Dagster Running Dagster

A Deep Dive into Our Open Platform

Nick Roach (@nicklausroach) - Data Engineer - 2025-05-13

Speakers

Nick Roach
Data Engineer

Colton Padden
Developer Advocate

Alex Noonan
Developer Advocate

dagster

Table of contents

dagster

What is Dagster Open Platform?

01

Dagster's Internal Data Platform

DOP powers all of Dagster Labs' in-house analytics, consuming data from various sources using a variety of tools. Dagster acts as the engine powering all of these pipelines from ingestion to reporting.

02

Testing Environment for Experimental Features

As the internal Dagster data platform, DOP has access to bleeding edge Dagster features. We implement these features in the data platform and ensure they're ready to be released to a broader audience.

03

Open Source Codebase

We publish much of DOP's code to a public Github repository as a learning resource and guide for running Dagster in a production environment.

Dagster+ Event Ingestion

Current State

Dagster+ Events

A log of activities performed by our Dagster+ customers

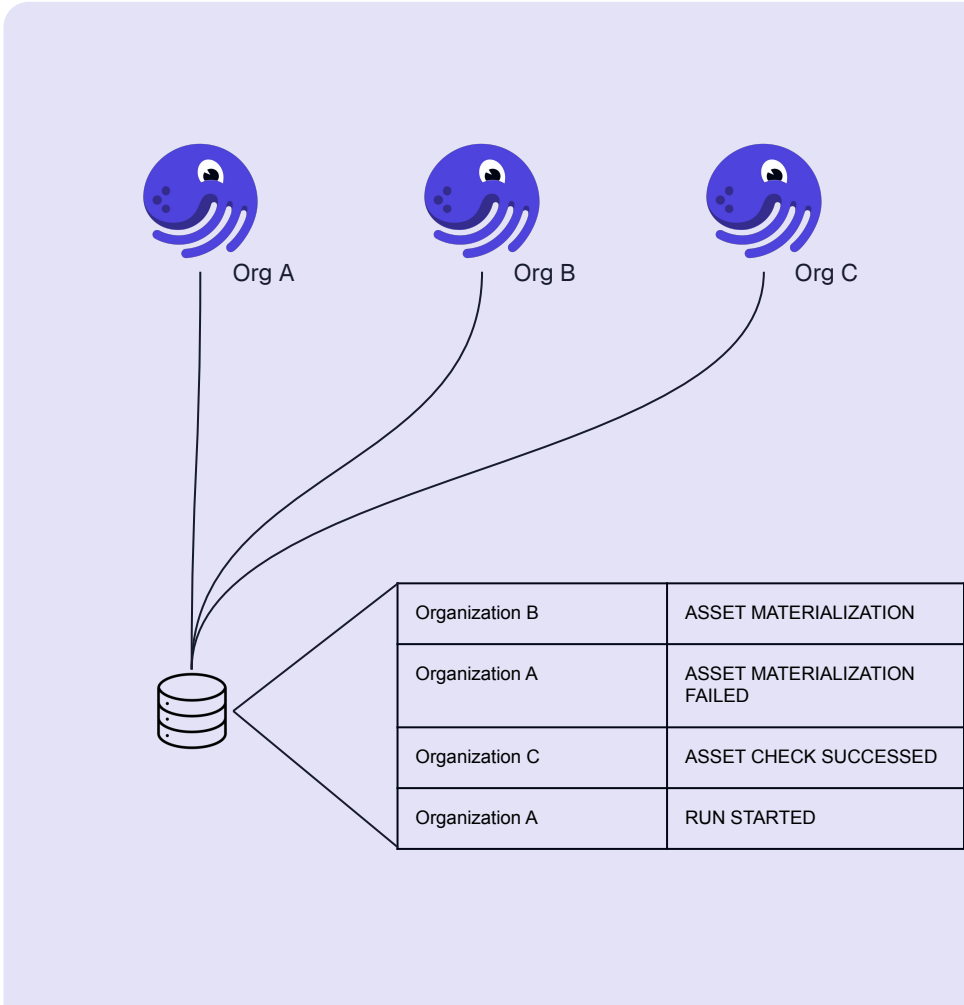
The pipeline we'll walk through ingests events produced by Dagster+ organizations

This event data gives us second-by-second information about what is going on inside a deployment

An event can be an asset materialization, a run start, an asset check failure, etc.

The event data is easily our most used source of data at Dagster, powering billing, Dagster Insights, and many of our analytical workflows

dagster



How do we currently ingest this data into our data warehouse?

- This event data lands in a **postgres database** that powers the Dagster+ product.
- We use a tool called **Sling** to poll this postgres database on a very short interval (~every 5 minutes).
- Sling grabs all events that have landed in the database since the last event that landed in our data warehouse.
- This set of events is pulled into a database in our warehouse and then modeled via **dbt** downstream.

dagster

What are the problems with this current method of ingestion?

- This regular polling puts unnecessary stress on the postgres database.
- If we fall behind collecting events, we have to “catch back up.”
- We have begun to shard our postgres database, and will need to repeat this ingestion scheme for every new shard.
- We have little visibility into *which* events were ingested during a given run of the pipeline.

What is Streaming?

Streaming Event Data

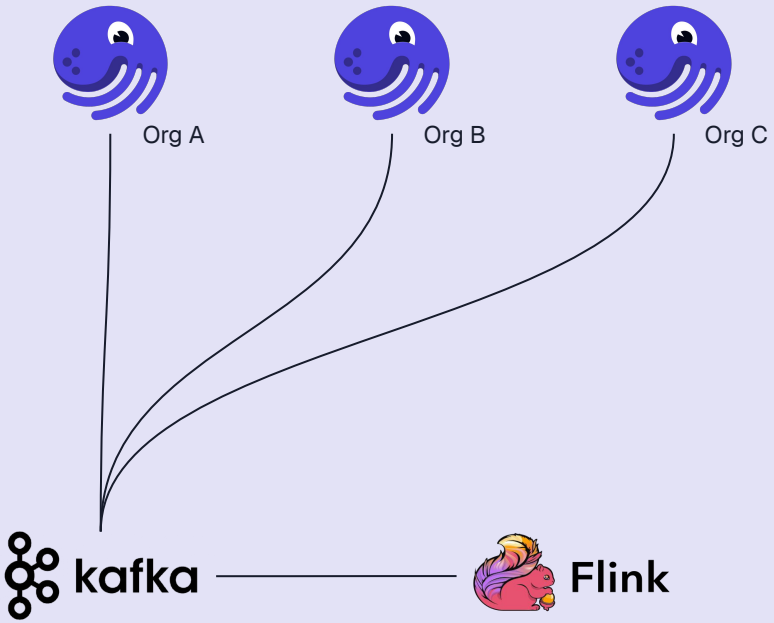
Processing our event data in near real time

When data is “streamed” it is processed continuously and in near real time as the data is generated

We now have a Kafka producer that allows us to stream process our event data

We’ll use a tool called Flink to land this data in a location that can be accessed by our data warehouse

We can then batch this data into our data warehouse at a cadence of our choosing



dagster

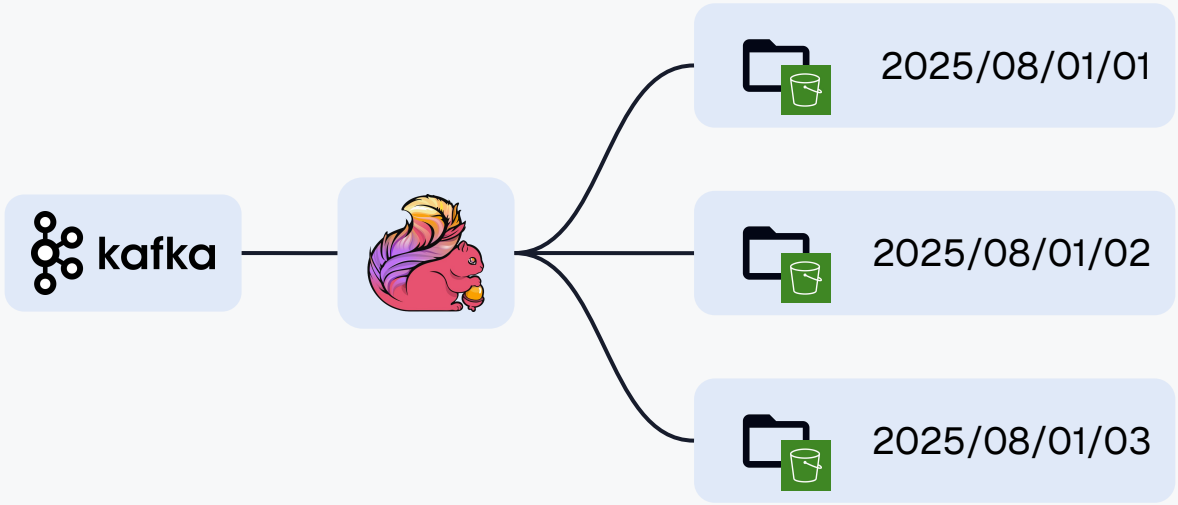
What makes this better than our current method of ingestion?

- We no longer have to directly connect to our postgres database.
- “Catching back up” after a loss of connection is no longer an issue.
- Our database shards are no longer an issue, all events can be written to the same location.
- We can easily monitor all steps in this pipeline and which events are landed in our warehouse.

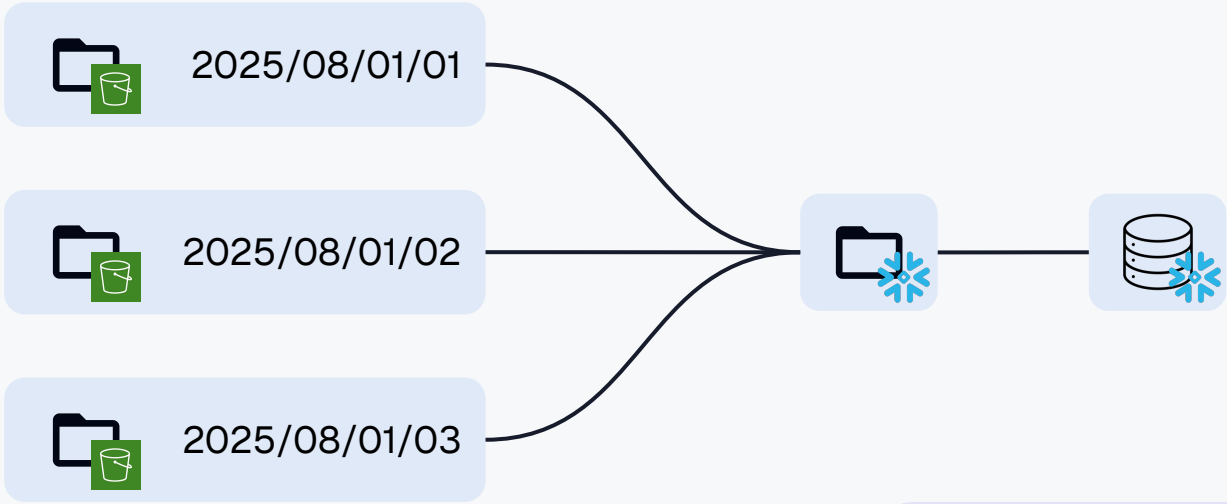
dagster

Dagster+ Event Ingestion

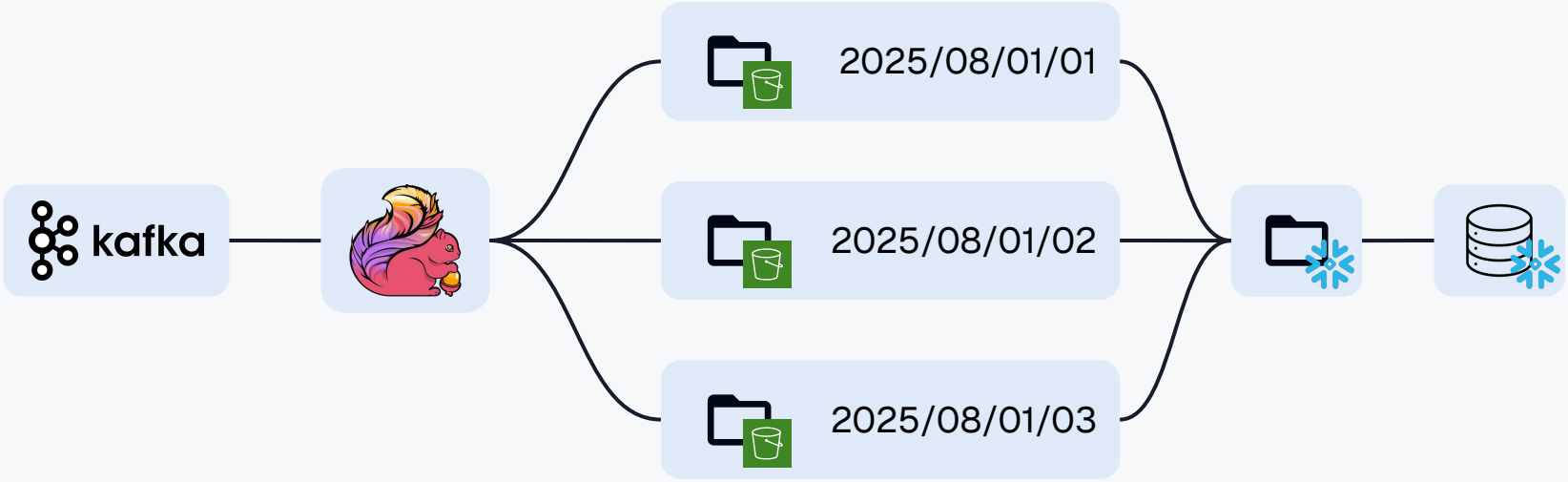
Our New Solution

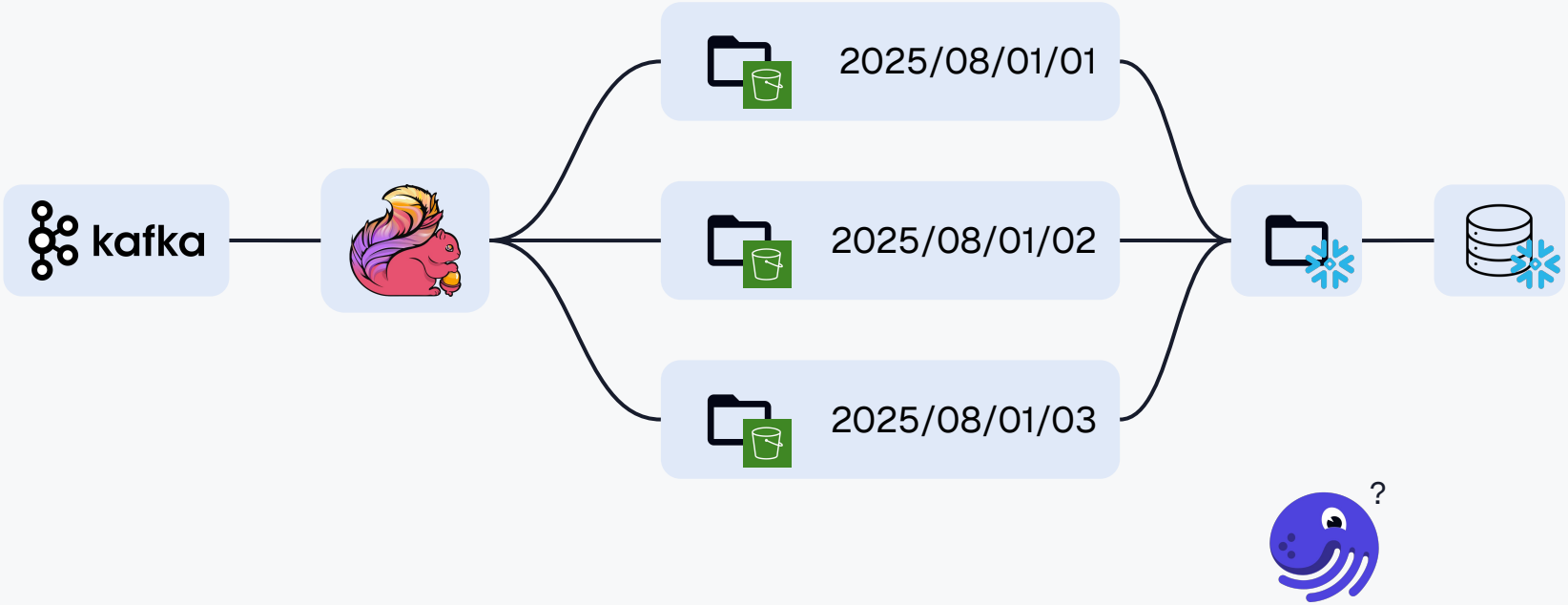


We use Apache Flink to consume the Kafka stream and land the event data into an hourly-partitioned directory in S3



We have an external Snowflake stage connected to the S3 bucket pointed at the partitioned directory, we then copy that data into our warehouse

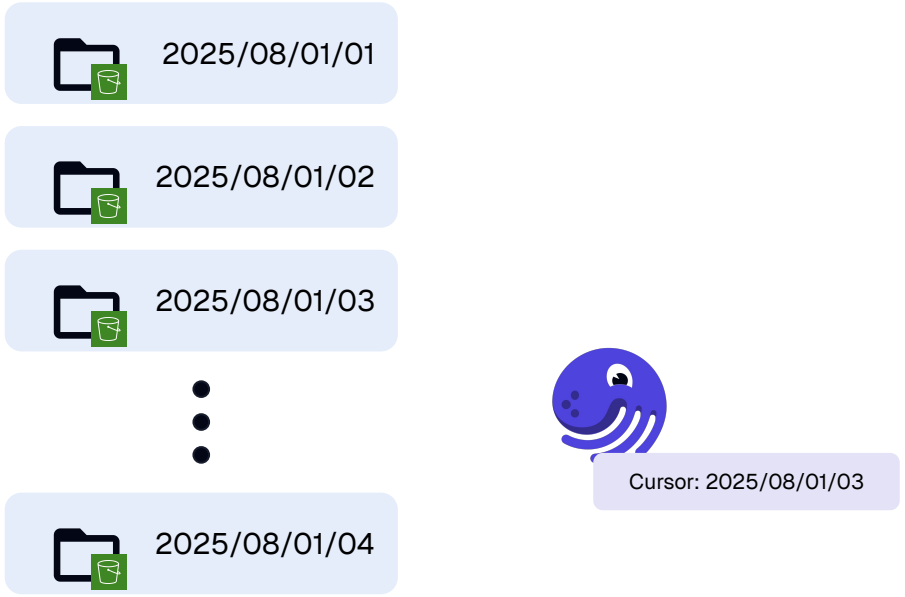

kafka
2025/08/01/01
2025/08/01/02
2025/08/01/03


kafka
2025/08/01/01
2025/08/01/02
2025/08/01/03
?

Using Sensors to trigger a run when new data arrives

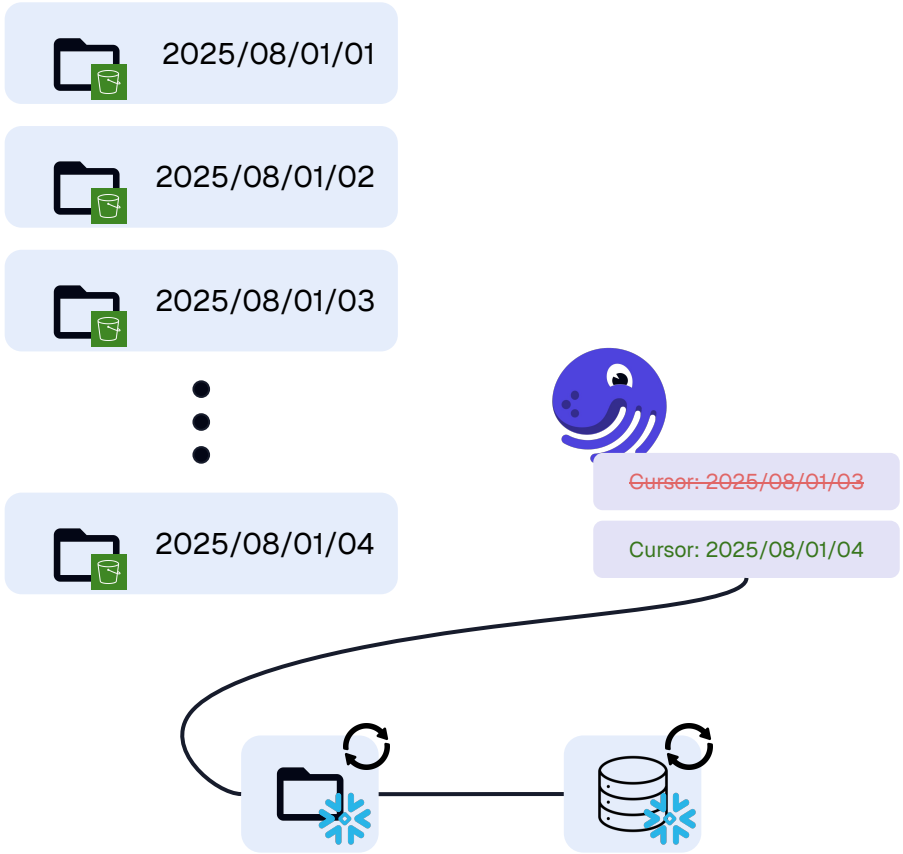
dagster

Using Sensors to trigger a run when new data arrives



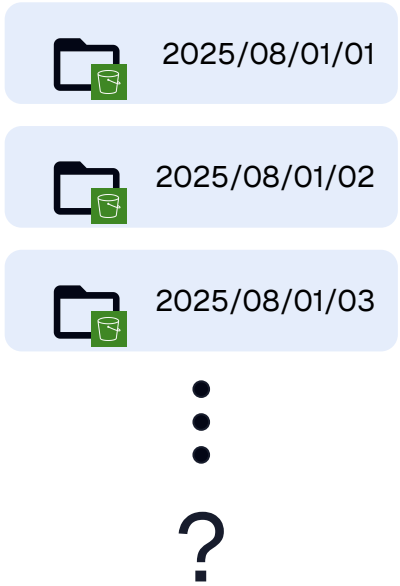
dagster

Using Sensors to trigger a run when new data arrives



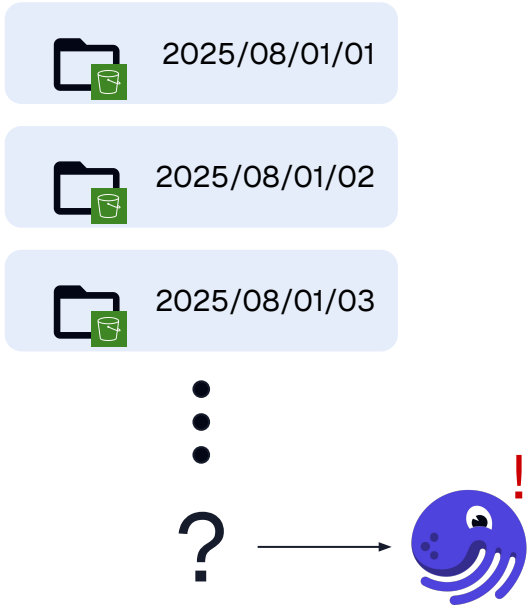
dagster

Using Asset Checks to ensure our data is landing



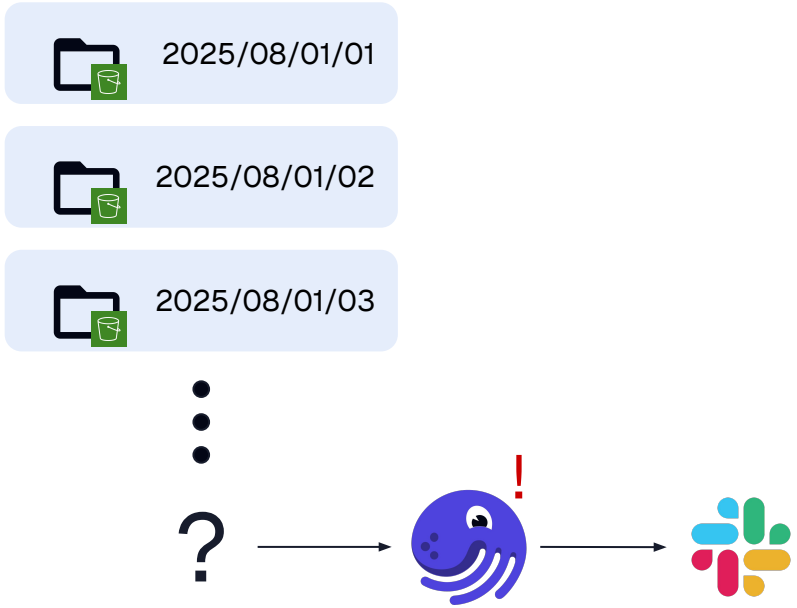
dagster

Using Asset Checks to ensure our data is landing



dagster

Using Asset Checks to ensure our data is landing

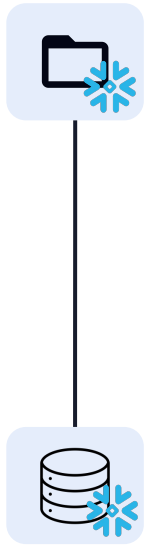


dagster

Using Components to define and manage our Snowflake objects

```yaml
type: dagster_open_platform.lib.SnowflakeCreateOrRefreshComponent
attributes:
  name: stage_flink_event_log
  database_name: "aws"
  schema_name: "dev"
  type: "stage"
  role: AWS_WRITER
  options:
    url:
    storage_integration:
    directory:
      enable: true
    file_format:
      type: JSON
      strip_outer_array: true
      file_extension: ".jsonl"
  asset_attributes:
    description: "Snowflake stage for Dagster+ event log data."
    automation_condition: "{{ automation_condition.on_cron('0 3 * * *') }}"
    group_name: "aws_stages"
    deps:
      - "flink_event_log_writer"
```

```yaml
type: dagster_open_platform.lib.SnowflakeCreateOrRefreshComponent
attributes:
  name: ext_flink_event_log
  database_name: "aws"
  schema_name: "dev"
  type: "external table"
  role: AWS_WRITER
  partition_by: REPLICATION_HOUR
  columns:
    - name: FILENAME
      type: VARCHAR
      as: METADATA$FILENAME
    - name: REPLICATION_HOUR
      type: TIMESTAMP
      as: "cast(split_part(METADATA$FILENAME, '/', 3) || '-' || split_part(ME
  options:
    location: "@stage_flink_event_log"
    file_format:
      type: JSON
      strip_outer_array: false
      file_extension: ""
    auto_refresh: false
    refresh_on_create: true
    comment: "External table for stage stage_flink_event_log"
  asset_attributes:
    description: "Snowflake external table for Dagster+ event log data."
    automation_condition: "{{ automation_condition.on_cron('0 3 * * *') }}"
    group_name: "aws_external_tables"
    deps:
      - "aws/dev/stage_flink_event_log"
```



dagster

Walking Through the Pipeline

Q&A

Join Our Beloved Slack Community

Check Out DOP

Register for the Neurospace Deep Dive

dagster

Subtitle goes here

Thank you!