Amundsen平台使用总结：

官方资料

平台官方网站：https://www.amundsen.io/amundsen/

项目github页面：https://github.com/amundsen-io/amundsen

官方部署教程（neo4j与atlas两种后端）https://www.amundsen.io/amundsen/installation/

部署流程

1. 要求服务器安装了Docker和Docker-compose:

Linux上安装Docker:https://docs.docker.com/desktop/install/linux-install/

https://zhuanlan.zhihu.com/p/54147784

Linux上安装Docker-compose: https://docs.docker.com/compose/install/linux/

2. 由于服务器连接Github相当不稳定，Amundsen项目中包含了多个子项目，在git clone的过程中会经常遇到连接断开的情况，采用的方法是本地直接下载项目包后传给服务器。

```
1       git clone --recursive https://github.com/amundsen-io/amundsen.git
```

3. 最终使用了Neo4j作为后端，进入Github项目文件夹后，在命令行输入命令：

```
1 docker-compose -f docker-amundsen.yml up
```

4. 进入Github项目文件夹下的databuilder目录，配置用于Data ingestion的虚拟环境：

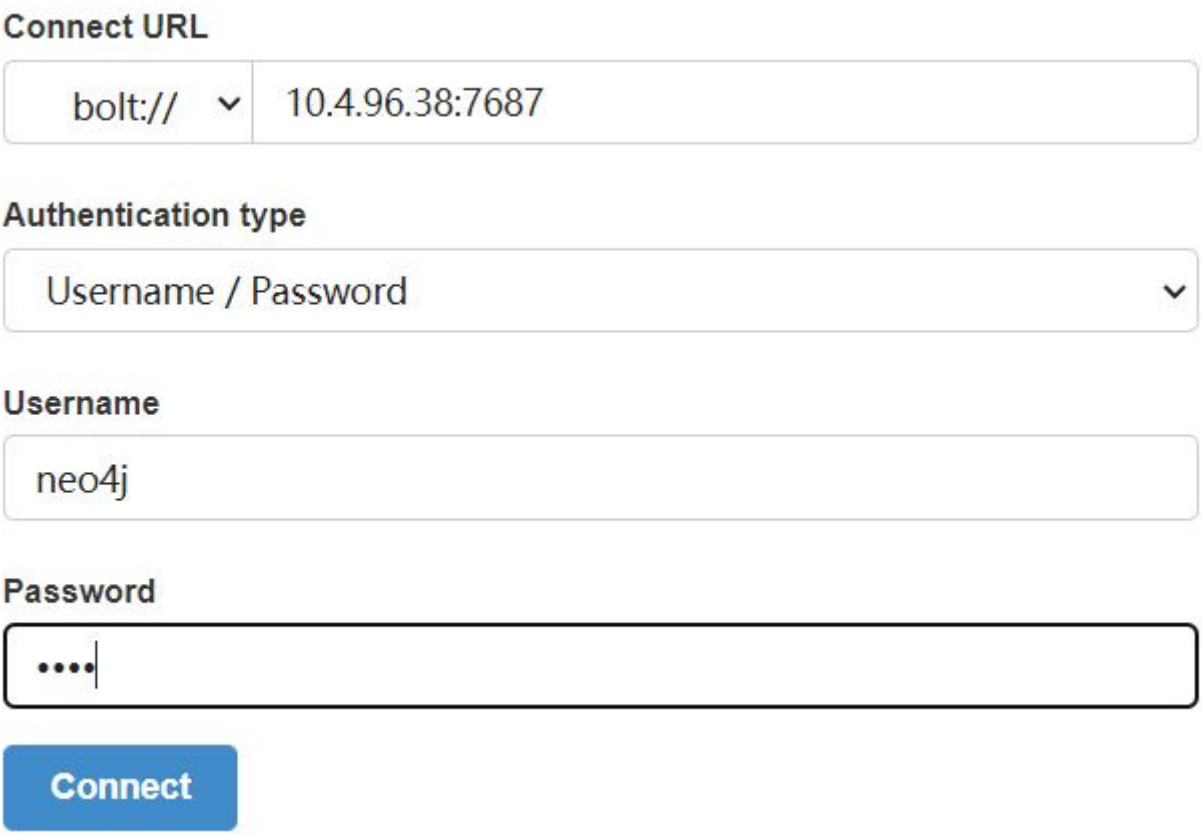
```
1 python3 -m venv venv
2 source venv/bin/activate
3 pip3 install --upgrade pip
4 pip3 install -r requirements.txt
5 python3 setup.py install
```

尝试导入Amundsen提供的示例数据：

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

5. 上述步骤完成后通过隧道连接到5000端口，可以看到Amundsen前端,Amundsen没有登入控制，无需账号密码。

6. 通过隧道连接到7474端口，可以看到Neo4j后端，默认的登入账号为neo4j/root，注意登入时需要将Connect URL默认的Localhost改为服务器IP，此处为38服务器的IP。



功能使用

元数据导入

使用mysql作为导入源，mysql的安装和部署参考链接：

https://blog.csdn.net/m0_67392010/article/details/126034669

1. 收集数据集，将非结构化原始数据转化为结构化的表格

2. 将数据导入mysql中，个人在实验中使用了navicat软件远程连接服务器上的mysql，并直接将本地的CSV文件上传至服务器的mysql中。

该方案在外网上需要开放服务器的3306端口，建议做好防护，或不使用navicat在服务器上用sql命令完成导入。

3. 修改官方提供的从mysql导入数据的脚本，使用modified_script文件夹下sample_mysql_loader.py文件代替官方Github项目目录../databuilder/example/scripts/下的同名文件，然后回到databuilder目录下，使用如下命令导入数据：

```
source venv/bin/activate
python3 example/scripts/sample_mysql_loader.py
```

```
def connection_string():
    user = 'root:Lst990808!'
    host = 'localhost'
    port = '2023'
    db = 'funsd'
# I have changed this code -zhouyifan for python3 using mysql
#    return "mysql://%s@%s:%s/%s" % (user, host, port, db)
    return "mysql+pymysql://%s@%s:%s/%s" % (user, host, port, db)
```

```
where_clause_suffix = textwrap.dedent("""
    where c.table_schema = 'funsd'
""")
```

主要修改的参数如两张图所示，包括了Mysql连接密钥，端口和指定需要导入的database，注意c.table_schema后值也需要改为你想要导入的mysql database名。

元数据删除

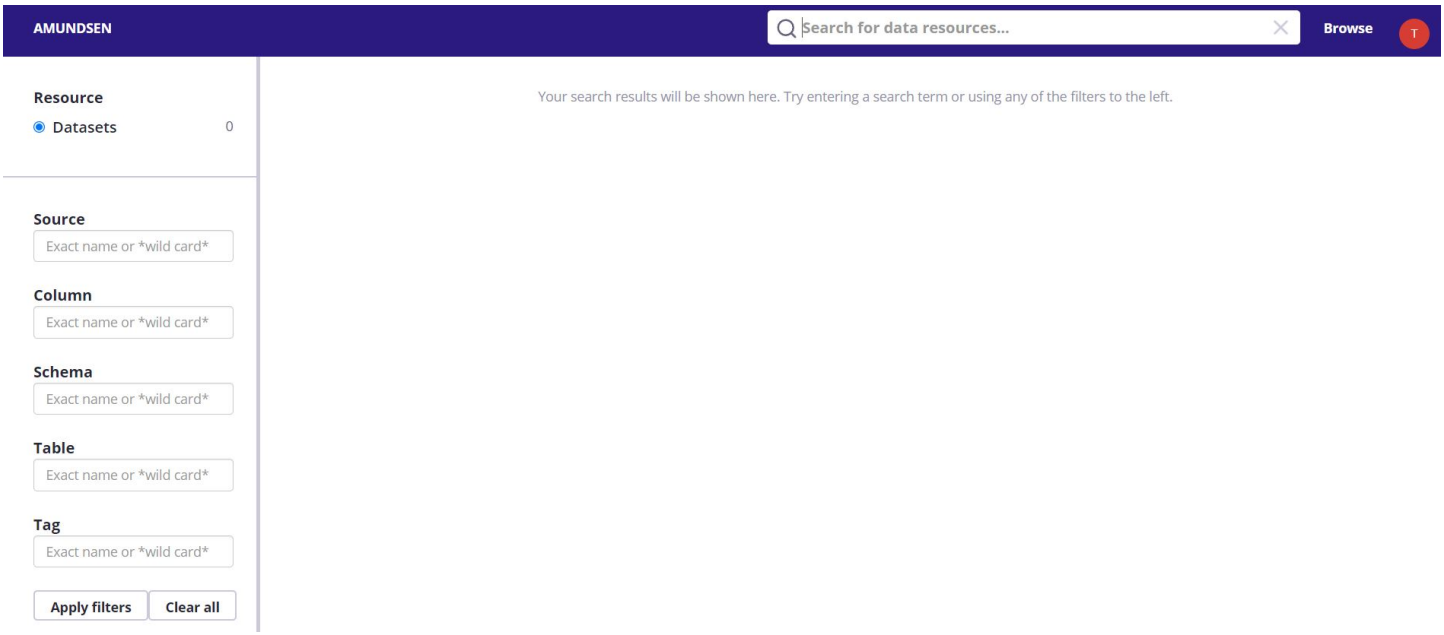
目前使用的元数据清理方法是直接在后端Neo4j图数据库中，通过cypher语言命令，删去不需要的节点和关系，此时结果反馈在前端是对应项的消失。

下图展示的是一个清空Amundsen内所有导入的元数据的例子。

```
MATCH (n)
OPTIONAL MATCH (n)-[r]-()
DELETE n,r
```

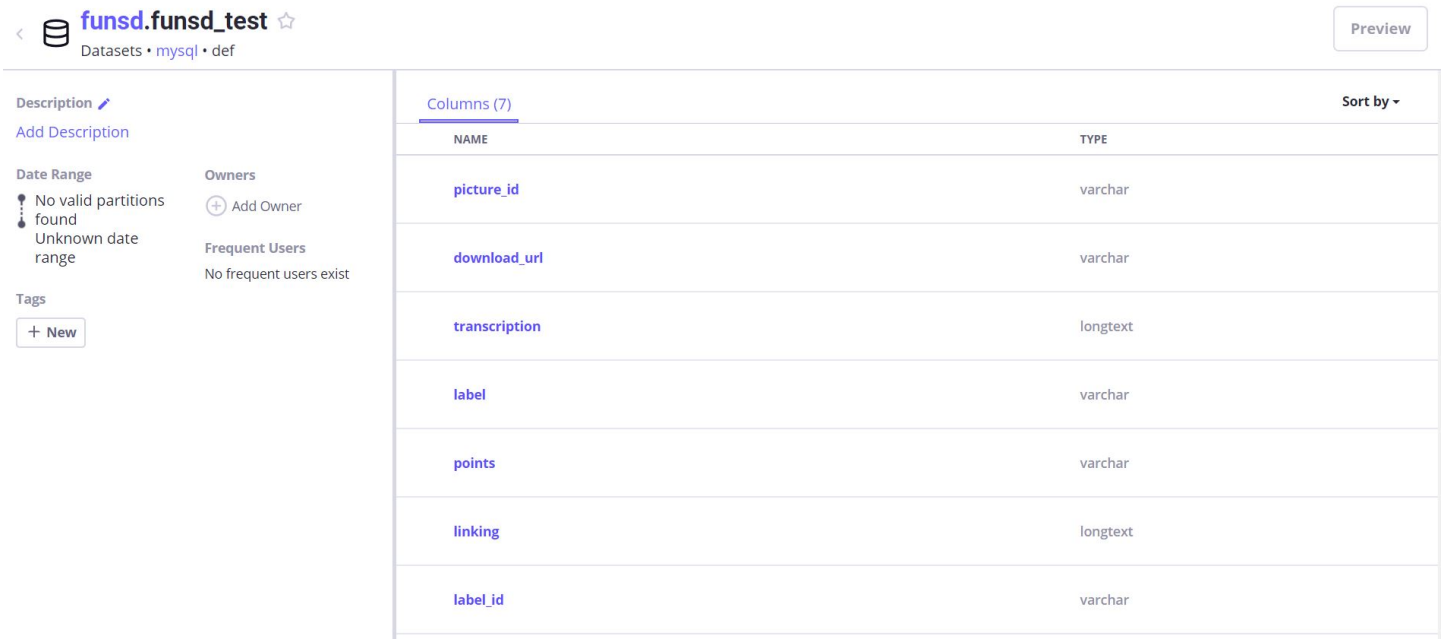
元数据检索

- 基于Elastic search的元数据检索：支持5种高级检索



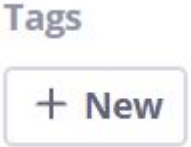
检索时以"_"为分割，需要输入两个“_"之间完整的字符串，或者使用正则表达式

检索结果可以显示数据集元数据表的来源，所属schema，列名和列数据种类。



标签添加

1. Amundsen提供了两种标签，分别是tag和badge，tag可以在前端交互的过程中进行添加和修改。



2. Badge则只能通过脚本从后端命令行输入添加。
    a. 使用的脚本为add_badge文件夹下的add_badge.py文件，使用时将其放在Github项目的../databuilder/example/文件夹下。

b. 使用前准备两个csv文件，分别是sample_table.csv和sample_badge.csv。

sample_table.csv 中包含所有需添加badge元数据表的database、cluster、schema、name。

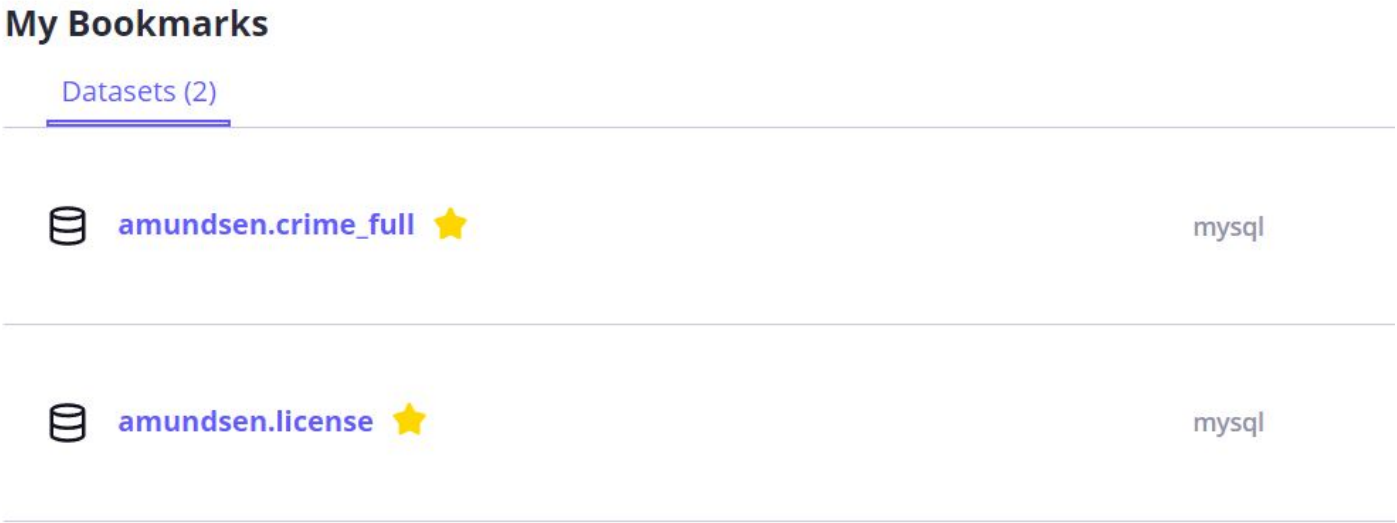
sample_badge.csv包含了需要添加badge的名称和badge种类（table/column），以及对应所属元数据表的database、cluster、schema、name。

badge添加后在前端展示效果如下：

tag和badge添加后都会展示在主页面上，方便用户搜索。

数据集收藏

对需要添加至收藏夹的数据集，点击数据旁边的五角星，使其变为黄色，如此主页便会显示用户收藏的数据集。

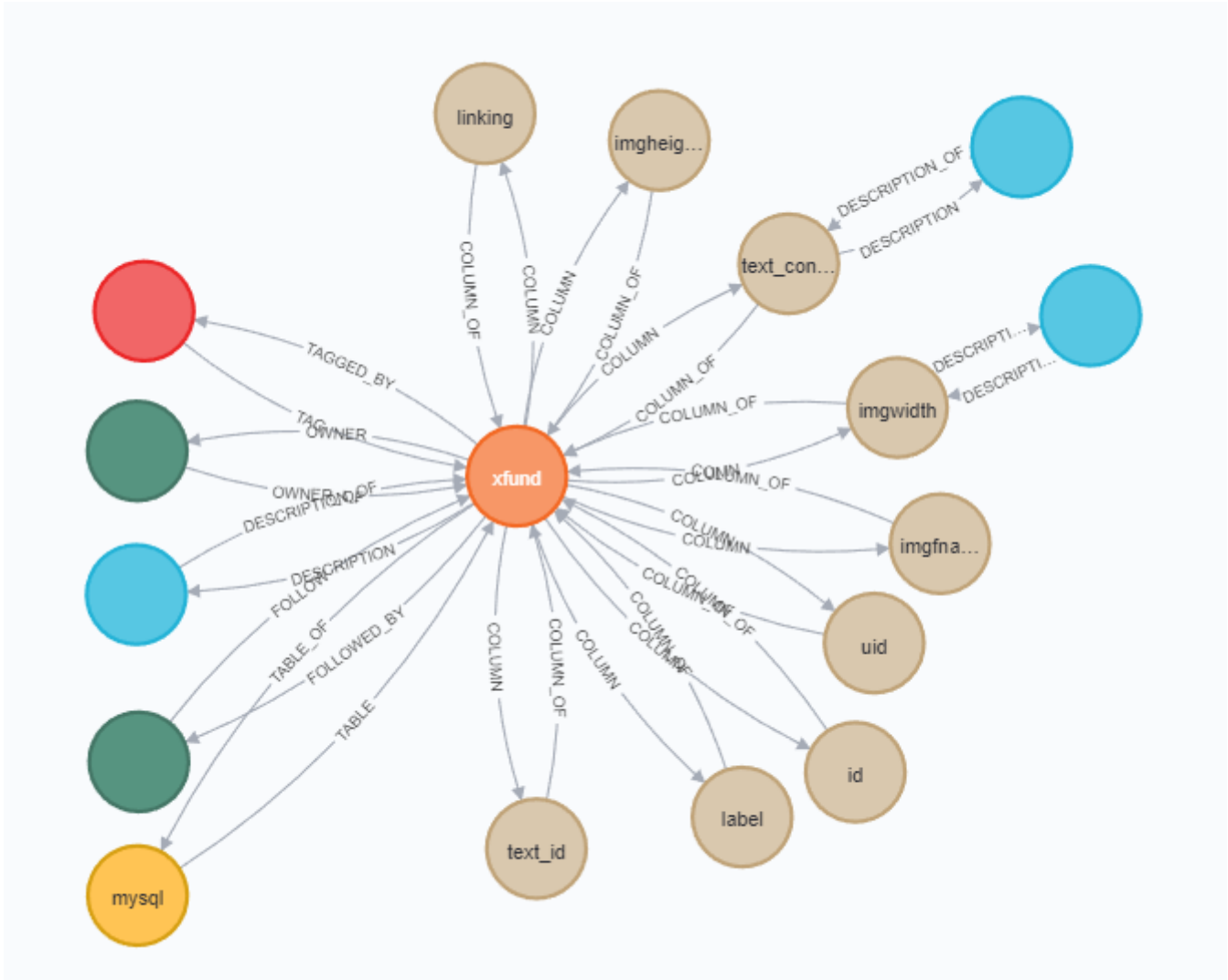


对Amundsen使用的后端数据库Neo4j的调研：

Elastic search与Neo4j组件的协作实现了amundsen的前后端交互。

Neo4j数据结构

以下图为例，介绍一个数据集元数据在Neo4j图数据库中的基本存储方式：



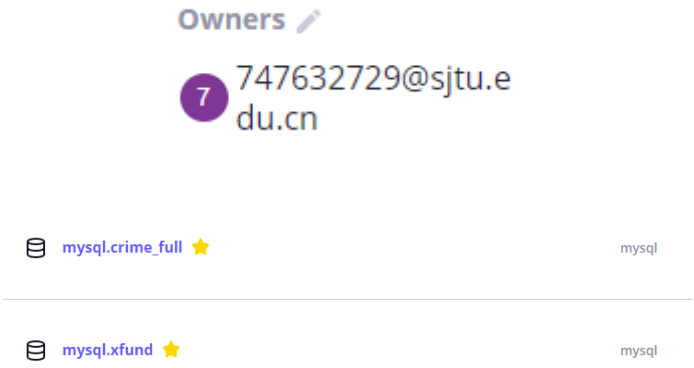
- 图中心橙色xfund节点代表Neo4j图数据库中存储的，XFUND数据集元数据的中心Table节点。
- 右侧9个褐色的节点，每一个节点对应元数据表格中的一列，节点名即为列名，他们与中心节点的s双向关系是[xfund]->COLOMN->[linking]与[linking]->COLUMN_OF->[xfund]。

- 中心节点左侧的红色节点是数据集的Tag节点，对应的是Amundsen前端显示数据集Tag的部分。同一个数据集可以有多个Tag，反馈在Neo4j后端即为多个与中心节点相连的tag节点。在前端可以为数据集添加tag，结果会同步到Neo4j中。

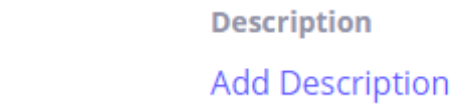
[Table]->Tagged_by->[Tag]

[Tag]->Tag->[Table]

- 中心节点左侧的绿色节点是User的信息。User与元数据Table之间由两种关系，第一种对应Amundsen前端Owners部分，表示数据集的所有者，该部分同样前端可编辑。（关系名为Owner,Owner_of）
- 第二种对应Amundsen前端对于数据集的收藏功能，收藏后的数据集会显示在Amundsen平台主页上。（关系名为Follow,Followed_by）

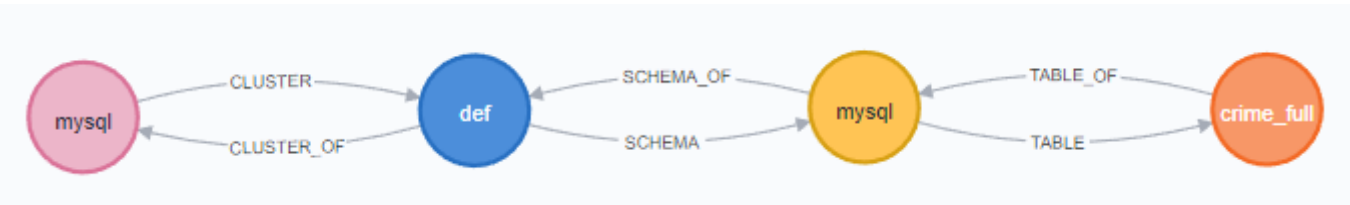


- 图上的蓝色节点是Description节点，该节点可能与数据集节点和列节点相连，对应amundsen前端Description部分，用于存储对于数据集/列的文字描述。

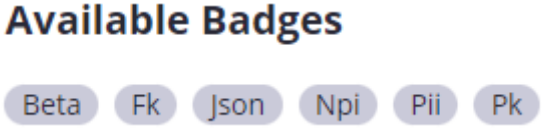


除了上图的基本结构之外，对于从不同数据库中提取得到的元数据表，存储结构如下图所示：

数据库名->集群（cluster）名->对象集合（schema）->表（table）名



- 除了Tag可用来标记数据集外，amundsen同时使用了Badge状态标记表示数据集的状态，每一种badge同样作为Neo4j图谱中的一个节点存储，与数据表之间有（Has_badge,Bagde_for）关系。



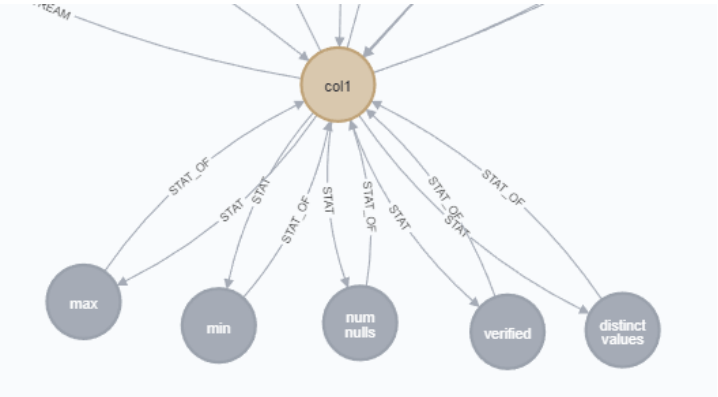
- Neo4j存储了数据集之间的上下游关系，但在目前的Amundsen前端并没有体现。

Index Processes

What: we want to index ETLs and pipelines from our Machine Learning Engine

Status: not planned

对于Amundsen示例元数据中提供的对于数据表中某一列的信息统计，前端显示如下，而在图数据库后端，每一行同样是一个Neo4j节点，与列节点之间有关系stat,stat_of。

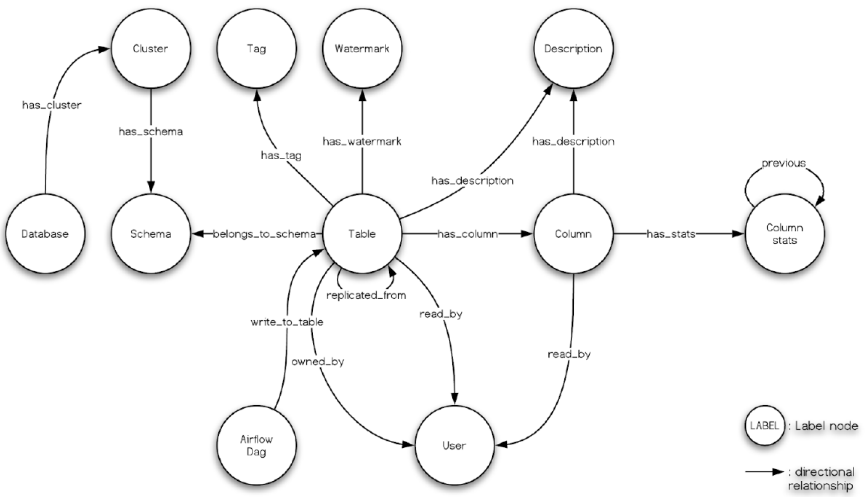


Column Statistics *Stats reflect data collected between May 22, 2015 and Jul 05, 2019.*

num nulls	"500320"
verified	"230430"
min	"aardvark"
distinct values	8
max	"zebra"

综上是非结构化数据提取结构化元数据后，元数据表存储在Neo4j图数据库中的方式。

Airflow Dag指的是开源workflow项目中的有向无环图（Directed Acyclic Graph），即支持从Airflow中导入整个工作流中使用的数据。

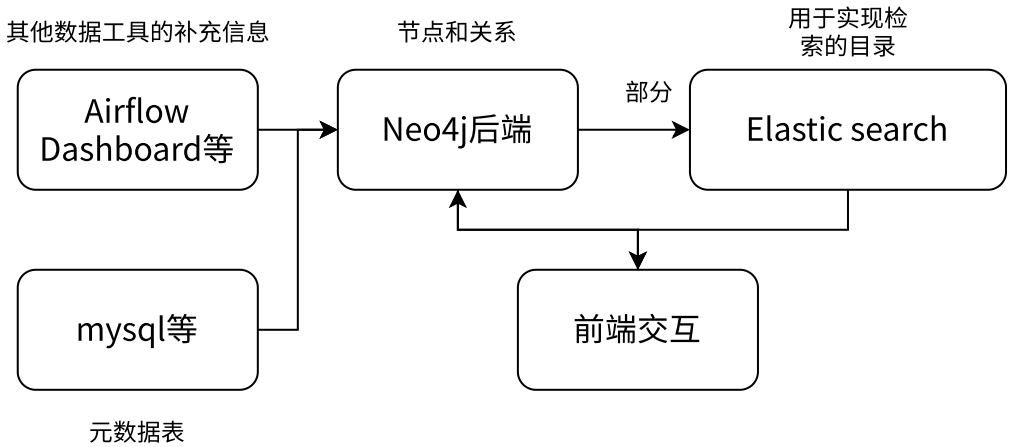


Neo4j后端与前端的交互

neo4j图数据库中使用其特有的Cypher语言进行检索。

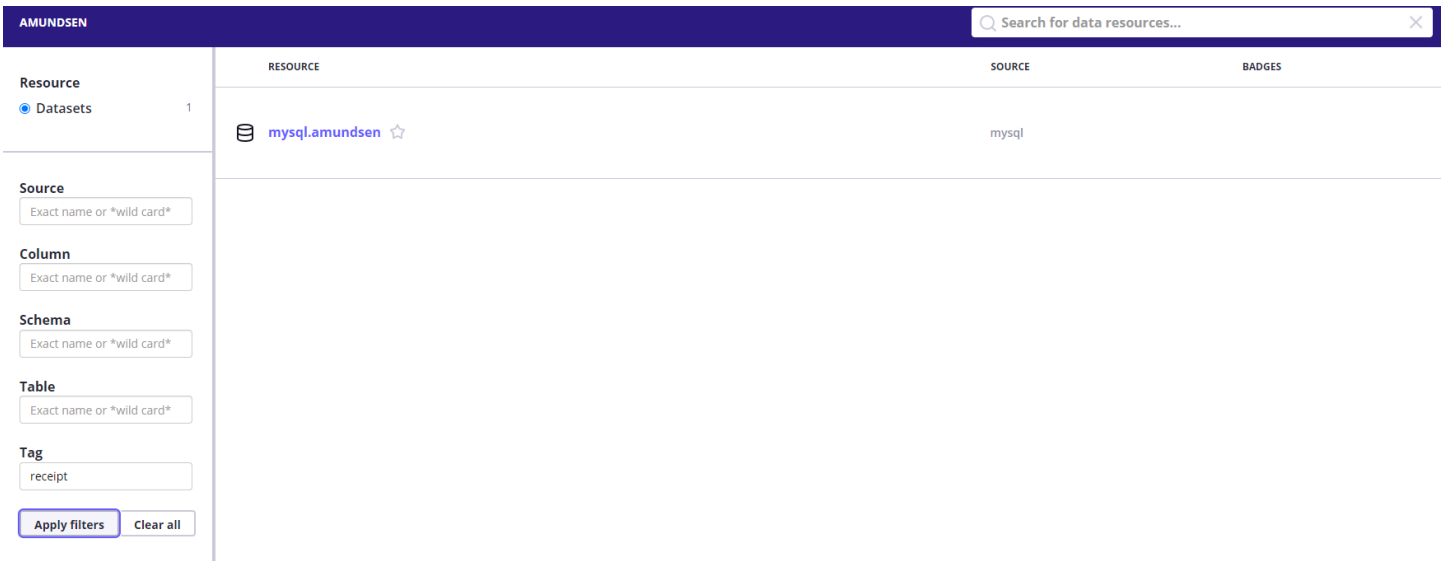
通过观察docker运行日志，以下介绍Neo4j，elastic search与前端的联动运作方式。

首先在从别的数据库向Amundsen引入数据集元数据时，工作流程如下：



即为了Elastic search检索功能所需Map均由Neo4j提供，由ElasticsearchPublisher脚本实现。

在使用前端检索时，不论是根据表格名称，列名称，数据库名称，数据库对象集合（schema）名称还是Tag名，都是使用Elastic search服务得到结果，即从输入检索关键字到反馈检索结果这一步，Neo4j完全不介入。



接下来，从检索结果页面离开回到主页或者点击查看数据集详情时，会触发Neo4jSearchDataExtractor脚本，即使用Cypher语言提取Neo4j图数据库，更新提供给Elastic search用于检索的内容。

```
amundsenmetadata    | DEBUG:metadata_service.proxy.neo4j_proxy:Executing Cypher query:
amundsenmetadata    | MATCH (source:Column {key: $query_key})
amundsenmetadata    | OPTIONAL MATCH dpath=(source)-[downstream_len:HAS_DOWNSTREAM*..1]->(downstream_entity:Column)
amundsenmetadata    | OPTIONAL MATCH upath=(source)-[upstream_len:HAS_UPSTREAM*..1]->(upstream_entity:Column)
amundsenmetadata    | WITH downstream_entity, upstream_entity, downstream_len, upstream_len, upath, dpath
amundsenmetadata    | OPTIONAL MATCH (upstream_entity)-[:HAS_BADGE]->(upstream_badge:Badge)
amundsenmetadata    | OPTIONAL MATCH (downstream_entity)-[:HAS_BADGE]->(downstream_badge:Badge)
amundsenmetadata    | WITH CASE WHEN downstream_badge IS NULL THEN []
amundsenmetadata    | ELSE collect(distinct {key:downstream_badge.key,category:downstream_badge.category})
amundsenmetadata    | END AS downstream_badges, CASE WHEN upstream_badge IS NULL THEN []
amundsenmetadata    | ELSE collect(distinct {key:upstream_badge.key,category:upstream_badge.category})
amundsenmetadata    | END AS upstream_badges, upstream_entity, downstream_entity, upstream_len, downstream_len, upath, dpath
amundsenmetadata    | OPTIONAL MATCH (downstream_entity:Column)-[downstream_read:READ_BY]->(:User)
amundsenmetadata    | WITH upstream_entity, downstream_entity, upstream_len, downstream_len, upath, dpath,
amundsenmetadata    | downstream_badges, upstream_badges, sum(downstream_read.read_count) as downstream_read_count
amundsenmetadata    | OPTIONAL MATCH (upstream_entity:Column)-[upstream_read:READ_BY]->(:User)
amundsenmetadata    | WITH upstream_entity, downstream_entity, upstream_len, downstream_len,
amundsenmetadata    | downstream_badges, upstream_badges, downstream_read_count,
amundsenmetadata    | sum(upstream_read.read_count) as upstream_read_count, upath, dpath
amundsenmetadata    | WITH CASE WHEN upstream_len IS NULL THEN []
amundsenmetadata    | ELSE COLLECT(distinct{level:SIZE(upstream_len), source:split(upstream_entity.key,'://')[0],
amundsenmetadata    | key:upstream_entity.key, badges:upstream_badges, usage:upstream_read_count, parent:nodes(upath)[-2].key})
amundsenmetadata    | END AS upstream_entities, CASE WHEN downstream_len IS NULL THEN []
amundsenmetadata    | ELSE COLLECT(distinct{level:SIZE(downstream_len), source:split(downstream_entity.key,'://')[0],
amundsenmetadata    | key:downstream_entity.key, badges:downstream_badges, usage:downstream_read_count, parent:nodes(dpath)[-2].key})
```

另外，当使用者在Amundsen前端对数据集添加Tag或Badge标签，对数据表和列添加描述description或是添加数据集使用者等信息，这些内容均会反馈到Neo4j图数据库中，添加新的对应节点，并在页面刷新后反馈至前端和通过上述两个脚本将更新同步到Elastic search。