

Install + basic

Useful Link

- Slack <https://openmetadata.slack.com/ssb/redirect>
- Github <https://github.com/open-metadata>
- Document <https://docs.open-metadata.org/latest>
- Api document <https://docs.open-metadata.org/swagger.html#section/APIs>
- user's manual <https://docs.open-metadata.org/latest/how-to-guides>

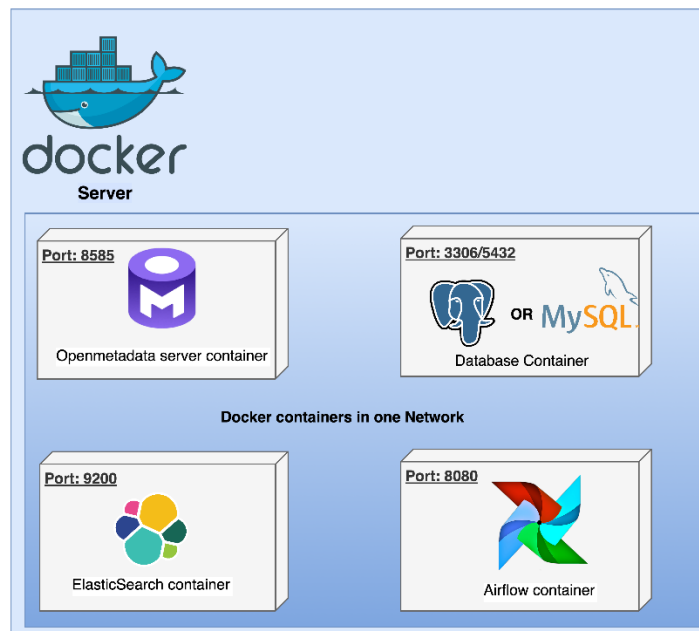


Fig: Docker Deployment Architecture

การติดตั้ง OpenMetadata ด้วย Docker

- ดาวน์โหลดไฟล์ docker-compose.yaml
- <https://github.com/open-metadata/OpenMetadata/releases>

- docker-compose up -d

ติดตั้ง Python SDK

pip install openmetadata-ingestion

ติดตั้ง Airflow Plugin

- ดาวน์โหลด airflow-provider_openmetadata
- <https://github.com/open-metadata/OpenMetadata/tree/main/ingestion/src>
- วางไว้ในที่ที่สามารถ reference จาก airflow ได้ง่าย

การเข้าใช้งานผ่านหน้าเว็บไซต์

- Default ที่ <http://localhost:8585>
- Admin default login: Email: admin@open-metadata.org Password: admin
- สร้าง Admin คนแรกที่ <http://localhost:8585/settings/members/admins> (ต้องลบ default admin ออกเอง)

การใช้งาน SDK

- Sdk จะประกอบด้วย class OpenMetadata ที่เก็บข้อมูล connection และมีฟังก์ชันให้ใช้งานมากมาย
- มี class ของ data asset และส่วนประกอบของ data asset
- แต่ละ data asset ก็จะมีคลาส Create__Request อยู่ สำหรับใช้สร้างข้อมูล
- ข้อมูล connection จะต้องใช้ jwt_token ของ ingestion bot ในการใช้งาน ซึ่งได้จาก <http://localhost:8585/bots/ingestion-bot>

ตัวอย่าง

```
server_config = OpenMetadataConnection(
    hostPort="http://localhost:8585/api",
    authProvider=AuthProvider.openmetadata,
    securityConfig=OpenMetadataJWTClientConfig(
        jwtToken=<token>
    ),
)
connection_obj = OpenMetadata(server_config)
```

การใช้งาน โค้ดที่เขียน

- แก้ Config เกี่ยวกับ connection ถึง OpenMetadata/ mssql
- ข้อมูลเขียนใน ./Data/
- ในโฟลเดอร์ Script จะรวมฟังก์ชันสำหรับสร้าง metadata ต่าง ๆ
- โฟลเดอร์ Exe คือตัวอย่างที่ใช้สร้าง demo แสดงการใช้ฟังก์ชัน
 - _1 สร้าง tag glossary policy role team user service dashboard api
 - _3 เพิ่มข้อมูล lineage, sample, ใส่ tag ให้ column

สร้าง/ดึงข้อมูล

Business glossary

ผ่านเว็บไซต์

<http://localhost:8585/glossary/>

ใช้ Library

สร้าง obj ของคลาส CreateGlossaryRequest, CreateGlossaryTermRequest

และส่ง request ด้วย create_or_update() ของ connection obj

โค้ดที่เขียนไว้

Script.classification.create_glossary_term_from_file

Tags

ผ่านเว็บไซต์

<http://localhost:8585/tags/>

ใช้ Library

สร้าง obj ของคลาส CreateClassificationRequest, CreateTagRequest

และส่ง request ด้วย create_or_update() ของ connection obj

โค้ดที่เขียนไว้

Script.classification.create_classification_tag_from_yaml()

การสร้าง Service

ผ่านหน้าเว็บไซต์

<http://localhost:8585/settings/services>

เลือกประเภทของ Service ตามต้องการ

กดปุ่ม Add New Service

เลือก Connector ของ software/platform ที่ต้องการ

กรอกรายละเอียด ชื่อที่จะใช้ใน OpenMetadata

กรอกข้อมูลการเชื่อมต่อ ตามที่แต่ละ connector ต้องการ

ใช้ Library

สร้าง object ของคลาส Create____ServiceRequest เช่น CreateDatabaseServiceRequest
CreatePipelineServiceRequest และใช้ create_or_update ของ connection object ส่ง request สร้าง
metadata

โค้ดที่เขียนไว้

อยู่ในไฟล์ของ entity ชนิดต่าง ๆ

```
Script.database.create_mssql_service(), Script.pipeline.create_airflow_service(),  
Script.file.create_storage_service()
```

การ ingest ข้อมูล mssql

ผ่านหน้าเว็บไซต์

<http://localhost:8585/service/databaseServices/DWH/add-ingestion/metadata>

กรอกข้อมูล filter database schema table

ตั้งกำหนดการทำซ้ำ -> add & deploy

ถ้าไม่ได้ตั้งกำหนดการทำซ้ำไว้จะต้อง กด run เอง

<http://localhost:8585/service/databaseServices/DWH/ingestions>

ใช้ Library

ไม่ได้ศึกษา

โค้ดที่เขียนไว้

ไม่ได้เขียน

การ ingest ข้อมูล airflow

ผ่านหน้าเว็บไซต์

สำหรับ airflow ก็สามารถสร้าง ingest ผ่าน UI ได้เช่นกัน แต่ต้องใช้รายละเอียด database connection ของ airflow (ที่เก็บข้อมูลเกี่ยวกับ dag) เลยไปใช้ airflow plugin แทน

ใช้ Library

ไม่ได้ศึกษา

โค้ดที่เขียนไว้

ไม่ได้เขียน

Airflow Plugin

```
1172
1173 [lineage]
1174 # what lineage backend to use
1175 #
1176 # Variable: AIRFLOW__LINEAGE__BACKEND
1177 #
1178 backend = script.airflow_provider_openmetadata.lineage.backend.OpenMetadataLineageBackend
1179 airflow_service_name = local_airflow
1180 openmetadata_api_endpoint = http://host.docker.internal:8585/api
1181 jwt_token = <token_from_web>
```

- Backend ที่ไปยัง OpenMetadataLineageBackend
- ใช้ service name ที่ตั้งในขั้นตอนสร้าง service

- Endpoint <http://host.docker.internal:8585/api>
- Token <http://localhost:8585/bots/ingestion-bot>

หรือสร้าง Task จากคลาส OpenMetadataLineageOperator จาก plugin ที่ติดตั้ง

```

10 from airflow_provider_openmetadata.lineage.operator import OpenMetadataLineageOperator
11 from airflow_provider_openmetadata.hooks.openmetadata import OpenMetadataHook
12
13 openmetadata_hook = OpenMetadataHook(openmetadata_conn_id="openmetadata") # The ID you provided
14 server_config = openmetadata_hook.get_conn()

```

```

76 opm = OpenMetadataLineageOperator(
77     task_id='lineage_op',
78     depends_on_past=False,
79     server_config=server_config,
80     service_name="local_airflow",
81 )
82 eod_operator >> opm

```

จากทั้งสองวิธีเมื่อ DAG ทำงานก็จะส่งข้อมูลไปที่ Openmetadata เอง

Sample data

ผ่านหน้าเว็บไซต์

<http://localhost:8585/service/databaseServices/DWH/add-ingestion/profiler>

จะสร้าง Ingestion Pipeline ที่จะดึงข้อมูลมา และมีการวัดค่าต่าง ๆ เกี่ยวกับตารางหรือคอลัมน์ เช่น

Null% unique% ... และข้อมูลที่ดึงมาก็จะมีแสดงในหน้า entity นั้น ๆ (ยังไม่แน่ใจว่า ค่าที่วัด เอามาจากทั้งหมดหรือ เฉพาะที่แสดง)

ใช้ Library

สร้าง object ของคลาส TableData และใช้ ingest_table_sample_data ของ connection object

ส่ง request

โค้ดที่เขียนไว้

```
Script.sample_data.put_sample_data_from_csv()
```

ข้อมูล data lineage

ผ่านหน้าเว็บไซต์

- สามารถไปที่ entity เช่น table, api endpoint, pipeline, dashboard
- ที่ tab lineage ตัวอย่างเช่น <http://localhost:8585/table/DWH.SetDB.dbo.C2WAccount/lineage>
- จะมีปุ่ม edit lineage ขวบน และสามารถลากวาง entity และลากเส้นเชื่อมได้ สามารถระบุรายละเอียดของเส้นเชื่อมได้ด้วย เช่น ระบุ pipeline, คำอธิบาย

ใช้ Library

- ต้องใช้คลาส AddLineageRequest
- ส่ง request ด้วย add_lineage() ซึ่งแตกต่างจาก entity อื่น ๆ ที่ใช้ create_or_update() ได้

โค้ดที่เขียนไว้

```
Script.lineage.add_lineage_from_yaml()
```


DAG's inlet outlet

- ระบุ inlets=[] และ outlets=[] แต่ละ Task
- ใช้คลาส OMEntity (ของ SDK)
- ใช้ key เพื่อแยกการเชื่อมโยงแต่ละเส้น
- ระบุระดับคอลัมน์ไม่ได้
- จะเข้าพร้อมกับข้อมูลของ DAG

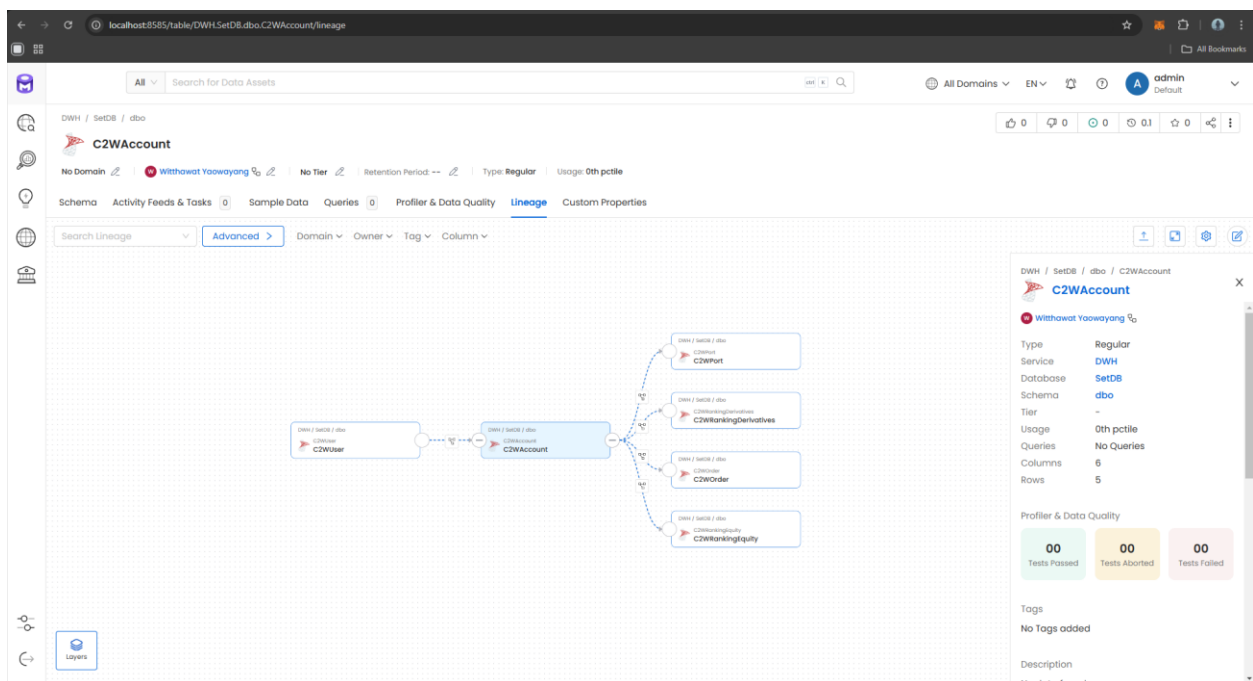
```
79 fund_statistic_operator = PythonOperator(  
80     task_id= 'fund_statistic',  
81     python_callable=pvd_fund_statistic.delete_insert_fund_statistic,  
82     inlets=[  
83         OMEntity(entity=Table, fqcn="DWH.SetDB.dbo.PVDFundProfile",key="lookup fund_id"),  
84     ],  
85     outlets=[  
86         OMEntity(entity=Table, fqcn="DWH.SetDB.dbo.PVDFundStatistics",key="lookup fund_id"),  
87     ],  
88 )
```

Other

การดูข้อมูล Data Lineage

Entity ที่สามารถมี lineage ได้ ก็สามารถดูจาก tab Lineage และสามารถดูรายละเอียด lineage แต่ละเส้น หรือ ดูรายละเอียด data asset ที่อยู่ใน lineage ได้

<http://localhost:8585/table/DWH.SetDB.dbo.C2WAccount/lineage>



Backup & Restore

Backup

จะใช้การ Logical Backup (เขียน SQL ที่ สร้างตารางและinsert เข้าให้) โดยไปที่ container mysql แล้วใช้ mysqldump ลองใช้ในบางคำสั่งแล้วต่อ .sql มีขนาดใหญ่ เลยทดลองไม่สมบูรณ์

create back up file

(exec inside mysql container)

```
mysqldump -u root -p'password' --all-databases > all_db_backup.sql
mysqldump -u root -p --all-databases > all_db_backup.sql
```

copy to desired location

(exec on host)

```
docker cp openmetadata_mysql:/docker-entrypoint-initdb.d/all_db_backup.sql
D:/SET/metadata_project/all_db_backup.sql
```

Restore

ก็สั่งรัน SQL ได้เลย

copy to mysql container

```
docker cp D:/SET/metadata_project/all_db_backup.sql
openmetadata_mysql:/docker-entrypoint-initdb.d/all_db_backup.sql
```

execute

```
mysql -u root -p'password' < /docker-entrypoint-initdb.d/all_db_backup.sql
```

การทำงานของ Ingestion pipeline


ตรวจสอบการทำงานของ ingestion pipeline ให้ไปที่ airflow (ของ openmetadata)

include ddl

option ในขั้นตอนนี้ ingest database

ได้ tab Schema definition เพิ่มมา

DWH / SetDB / dbo

 **C2WSocialUser**

No Domain [🔗](#) | No Owner [🔗](#) | No Tier [🔗](#) | Retention Period: -- [🔗](#) | Type: Regular | Usage: 0th pctile

Schema Activity Feeds & Tasks **5** Sample Data Queries **0** Profiler & Data Quality Lineage **Schema Definition** Cu

```

1 CREATE TABLE dbo."C2WSocialUser" (
2   "Member_Id" VARCHAR(20) COLLATE "Thai_CI_AS" NOT NULL,
3   "Social_Username" VARCHAR(20) COLLATE "Thai_CI_AS" NOT NULL,
4   "Create_Date" DATETIME,
5   "Email" VARCHAR(255) COLLATE "Thai_CI_AS",
6   "Social_Type" VARCHAR(255) COLLATE "Thai_CI_AS",
7   CONSTRAINT "PK_C2WSocialUser" PRIMARY KEY ("Member_Id")
8 )

```

Process Pii Sensitive

ในหน้าแรกของการสร้าง profiler pipeline มี option นี้ OpenMetadata อธิบายไว้ดังนี้

Auto Tag PII

Set the Auto Tag PII toggle to control whether to automatically tag columns that might contain sensitive information as part of profiler ingestion.

If Ingest Sample Data is enabled, OpenMetadata will leverage machine learning to infer which column may contain PII sensitive data. If disabled, OpenMetadata will infer this information from the column name. Use the Confidence setting in the "DatabaseServiceProfilerPipeline Advanced Config" to set the confidence level when inferring the PII status of a column.

ผลลัพธ์คือข้อมูล col จะมีติด tag Sensitive , NonSensitive และบาง col ไม่มีอะไรติด (id บางตารางเป็นบางตารางไม่เป็น ไม่แน่ใจกลไกการจำแนก)

สามารถใส่ Sample data ผ่านฟังก์ชัน `ingest_table_sample_data()` ของ `sdk` โดยจะแทนที่ข้อมูล sample ที่มีอยู่

Theme + Icon

สี และไอคอน <http://localhost:8585/settings/preferences/appearance>

Access control

สร้าง `policy` ที่สามารถมอบให้กับ ทีม หรือ `role` ได้

`policy` ประกอบด้วย `rule` ที่ระบุ `Resources`(db, pipeline ...) `Operations`(view, edit, ...) `Effect` (allow / deny) `Condition`(ใช้ฟังก์ชันที่มีประกอบ `expression` ขึ้นมา เช่น `isOwner()` | `matchAnyTag('access.tag1')`)

<http://localhost:8585/settings/access/policies>