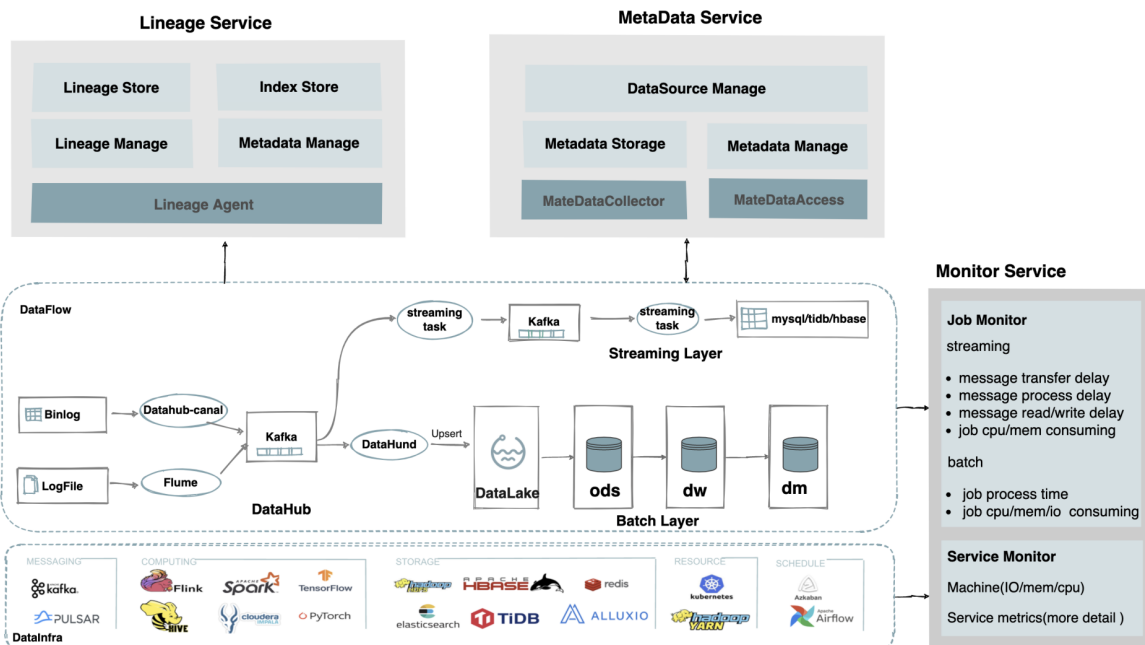
[Hudi设计和架构解读](#)

[新一代流式数据湖平台](#)



网易严选Iceberg应用实践





Datahub-canal: 主要用途是基于 MySQL 数据库增量日志解析，提供增量数据订阅和消费。

Flume: Apache Flume是一个分布式、可靠及可用的服务，用于有效地收集、聚合和移动大量日志数据。

DataLake 数据湖

ODS (Operational Data Store) 运营数据存储. 面向主题的、集成的、可更新的、当前或接近当前的。

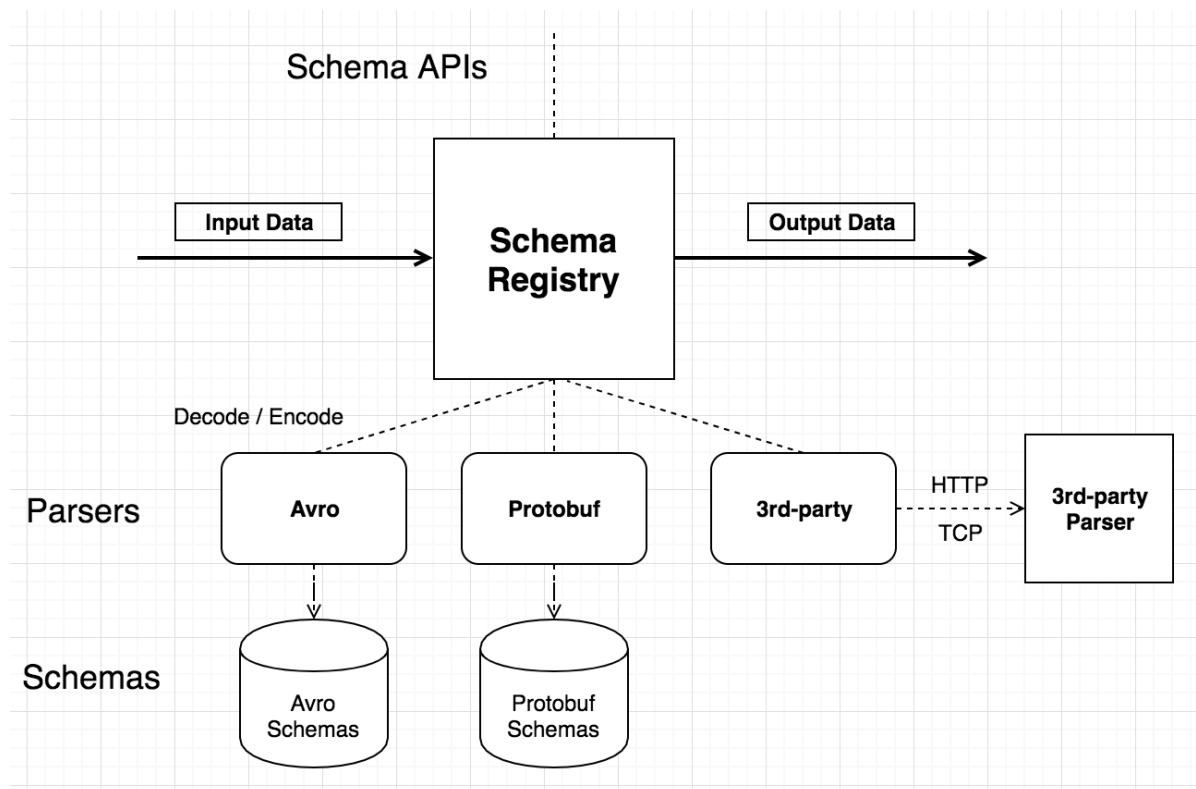
DW (Data Warehouse) 数据仓储

DM (Data Market) 数据集市

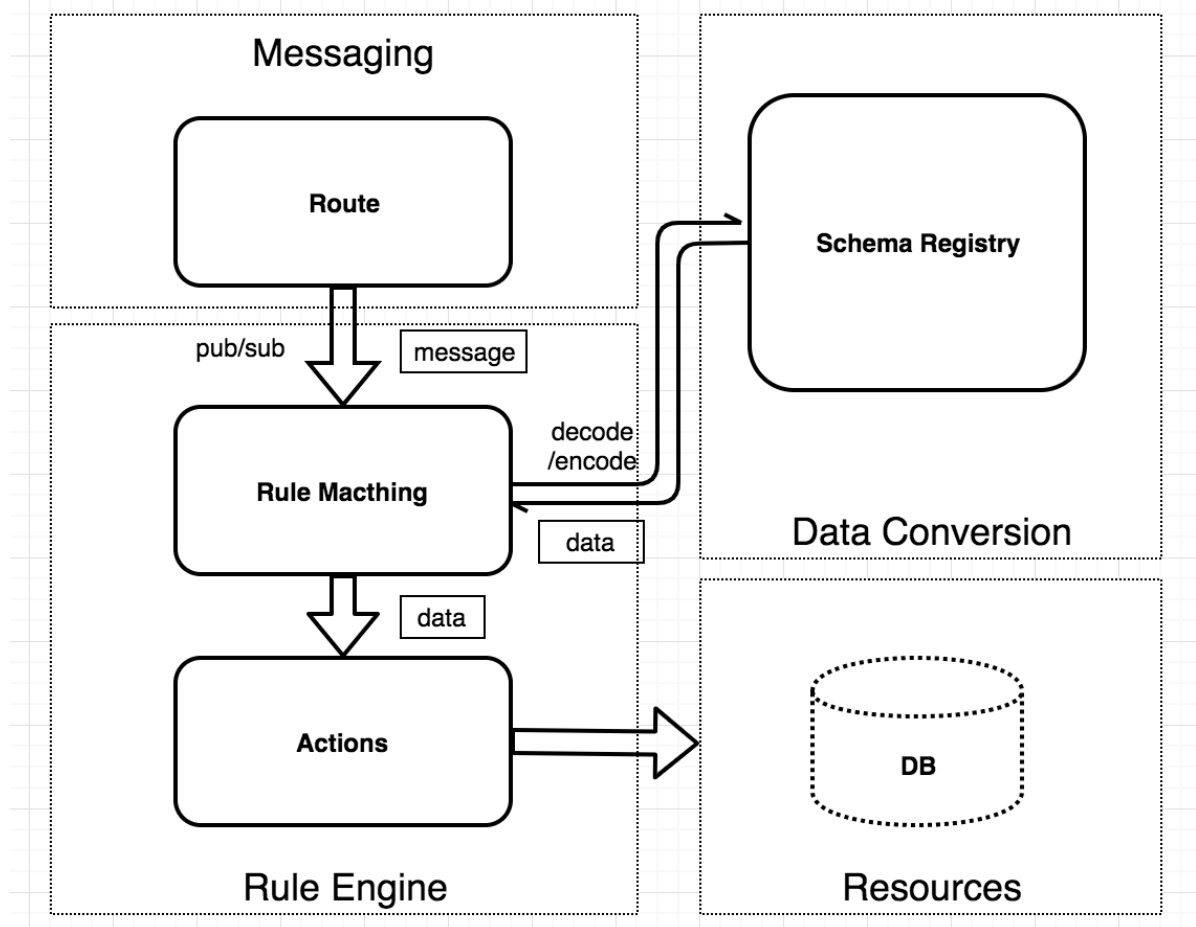
## Schema Registry

【[Github](#)】Confluent Schema Registry provides a serving layer for your metadata. It provides a RESTful interface for storing and retrieving your Avro®, JSON Schema, and Protobuf schemas. It stores a versioned history of all schemas based on a specified subject name strategy, provides multiple compatibility settings and allows evolution of schemas according to the configured compatibility settings and expanded support for these schema types. It provides serializers that plug into Apache Kafka® clients that handle schema storage and retrieval for Kafka messages that are sent in any of the supported formats.

[博文](#)中介绍了EMQ X企业版中Schema Registry功能，处理各类编解码协议。



Schema Registry和Rule Engine在EMQX中的应用:



# Oracle大数据湖服务集成工具

1. ML
2. 数据目录
3. Analytics 集成的数据分析工具
4. Graph Analytics 集成的图形分析工具