COMMUNITY
THE ASF CONFERENCE
CODE

bilibili

Apache Gravitino 在B站的最佳实践

李天航 / Bilibili 大数据开发工程师

COMMUNITY
THE ASF CONFERENCE
CODE

bilibili

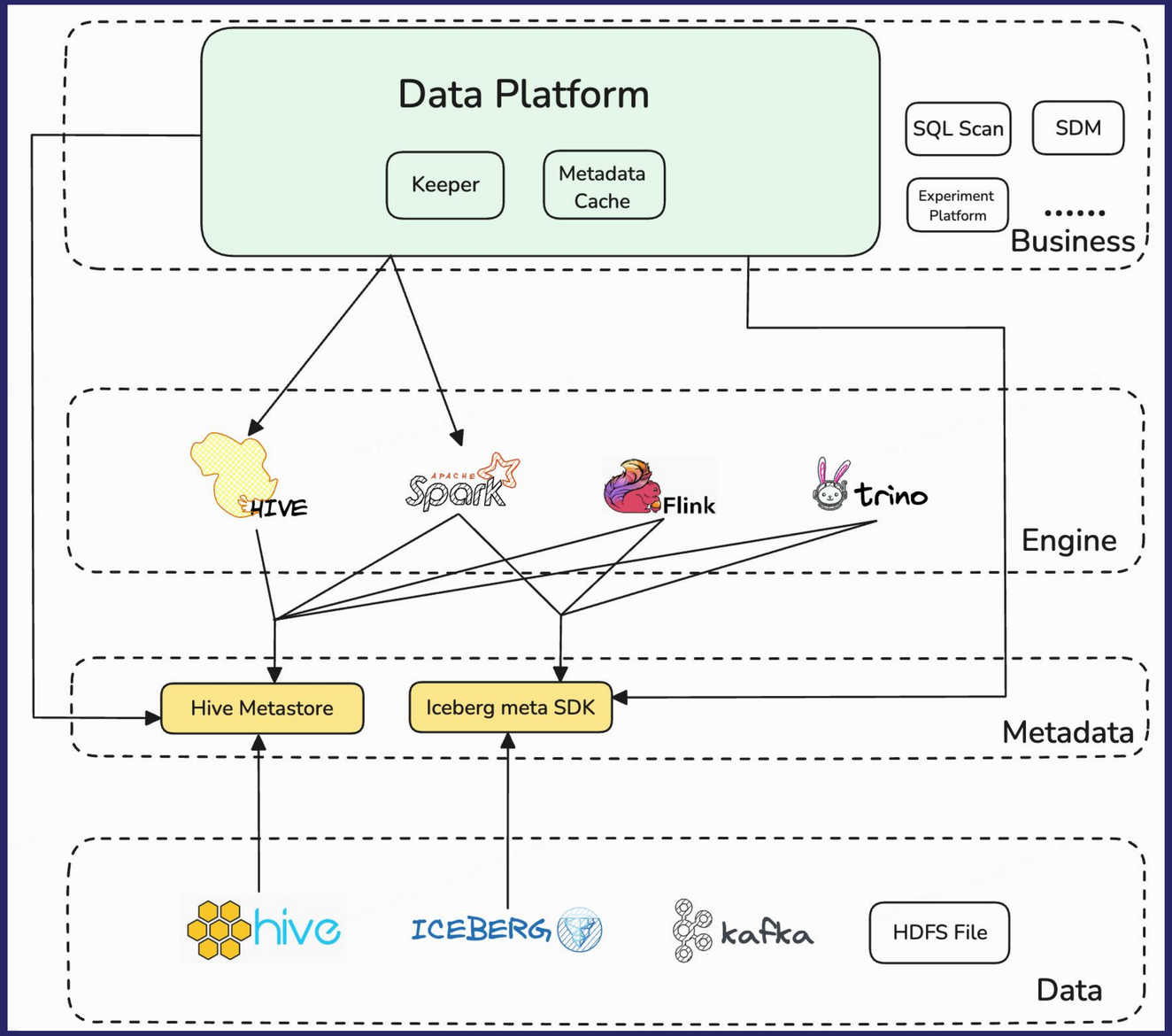
CONTENTS

1. 元数据管理痛点剖析
2. Apache Gravitino 背景概览
3. Apache Gravitino 生产实践
4. Apache Gravitino 规划展望

01

元数据管理痛点剖析

元数据管理的痛点



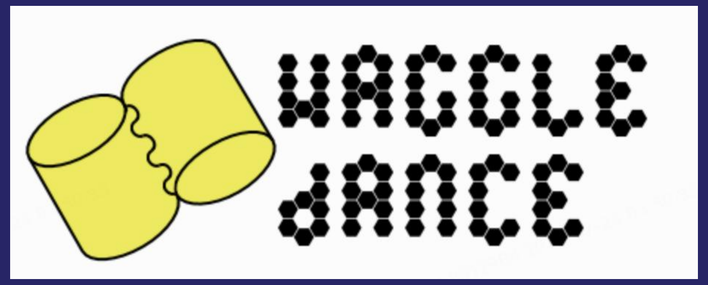
- 业务侧耦合度高：元数据使用方调用异构数据源方式多种多样
- 数据治理能力有限：无法提供统一的审计、权限管理、TTL能力
- 半结构化/非结构化数据源缺乏管理
- 跨源数据 Schema 维护成本高

02
Apache Gravitino 背景概览

元数据管理中心化

- ✓ 解耦元数据使用方与各类组件，降低元数据使用成本
- ✓ 跨源数据统一维护 Schema，降低跨源数据使用成本
- ✓ 提供统一的审计、TTL特性，HDFS EC 通过数据治理实现降本
- ✓ AI 数据资产集中数据治理
- ✓ 建立统一权限管控机制，提高数据安全性

元数据管理组件对比



- 多引擎支持有限

1. 支持的引擎种类有限
2. 引擎提供的API不完善

- 功能较为单一

1. 支持数据种类有限，无法管理非结构化数据，例如 Fileset 等
2. 只提供元数据基本功能，难以扩展权限管理等能力

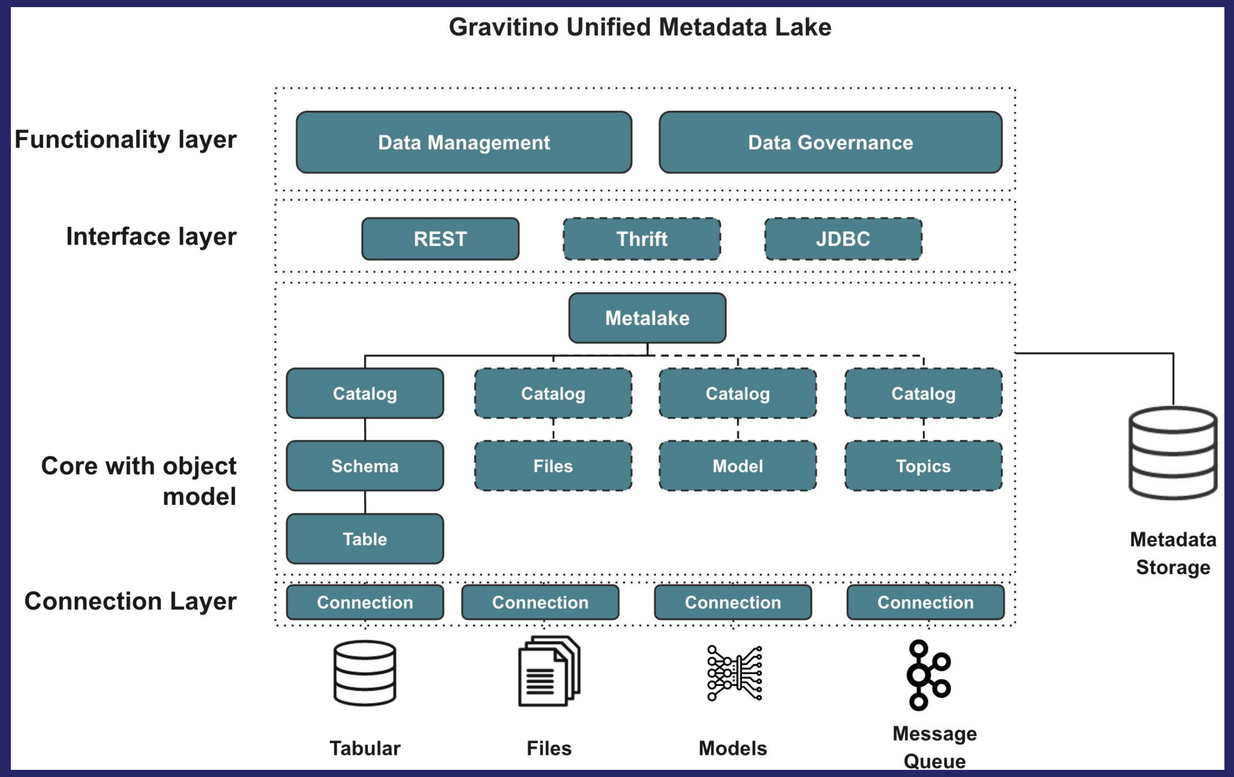
- 社区活跃度有限

1. 社区活跃度低
2. 功能特性迭代缓慢

What is Gravitino ?

Gravitino is a high-performance, geo-distributed, and federated metadata lake. It manages the metadata directly in different sources, types, and regions. It also provides users with unified metadata access for data and AI assets.

Gravitino Architecture

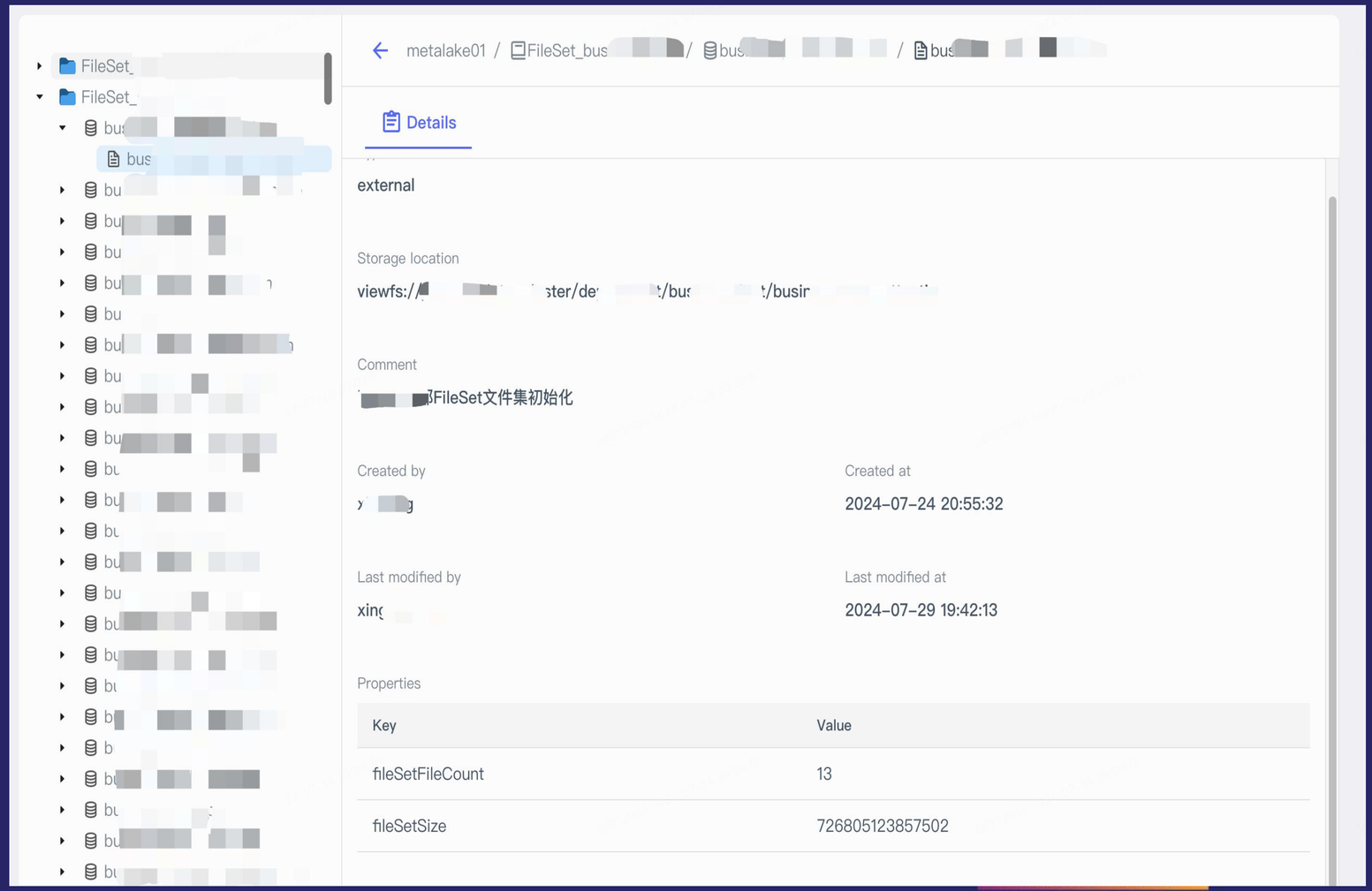


- ✓ 提供多种数据源统一的元数据视图
- ✓ 支持统一的权限管理等特性
- ✓ 支持多种数据源，包括半/非结构化数据源
- ✓ 支持多种计算引擎
- ✓ 开源社区活跃度高

Gravitino FileSet

FileSet：一组文件和目录的集合

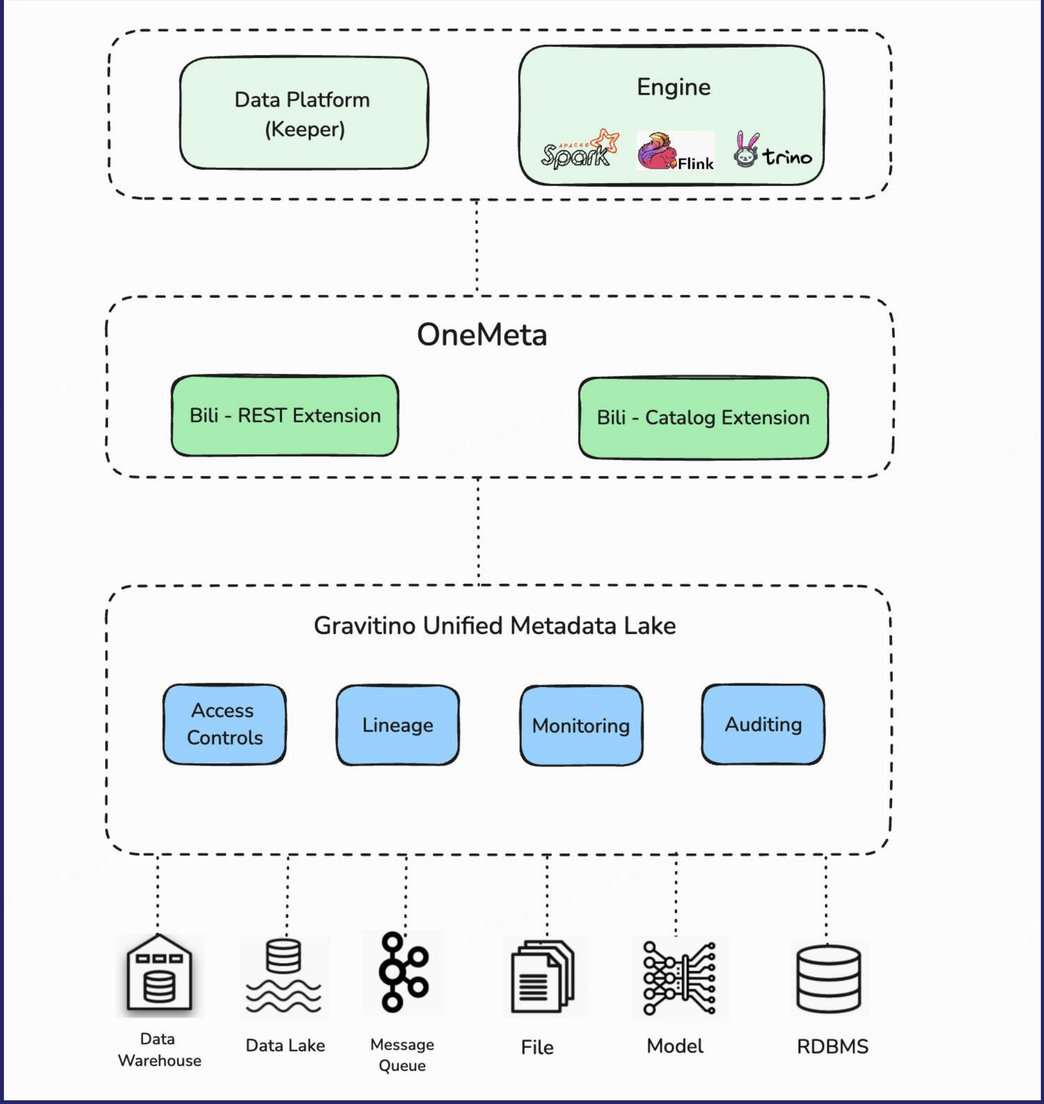
- 用户使用 FileSet 管理非表格数据
- Fileset 映射到像HDFS等文件系统的一个目录上
- 通过 Gravitino 管理的 FileSet 将非表格数据与表格数据一起以统一的方式作为资产进行管理
- 用户通过 FileSet 轻松访问和管理文件/目录，无需获取管理数据集的物理路径



03
Apache Gravitino 生产实践

COMMUNITY
THE ASF CONFERENCE
CODE

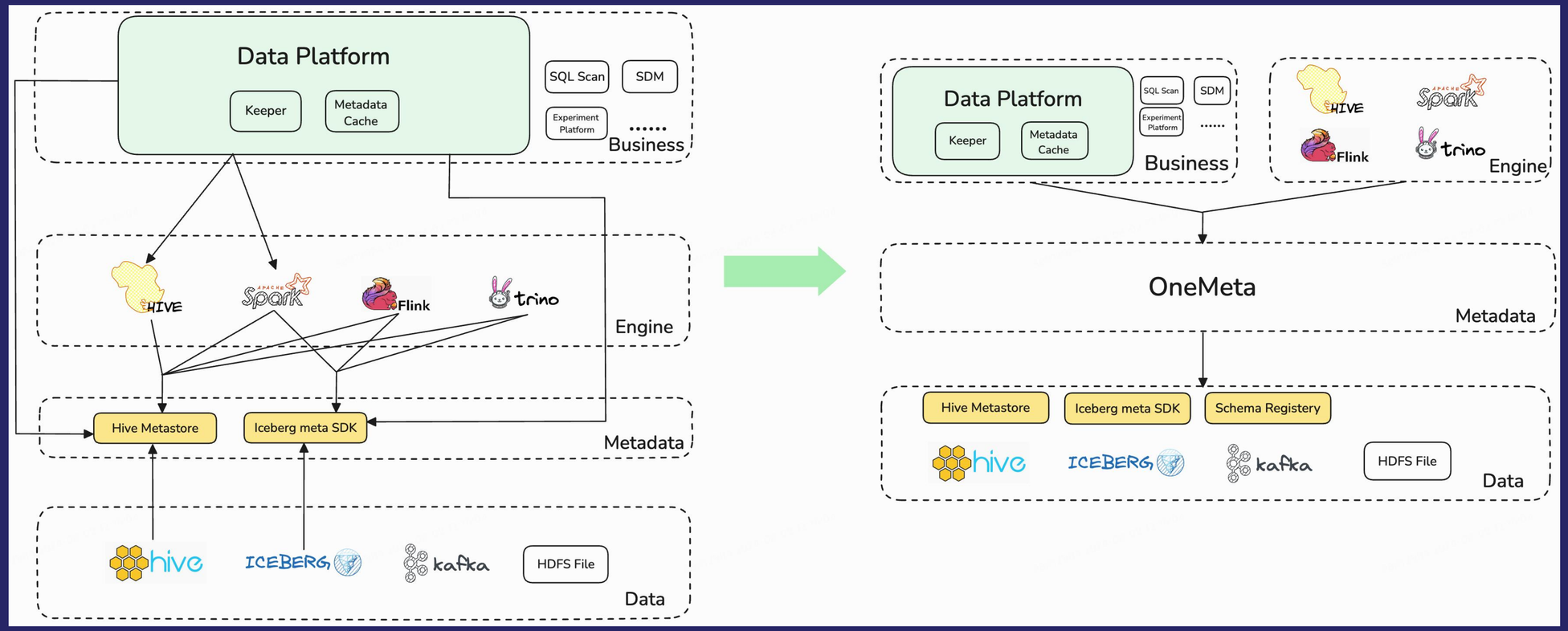
OneMeta：统一的元数据管理服务



OneMeta：Extension 集成 Gravitino

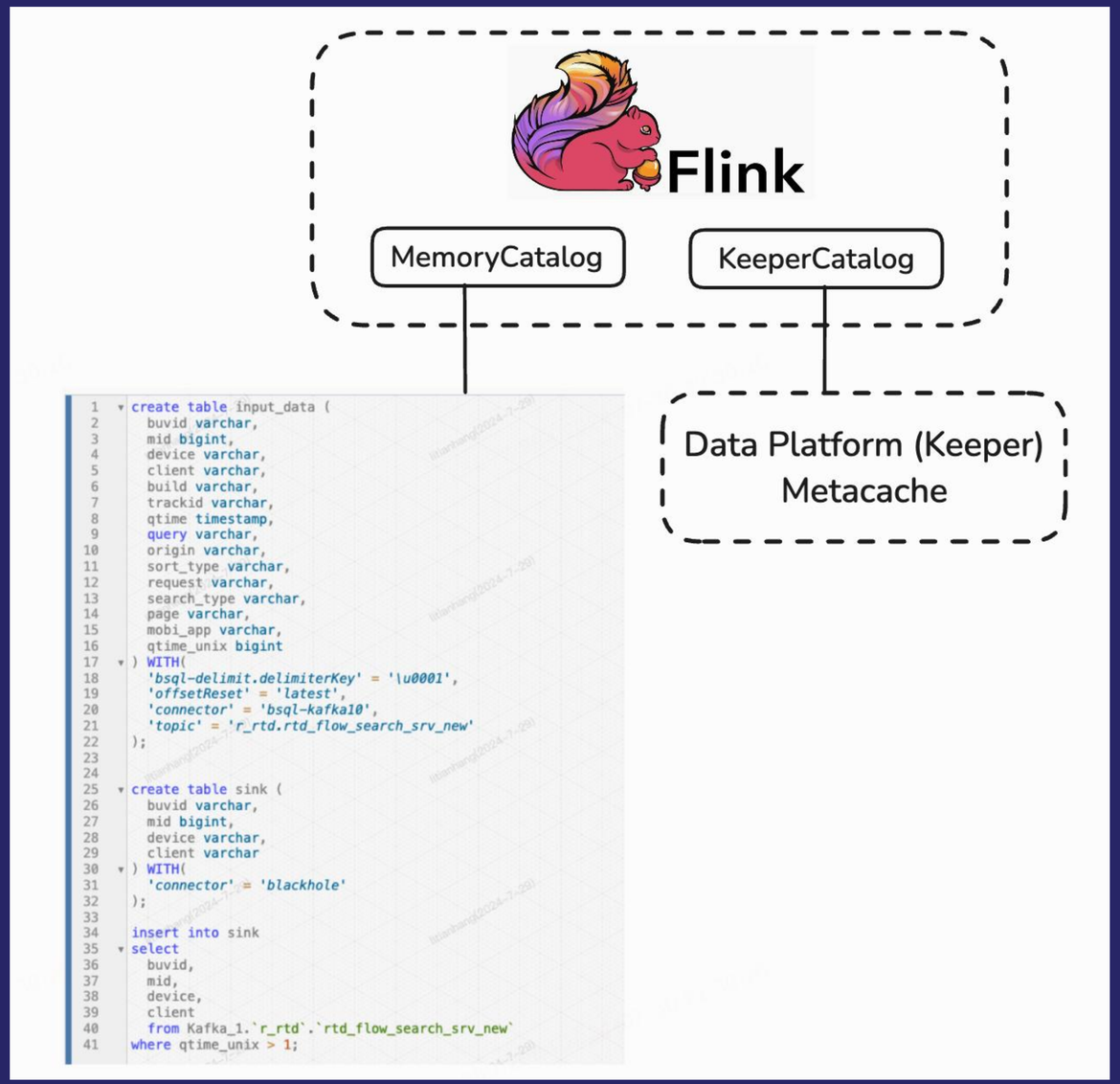
- ✓ 提供定制化接口：dropPartitiosByFilter / loadFileDetail / loadFiles .....
- ✓ 提供定制化 catalog 实现：BiliIcebergCatalog / BiliKafkaCatalog / BiliDatabus Catalog .....
- ✓ 降低代码的侵入性，便于同步社区最新代码

元数据管理架构演进



- ✓ 解耦业务方复杂依赖，降低元数据使用成本
- ✓ 解决由于引擎间差异、数据源差异造成的元数据不一致问题
- ✓ 解决由于 Hive MetaStore 造成的性能瓶颈
  1. dropTable：分区数过多时无法删除
  2. getTable：用户对 HMS client 的使用方式存在问题

跨源数据 Schema 管理现状



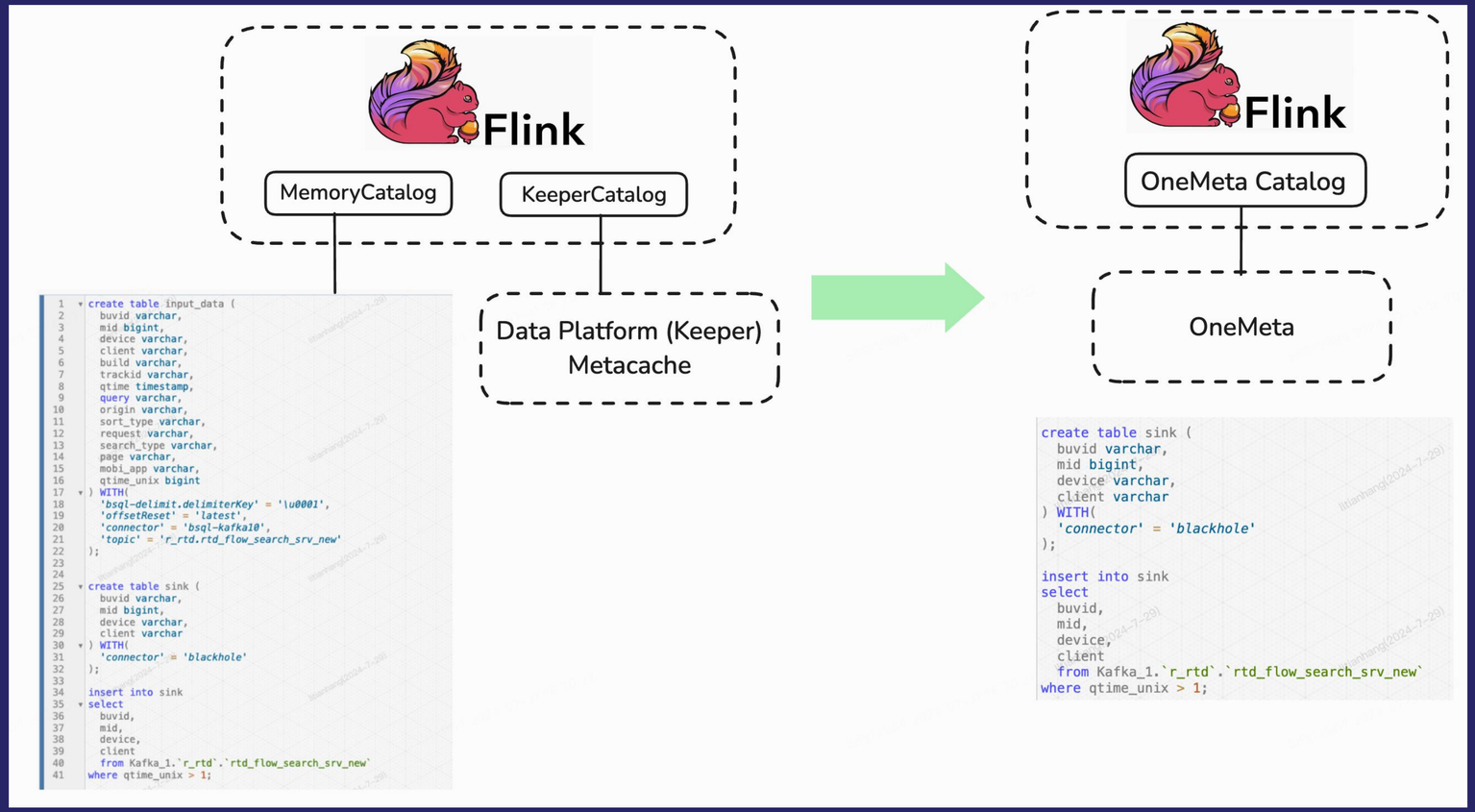

```
create table input_data (
  buvid varchar,
  mid bigint,
  device varchar,
  client varchar,
  build varchar,
  trackid varchar,
  qtime timestamp,
  query varchar,
  origin varchar,
  sort_type varchar,
  request varchar,
  search_type varchar,
  page varchar,
  mobi_app varchar,
  qtime_unix bigint
) WITH(
  'bsql-delimit.delimiterKey' = '\u0001',
  'offsetReset' = 'latest',
  'connector' = 'bsql-kafka10',
  'topic' = 'r_rtd.rtd_flow_search_srv_new'
);


create table sink (
  buvid varchar,
  mid bigint,
  device varchar,
  client varchar
) WITH(
  'connector' = 'blackhole'
);

insert into sink
select
  buvid,
  mid,
  device,
  client
  from Kafka_1.`r_rtd`.`rtd_flow_search_srv_new`
where qtime_unix > 1;
```

Flink Catalog 管理跨源数据 Schema：

Flink Keeper Catalog & Memory Catalog：处理 Catalog 时根据 SQL 判断此 SQL 对应的 Catalog

1. 如果用户的是自己的 DDL 语句，则使用memory-catalog，从内存中加载用户给到的 Schema
2. 如果是三段式的结构，则会使用，则使用keeper-catalog，从 keeper 侧加载对应数据源的 Schema

跨源数据 Schema 管理现状




```
create table input_data (
  buvid varchar,
  mid bigint,
  device varchar,
  client varchar,
  build varchar,
  trackid varchar,
  qtime timestamp,
  query varchar,
  origin varchar,
  sort_type varchar,
  request varchar,
  search_type varchar,
  page varchar,
  mobi_app varchar,
  qtime_unix bigint
) WITH(
  'bsql-delimit.delimiterKey' = '\u0001',
  'offsetReset' = 'latest',
  'connector' = 'bsql-kafka10',
  'topic' = 'r_rtd.rtd_flow_search_srv_new'
);

create table sink (
  buvid varchar,
  mid bigint,
  device varchar,
  client varchar
) WITH(
  'connector' = 'blackhole'
);

insert into sink
select
  buvid,
  mid,
  device,
  client
  from Kafka_1.`r_rtd`.`rtd_flow_search_srv_new`
where qtime_unix > 1;
```

```
create table sink (
  buvid varchar,
  mid bigint,
  device varchar,
  client varchar
) WITH(
  'connector' = 'blackhole'
);

insert into sink
select
  buvid,
  mid,
  device,
  client
  from Kafka_1.`r_rtd`.`rtd_flow_search_srv_new`
where qtime_unix > 1;
```

Flink Catalog 管理跨源 Schema 缺陷

1. 对于用户手写 DDL 方式，需要注明 Kafka 的集群、链接，schema，维护成本较高，容易出现上下游 schema 不对齐，导致无法成功运行

2. 对于直接注明三段式的形式，则出现基架反向依赖业务组件；且业务存储的 schema 信息不一定可靠

跨源 Schema 管理：去 DDL

```sql
create table input_data (
  buvid varchar,
  mid bigint,
  device varchar,
  client varchar,
  build varchar,
  trackid varchar,
  qtime timestamp,
  query varchar,
  origin varchar,
  sort_type varchar,
  request varchar,
  search_type varchar,
  page varchar,
  mobi_app varchar,
  qtime_unix bigint
) WITH(
  'bsql-delimit.delimiterKey' = '\u0001',
  'offsetReset' = 'latest',
  'connector' = 'bsql-kafka10',
  'topic' = 'r_rtd.rtd_flow_search_srv_new'
);


create table sink (
  buvid varchar,
  mid bigint,
  device varchar,
  client varchar
) WITH(
  'connector' = 'blackhole'
);

insert into sink
select
  buvid,
  mid,
  device,
  client
  from Kafka_1.`r_rtd`.`rtd_flow_search_srv_new`
where qtime_unix > 1;
```

```sql
create table sink (
  buvid varchar,
  mid bigint,
  device varchar,
  client varchar
) WITH(
  'connector' = 'blackhole'
);

insert into sink
select
  buvid,
  mid,
  device,
  client
  from Kafka_1.`r_rtd`.`rtd_flow_search_srv_new`
where qtime_unix > 1;
```

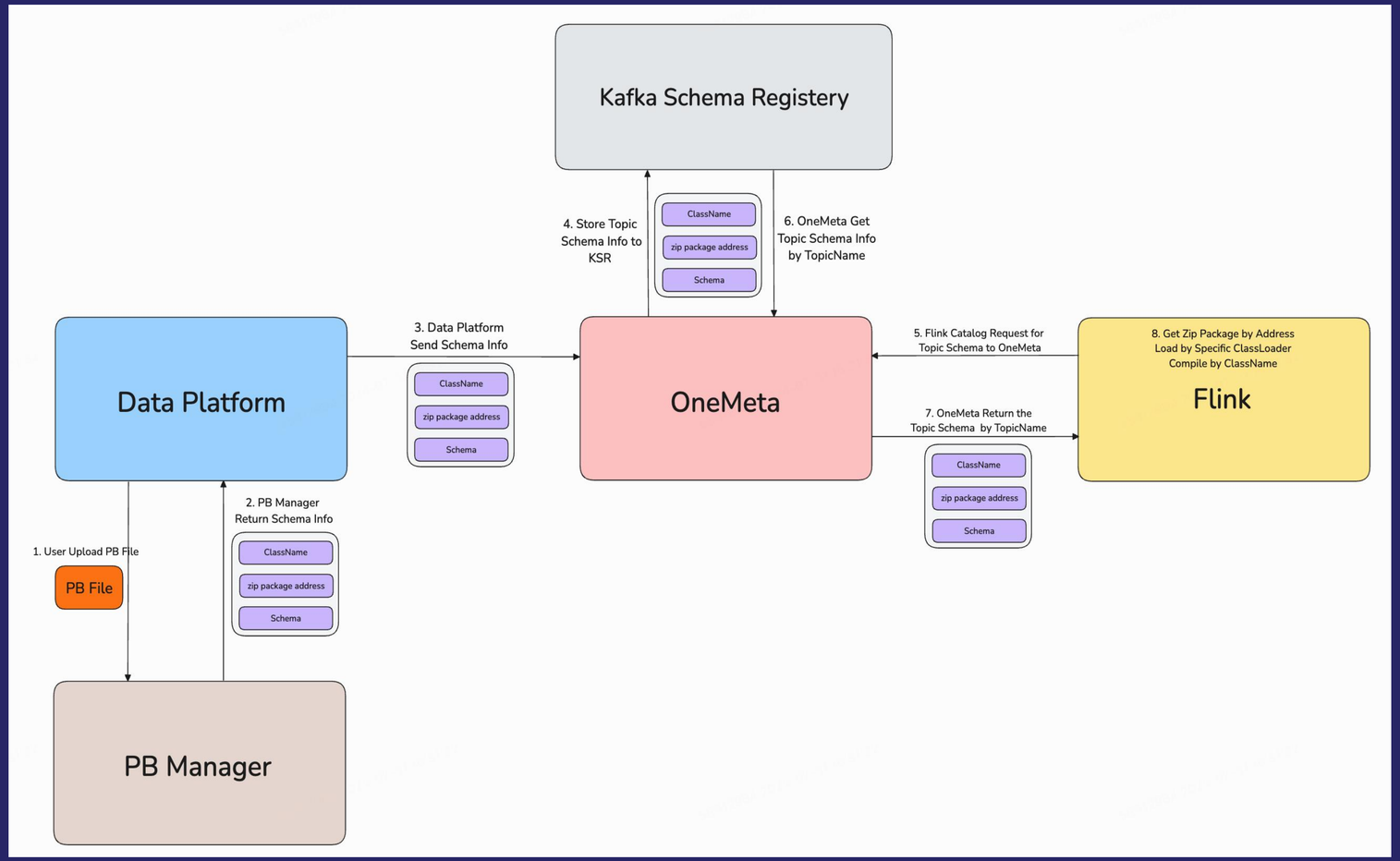
跨源 Schema 管理优化

Flink Catalog 管理跨源数据 Schema

Kafka 消息格式支持 JSON /分隔符/ PB

以 PB 为例 -- 基于 PB 打包编译全闭环的跨源 schema 管理

- 用户上传 PB 文件至 PB Manager 编译，产物信息传给 OneMeta；OneMeta 则存储至 KSR
- Flink 向 OneMeta 获取 Schema 信息；OneMeta 从 KSR 加载并返回给 Flink
- Flink 使用类加载器进行解析编译好的 Schema

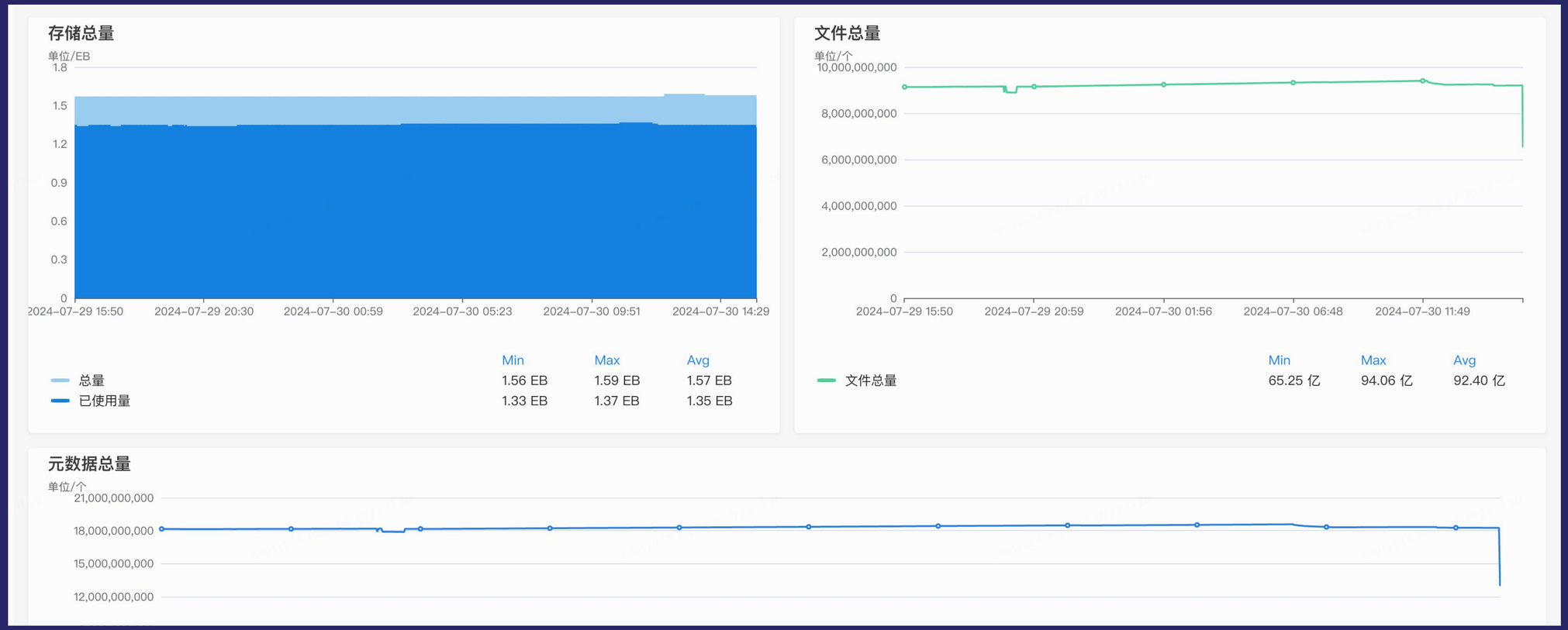


跨源 Schema 管理优化 收益

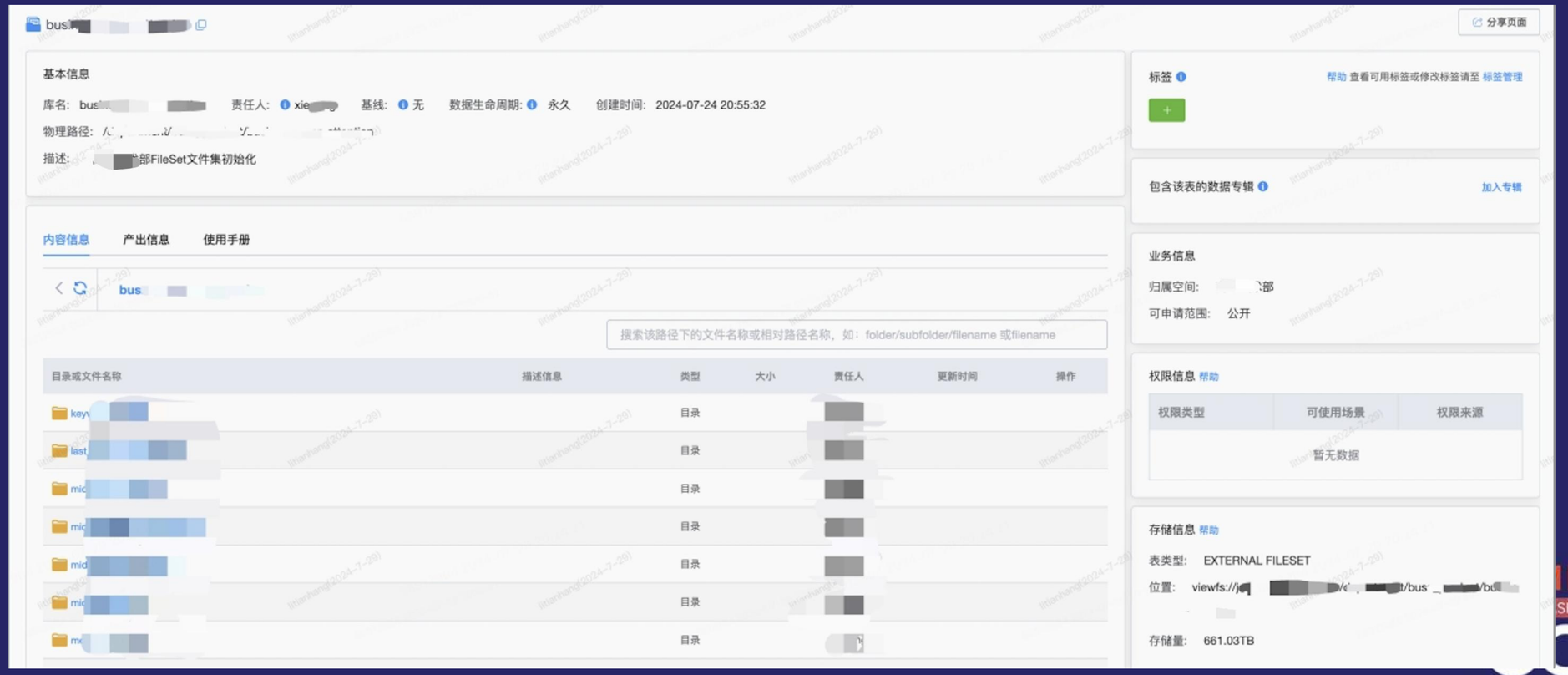
- ✓ 中心化管理 Kafka 元信息，避免数据不一致问题
- ✓ 降低用户使用 Kafka 消息成本
- ✓ 降低 Flink 任务维护成本，提高 Flink 任务上线效率

HDFS 文件治理现状

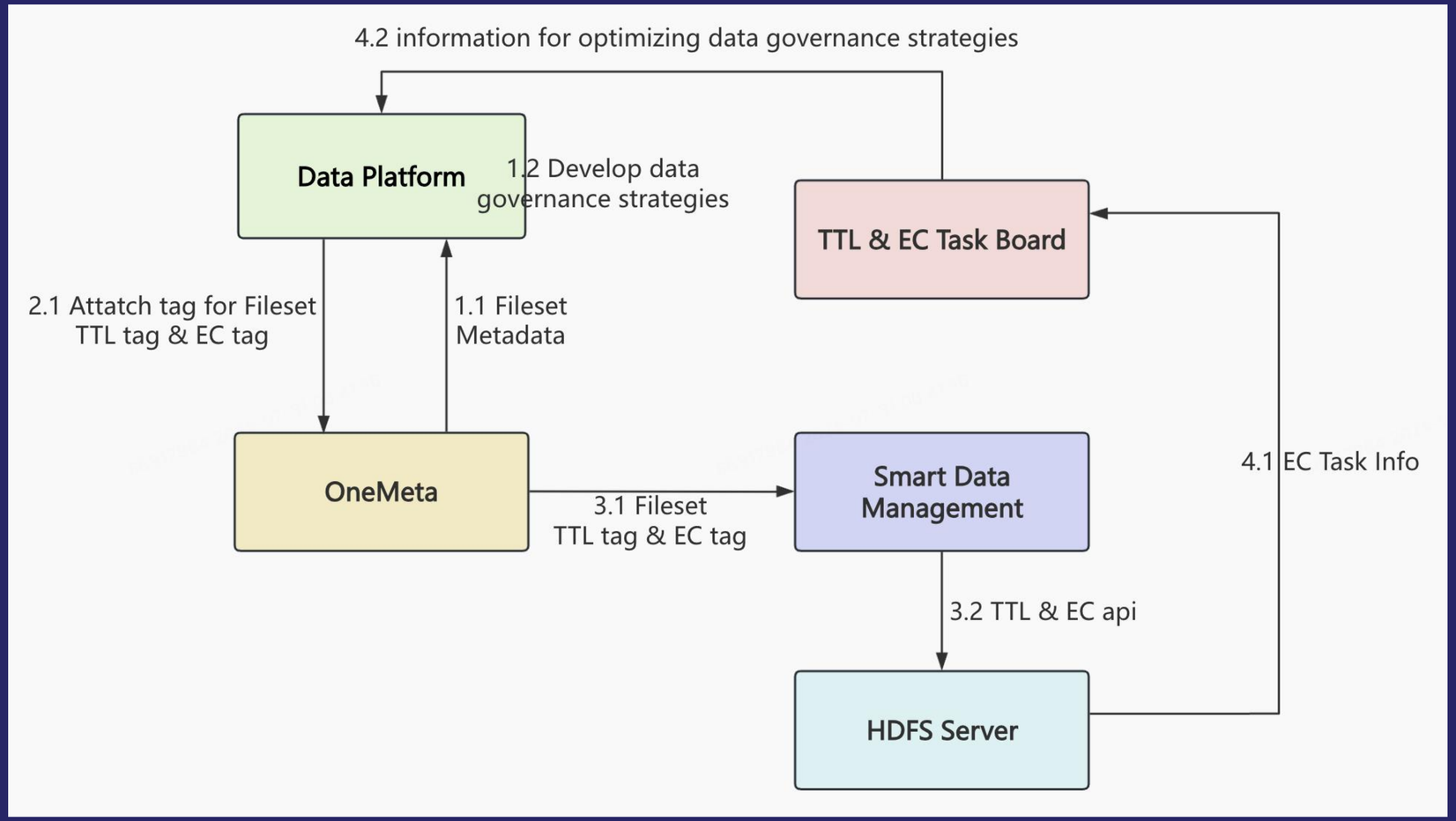
- HDFS 非表路径约总占的存储的30%，相关的治理收益较为可观
- 用户路径 HDFS 存储2024增长约 50%，AI/商业场景存在大量未被规范管理的 HDFS 路径
- 用户需要将 Airflow 管理的路径迁移至大数据平台
- 目前仅对 Hive 路径做了 EC/TTL，非表路径存在大量 EC/TTL 治理需求



Fileset 文件管理



Fileset 文件治理



- Fileset 文件治理主要流程：

- 1. 数据治理平台制定治理策略
- 2. 通过 OneMeta 对相应 Fileset 进行 TTL 和 EC 打标
- 3. SDM 读取 OneMeta tag，向 HDFS Server 发送 TTL & EC指令
- 4. 根据看板优化治理策略

✓ HDFS EC：**减少100PB+存储成本**

✓ HDFS TTL：**减少300PB+存储成本**

Fileset AI 文件管理应用

客户端使用 VIEWFS 场景：

1. Spark JAR：

```
# load fileset by Spark jar
df = spark.table("hiveCatalog01.test_database.exampleTable")
 .select('user_id', 'user_name')
 .where('log_date=20240401')

// transform logic ...
result = transform(df)

result.write
  .mode("overwrite")
  .parquet("viewfs://metalake01/Fileset_ai/user_info_schema/user_01_info/20240730/user.csv")
```

2. Spark SQL:

```
# Load fileset by Spark SQL
SELECT
  user_id,
  user_name,
  user_comment
FROM
  `viewfs://metalake01/Fileset_ai/user_info_schema/user_01_info/20240730/user.csv`
```

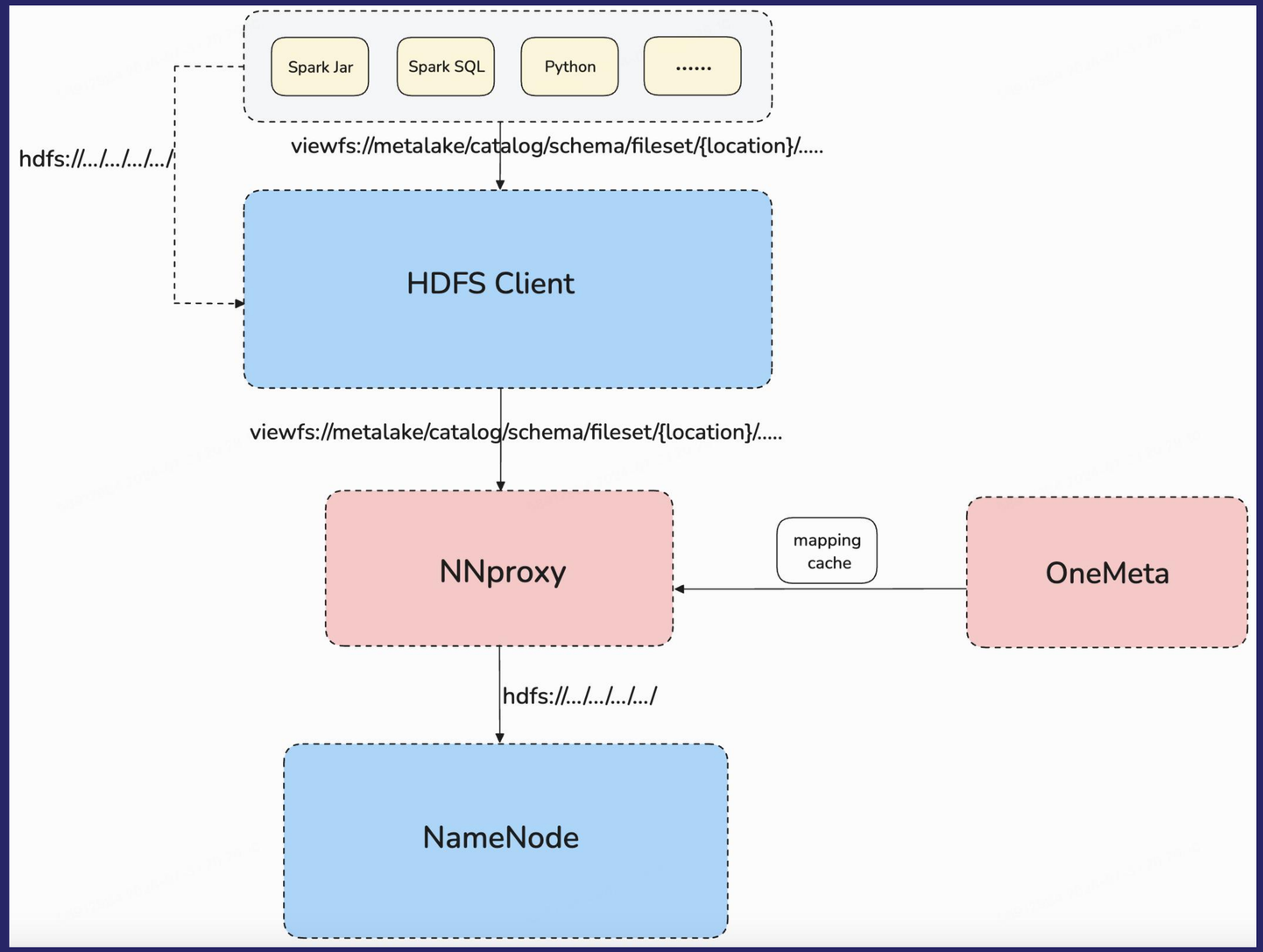
3. Python

```
# Load fileset by Python
# listFiles
fs = viewfs.GravitinoVirtualFileSystem(server_uri, metalake_name)
print(fs.ls('viewfs://metalake01/Fileset_ai/user_info_schema/user_01_info/20240730'))

# getFileDetailByFileName
with fs.open(path='viewfs://metalake01/Fileset_ai/user_info_schema/user_01_info/20240730/user.csv', mode='rd') as file:
  print(file.read().decode('utf-8'))
```

- ✓ AI 训练和数据处理共享一套 Meta，保证文件交付准确
- ✓ 文件资产可维护、可交接
- ✓ 规范 Fileset 使用方式

COMMUNITY
THE ASF CONFERENCE
CODE

VIEWFS 技术实现



基于对 HDFS Client 的改造，实现 VIEWFS

- 在HDFS Client 中内置 NNproxy，当用户传入 viewfs:// 时，根据 Mapping Cache 获取映射关系
- 优势：解耦用户与 VIEWFS，用户使用 VIEWFS 功能无需添加对应依赖，只需升级对应 HDFS Client 版本

04
Apache Gravitino 规划展望

GVFS 技术实现

1. 基于 Gravitino 的统一权限管理

2. 完善 AI 场景下的元数据治理

3. 提供 UDF 数据资源管理能力

4. 支持统一数据血缘

5. 接入内部多种数据源 Metadata

COMMUNITY
THE ASF CONFERENCE
CODE

bilibili

Thanks

哔哩哔哩技术

微信扫描二维码，关注我的公众号