Clase 209 — Pipelines con Prefect o Dagster

Parte: 5 — Ingeniería de Datos · Fuente: Prefect 3 docs + Dagster docs + Reis & Housley cap. 8. Duración estimada: 75 min.

Clase 209 — Pipelines con Prefect o Dagster

> *Parte: 5 — Ingeniería de Datos · Fuente: Prefect 3 docs + Dagster docs + Reis & Housley cap. 8. Duración estimada: 75 min.*

Objetivo

Construir el mismo pipeline de Clase 208 con Prefect 3 (API Python moderna, hybrid execution) y con Dagster (asset-oriented, mejor lineage). Entender qué problemas resuelven mejor que Airflow y cuándo elegir cada uno.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Escribir un flow Prefect con @flow/@task, deployments, work pools y workers.
- Definir assets Dagster (@asset) y entender la diferencia entre "task-oriented" (Airflow/Prefect) y "asset-oriented" (Dagster).
- Configurar hybrid execution Prefect: control plane en cloud, workers en tu infra (sin enviar data sensible).
- Usar Dagster's UI para ver lineage automático: qué asset depende de qué, cuándo se materializó cada uno.
- Decidir Airflow vs Prefect vs Dagster según contexto (equipo, escala, tipo de pipeline).

Temas

#	Tema	Por qué importa
1	Prefect 3: flows, tasks, deployments	Reemplaza DAGs con código Python idiomátic
2	Work pools + workers	Hybrid execution: control en cloud, comput
3	Dagster: asset-oriented vs task-oriented	Lineage automático, mejor para data produc
4	Software-defined assets (SDA)	Cada asset es código + metadata + checks.
5	Scheduling: cron, interval, event-driven	Las 3 formas de disparar.
6	Cuándo migrar de Airflow	Costo de migración vs beneficio.

Definiciones y características

- Prefect Flow: equivalente al DAG. Decorador @flow. Puede invocar tasks (@task) y sub-flows.
- Prefect Task: unidad de trabajo. Tiene retries, cache, timeout configurables. Diferente a Airflow: las tasks pueden tener loops/condicionales sin XCom hackery.
- Deployment: la receta de "cómo y cuándo correr este flow" (schedule, parameters, infra). Prefect 3 los empaqueta como código (flow.deploy(...)).
- Work pool: cola lógica donde los deployments mandan runs. Los workers la consumen.
- Worker: proceso que corre los runs. Puede vivir en tu laptop, K8s, ECS, etc. Cloud variant: control plane managed; workers tuyos (hybrid).
- Dagster Asset: objeto persistente versionado (una tabla, un modelo, un dashboard). Definido con

@asset. Las dependencias entre assets se infieren del código.
- Op (Dagster): equivalente más cercano a "task" — unidad atómica. Los assets generalmente componen ops.
- Materializar: ejecutar el código que produce el asset (re-genera el output). Dagster trackea cuándo se materializó cada asset por última vez.
- Sensor (Dagster/Prefect): trigger basado en eventos (nuevo archivo, mensaje en queue) en vez de schedule.

Dataset / recursos

- Mismo pipeline de Clase 208 (BTC price), implementado dos veces.
- Librerías: prefect>=3, dagster>=1.7, duckdb, pandas, requests.

Ejercicios

1. Prefect flow: copiá la lógica del DAG Airflow al patrón Prefect: @flow def btc_pipeline(): notify(transform(load(extract()))). Corré python btc.py directo (no necesita scheduler).
2. Deployment Prefect: flow.serve(name="btc-hourly", cron="0 "). Dejá corriendo, observá ejecuciones programadas en localhost:4200.
3. Dagster assets: convertí las funciones a @asset def btc_price(), @asset def daily_avg(btc_price). Dagster infiere daily_avg depende de btc_price. UI muestra el grafo.
4. Materializar: en Dagster UI, click "Materialize" sobre btc_price. Solo se ejecuta ese asset; daily_avg queda "stale" hasta que se materialice también.
5. Comparativa: mismo pipeline en Airflow + Prefect + Dagster. Compará LOC, claridad, UI, velocidad de feedback dev.

Homework verificable

Repo con el mismo pipeline implementado en los 3 frameworks:

1. airflow/dags/btc.py (de Clase 208).
2. prefect/btc.py con @flow/@task y deployment programado.
3. dagster/btc.py con @asset definitions y un Definitions object.
4. README comparativo: LOC, complejidad de setup, calidad de UI, lineage support, cuándo elegir cada uno.
5. Bonus: GitHub Actions que corre los 3 en CI y verifica que producen el mismo output.

Criterio de aceptación: los 3 pipelines producen idénticos resultados sobre el mismo input; el README compara honestamente fortalezas/debilidades.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
prefect server start falla	Otro proceso en :4200. Fix: --port 4201.
Deployment no se ejecuta automáticamente	Workers no están corriendo o pool mal conf
Dagster asset materialization "stale" pero	Auto-materialization no está habilitada. F
Re-import circular entre @assets	Dagster intenta resolver dependencias en i
Performance lento en Prefect con muchas ta	El backend default está en SQLite. Fix: pa

Mixed Airflow + Prefect + Dagster en el mi	Cada uno tiene su propio ecosistema. Fix:

Preguntas frecuentes

Airflow vs Prefect vs Dagster en una frase

- Airflow: el camión de carga industrial — feo pero llega.
- Prefect: el Tesla — UI moderna, código limpio, hybrid execution.
- Dagster: la BMW — lineage hermoso, ideal cuando pensás en data products.

¿Vale la pena migrar desde Airflow?

Calcular: (costo de migrar N DAGs) vs (ahorro mensual en mantenimiento + horas dev). Si Airflow funciona y nadie está sufriendo: no. Si nuevos pipelines: empezar con Prefect/Dagster, dejar los viejos donde están.

¿Prefect Cloud o self-hosted?

Cloud: control plane managed, free tier generoso, hybrid execution (workers tuyos). Self-hosted: prefect server start corre todo local — para dev. Para prod: Cloud es mucho más práctico.

¿Dagster's asset model es overkill para pipelines simples?

Para 3-5 tareas en cascada: sí, Prefect es más simple. Para data warehouse con 100+ tablas modeladas: Dagster brilla (lineage, freshness, partition awareness).

¿Cómo manejo secrets en Prefect/Dagster?

- Prefect: Secret.load("my-key").get() desde blocks (cifrados en backend).
- Dagster: EnvVar("MY_SECRET") o resources con ConfigurableResource.

Ambos integran con AWS Secrets Manager / GCP Secret Manager / Vault.

¿Puedo usar Dagster con dbt?

Sí — dagster-dbt carga modelos dbt como assets Dagster automático. Lineage atraviesa Python → dbt → SQL → tablas. Es el sweet spot del stack moderno.

Referencias

- Prefect 3 docs — empezar por Quickstart.
- Dagster docs — Concepts → Assets.
- Reis & Housley Fundamentals of Data Engineering (O'Reilly, 2022) cap. 8.
- Prefect vs Airflow (Prefect, 2024) — sesgado pero útil.
- dagster-dbt integration — el combo más usado.

Siguiente clase

Clase 210 — PySpark para datasets grandes

Apéndice: notebook (primer bloque)

Mismo pipeline BTC, implementado en Prefect y en Dagster. Comparación lado a lado.

```
import os, tempfile, shutil, json
from pathlib import Path
WORK = Path(tempfile.gettempdir()) / 'flows_demo'
if WORK.exists(): shutil.rmtree(WORK)
WORK.mkdir(); os.chdir(WORK)
```

Archivos complementarios

- notebook.ipynb