Week04｜课时 4｜Pylceberg 本地最小闭环：Catalog、Warehouse、写入、历史查看、Schema Evolution

Table of contents

先把本地最小闭环跑通，再谈更复杂的平台接入

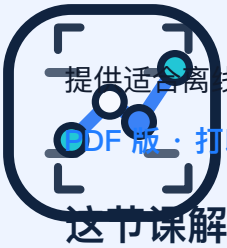
这一讲不做重型基础设施秀。

先解决最务实的问题：

如何在 PostgreSQL + MinIO + Pylceberg 的组合上，跑出一个当前项目真正可演示、可验收的 Week04 最小闭环。

进入课时 5 回看课时 3 返回 Week04 总览

下载讲义



提供适合离线阅读的 PDF 版和适合批注整理的 Word 版。

PDF 版 · 打印 / 离线阅读 Word 版 · 批注 / 二次整理

这节课解决什么问题

Week04 的前 3 课已经把判断、状态模型和最小 4 表设计讲清楚了。

这一讲开始真正收口到本地可跑闭环：

- catalog 怎样加载；
- warehouse 怎样声明；
- 表怎样 ensure / create；
- 数据怎样最小 materialize；
- history / snapshots 怎样查看；
- add-column schema evolution 怎样演示；
- Dagster 为什么本周只做 thin wrapper。

参考学习时间

建议按一节标准课安排：读完正文后，对照 repo 写清 catalog / warehouse / thin wrapper 路径。

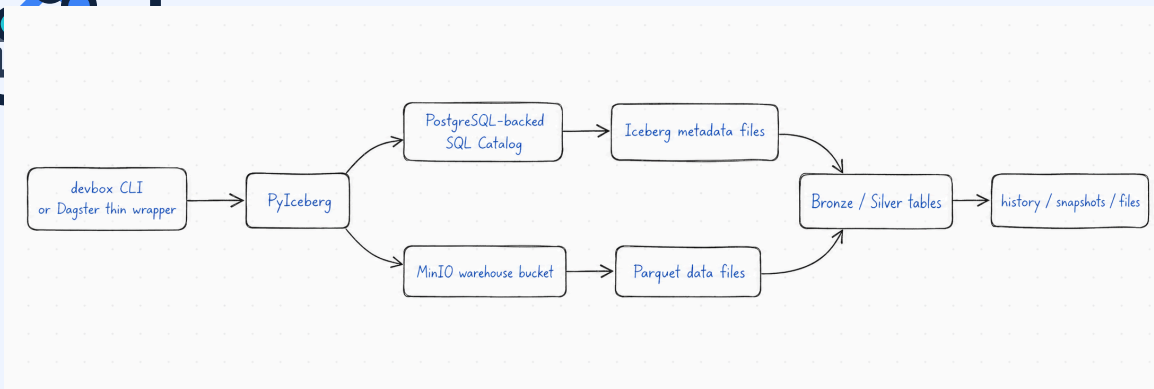
学完这一讲，你应该能做到什么

1. 解释为什么 PostgreSQL-backed SQL Catalog + MinIO warehouse 是当前最稳路线。
2. 解释为什么 Dagster 在本周只适合 thin wrapper，而不该先做全资产化重构。
3. 能说清 catalog / warehouse / table location 这三个配置层分别负责什么。
4. 能描述一次最小的 create table / append / inspect history / add column 演示。
5. 能写出 Week04 runbook 的主干。

本课产出

- docs/blueprints/week04/catalog_runtime_plan_v1.md
- runbooks/week04/README.md
- reports/week04/schema_evolution_demo_notes.md

先看一张总图



Catalog 记录“表在哪里、metadata 在哪里”；warehouse 存 Iceberg 的 metadata 和 data files；table location 是某张表在 warehouse 里的具体位置。PostgreSQL 在这里不是存所有 Iceberg 数据文件，MinIO 也不是普通文件堆，而是 warehouse 根。

1. 为什么当前课程先选这条技术路线

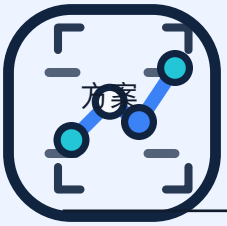
当前课程边界已经收得很清楚：

- 不引入 Spark / Hive Metastore / Nessie / Trino 作为学生主线；
- 不为了“平台看起来高级”牺牲学生本地可跑性；
- 继续复用现有 PostgreSQL、MinIO、Dagster 基线；
- 优先保证 devbox / Docker Compose 本地可跑。

因此，当前最合理的 Week04 路线是：

PyIceberg + PostgreSQL-backed SQL Catalog + MinIO warehouse。

方案	本地复杂度	能否覆盖 Week4 教学目标	学员安装负担	是否推荐	原因
PyIceberg + PostgreSQL SQL Catalog + MinIO	中	能覆盖 catalog、warehouse、write、inspect、schema evolution	中	推荐	复用现有 compose，核心机制可见



方案	本地复杂度	能否覆盖 Week4 教学目标	学员安装负担	是否推荐	原因
Pylceberg + SQLite Catalog	低	适合探索，但和教学容器环境不完全一致	低	不作为主线	SQLite 更适合本地试验，不适合多容器教学基线
REST Catalog	中高	能覆盖 catalog 机制	中高	暂不推荐	会把排错重点带到 catalog service
Spark + Hive/ Nessie/ Trino	高	能覆盖更多生产场景	高	不作为 Student Core Pack 主线	会抢走 table state reproducibility 的注意力

选择轻量路线不是“不生产级”，而是课程第一版要先让核心机制可见。真实企业可以接入更复杂 engine / catalog，但那不是 Week04 的主问题。

2. 为什么不先引入 Spark / Hive / Nessie / Trino

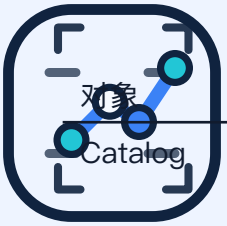
这些组件在真实平台里都可能合理，但它们不是本周教学主线。

组件	为什么不是本周主线
Spark	会把重点带到分布式计算，而不是表状态模型
Hive Metastore	对本地最小闭环负担偏重
Nessie	会引入 catalog branching 概念，超出 Week04
Trino	会把重点带到查询服务和 SQL engine
REST Catalog	对学生本地环境排错成本偏高

本周的目标不是“组件栈完整”，而是“状态闭环可证明”。

3. Catalog / Warehouse / Table Location 必须拆开

这 3 层最容易被混成一句“我已经配好 Iceberg 了”。



对象	真正负责什么	出错时的典型症状
Catalog	管理 namespace / table metadata 的入口	catalog 能连不上、namespace 找不到
Warehouse	默认对象存储根路径	表能注册但文件写不到正确 bucket
Table location	某张表最终真正落到哪里	表位置漂移、history 指向异常路径

如果这 3 层不拆开，create table 成功了，location 仍然可能错。

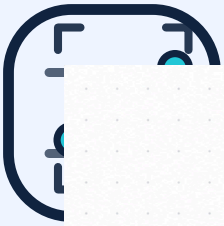
配置项	小白解释	典型来源	写错会怎样	是否可写死
catalog type	告诉 PyIceberg 用哪类 catalog	env / config	catalog 加载方式错误	no
catalog uri	PostgreSQL SQL Catalog 连接串	env	namespace / table 读写失败	no
warehouse root	Iceberg 默认存储根	env	表文件落到错误位置	no
s3 endpoint	MinIO / S3-compatible endpoint	env	本机能跑、容器里失败	no
access key / secret	warehouse 凭证	env / secret	写文件失败或权限错误	no
namespace	bronze / silver 等逻辑空间	code / config	表注册到错误 namespace	可配置，不应散落硬编码

⚠ 配置必须走 env，不要写死在课程代码里

课程页可以解释配置项，但真实运行值必须来自项目环境和 Week4 runbook。否则学员一换机器、容器或 bucket，就会得到一套不可复现的“本机成功”。

4. 最小 materialization 的输入输出

Week04 的 materialization 不应该凭空造数据。它应接住 Week03 的 ingest baseline：



这里的关键不是“写入成功”四个字，而是：

- 输入来自哪里；
- 字段怎样映射；
- 去重和幂等怎样保证；
- 写入后形成了哪个 snapshot；
- inspect 输出是否能证明表状态。

一条 materialize 命令背后至少应该发生这些动作：

1. 读取 env 与 Week4 配置；
2. 加载 SQL Catalog；
3. ensure namespace；
4. ensure table schema；
5. 从 Week3 源表或对象存储读取输入；
6. 按 source-to-Iceberg mapping 转成 Arrow / 可写数据；
7. append 或 overwrite 到 Iceberg；
8. 产生新 snapshot；
9. inspect history / files / snapshots；
10. 输出 report。

```
# 待 OmniSupport Copilot Week4 runbook 落地后替换为真实命令
python -m pipelines.lakehouse.materialize --table silver.ticket_fact --dry-run
```

这个代码块是结构化占位，不代表当前项目已经有这个命令。

5. Dry-run / plan / validation / idempotency

本周真实闭环里，至少要保留这些防线：

防线	作用
dry-run	先展示计划与影响，不直接污染表状态
materialization plan	明确 source、target、schema、dedupe key
schema validation	写入前确认字段、类型、nullable、语义边界
deterministic dedupe	同一输入重复到达，结果可预测
idempotency	同一次 materialization 重跑不产生不可解释副作用

这几件事决定 Week04 是工程闭环，不是“脚本碰巧跑过”。

6. 为什么 Dagster 本周只做 thin wrapper

这里必须先把边界讲清：

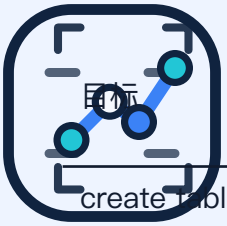
- Week04 的真实闭环，优先保证 CLI / devbox 路径可跑；
- 不要假设只改 pyproject.toml，Dagster 容器就自动拥有新依赖；
- Week06 会系统讲 asset factory、partition、backfill、asset checks；
- 因此，本周 Dagster 更适合做 thin wrapper，而不是先做全资产化重构。

这不是退让，而是工程顺序正确。

正确顺序是：核心逻辑先放在可直接调用的 Python module 和 CLI/devbox 路径里，便于本地排错；Dagster 只包装 orchestration 入口。等 Week06 讲 asset factory、partition、backfill、asset checks 时，再把这套能力系统资产化。

7. Pylceberg API 小抄

目标	Pylceberg API / 概念	本课用途	注意事项
load catalog	load_catalog	加载 PostgreSQL-backed SQL Catalog	配置来自 env



目标	Pylceberg API / 概念	本课用途	注意事项
create table	catalog.create_table	ensure 最小 4 表	schema 以项目 source of truth 为准
append	table.append	形成新 snapshot	注意幂等和去重
overwrite	table.overwrite	可控重写目标状态	不要无解释覆盖历史
scan	table.scan()	验证数据可读	可用于 smoke check
history	table.inspect.history()	看 snapshot 成为 current 的时间线	进入 baseline report
metadata log	table.inspect.metadata_log_entries()	看 metadata 演进	注意保留策略
schema add column	table.update_schema().add_column()	Week4 最小 evolution demo	不扩展到 rename/drop 主线

8. 当前命令如何写

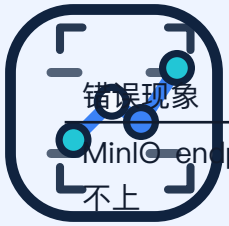
真实命令以 Week4 项目代码落地后同步为准。当前讲义只保留结构化占位：

```
# 待 Week4 项目代码落地后，同步 `pipelines.lakehouse.*` 的真实命令。
# 目标动作:
# 1. check catalog/env
# 2. ensure namespace/tables
# 3. materialize bronze/silver
# 4. inspect snapshots/history/files
# 5. run add-column schema evolution demo
```

不要在课程页面里编造 pipelines.ingestion.scripts.* 这类未经确认路径。

9. 常见错误排查

错误现象	可能原因	怎么排查	修复建议
catalog load 失败	uri、驱动、网络或凭证错误	先做 catalog smoke test	修正 env，确认容器网络



错误现象	可能原因	怎么排查	修复建议
MinIO endpoint 连不上	host/container endpoint 混用	分别在本机和容器内检查 endpoint	拆分外部和容器内 endpoint
bucket 不存在	warehouse 根未初始化	列 bucket / 检查 compose 初始化	创建 bucket 或修正 warehouse root
schema 不匹配	source schema 与 Iceberg schema 不一致	对照 iceberg_schemas.py 与 mapping	先改 mapping，不要硬改写入逻辑
重跑出现重复数据	dedupe key / idempotency 缺失	比对 row count 与 source coverage	增加 deterministic dedupe
inspect 没有 snapshot	只有 create table，没有实际写入	看 materialization report	执行真实 append / overwrite
add column 后写入失败	nullable、类型或 schema update 未提交	看 schema id 与 latest schema	只演示 nullable add-column

10. 最小演示应该包含什么

一套合格的 Week04 最小演示，至少要能回答：

1. catalog 是不是能正常加载；
2. warehouse 是不是能真正写入；
3. 最小 4 表是不是已经存在；
4. snapshot / history / files 能不能查看；
5. add-column evolution 能不能被证明。

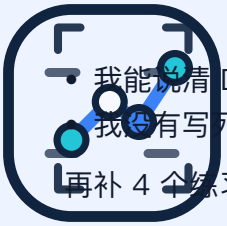
11. 本课收尾判断

! 核心判断

Week04 的本地最小闭环优先保证“可跑、可看、可解释”，不优先追求“平台全栈已经长齐”。

12. 本课自检清单

- 我能拆开 catalog、warehouse、table location。
- 我能解释为什么本周不先上 Spark / Hive / Nessie / Trino。



- 我能说清 Dagster thin wrapper 与 Week06 asset factory 的边界。
- 我没有写死项目仓库尚未落地的命令路径。

再补 4 个练习：

1. 用自己的话解释 catalog 和 warehouse 的差异。
2. 画出本地 Pyiceberg 闭环。
3. 写出一次 materialize 的 8 个关键步骤。
4. 说明为什么本周不需要 Spark 也能讲清 Iceberg 核心能力。

13. 课后最小行动

新建：

```
docs/blueprints/week04/catalog_runtime_plan_v1.md
```

至少写清：

1. catalog 配置项；
2. warehouse 根路径；
3. 4 张表的 namespace / table location 规划；
4. CLI 与 Dagster thin wrapper 边界；
5. 3 个最可能出错的环境问题。

延伸阅读

- Pyiceberg / Configuration – Catalog 配置[1]
- Pyiceberg / API – Catalog、Table、Scan、Inspect[2]
- Apache Iceberg / Reliability[3]
- Apache Iceberg / Maintenance[4]

---

[1] Pyiceberg Configuration 文档说明 catalog 可以通过配置加载，SQL catalog 使用 `type: sql` 与 `uri` 等参数；本课的 PostgreSQL-backed SQL Catalog / MinIO warehouse 方案要以项目 runbook 的真实配置为准。Pyiceberg | Configuration

[2] Pyiceberg API 文档包含 catalog、table、scan、append、overwrite、schema update 与 inspection 等接口，是后续把本课“本地最小闭环”落成真实 runbook 的主要 API 参考。Pyiceberg | API

[3] Iceberg Reliability 文档解释了原子提交、可靠读取、version history 和 rollback；这能帮助你判断本课的最小闭环为什么必须包含 write、inspect、history 和 schema evolution，而不只是 create table。Apache Iceberg | Reliability

[4] Iceberg Maintenance 文档解释 snapshot、metadata、orphan files 等维护动作。本课不把 cleanup / compaction 做成主线，但需要你知道后续为什么要维护这些对象。Apache Iceberg | Maintenance