DNTA 2025 SF

Composable Data Workflows

Building Pipelines That Just Work

Dennis Hume
Data Engineer / DevRel / Education

DATA SF 2025

Dagster

- Building out examples and tutorials for Dagster
- Maintaining Dagster University
- Writing content

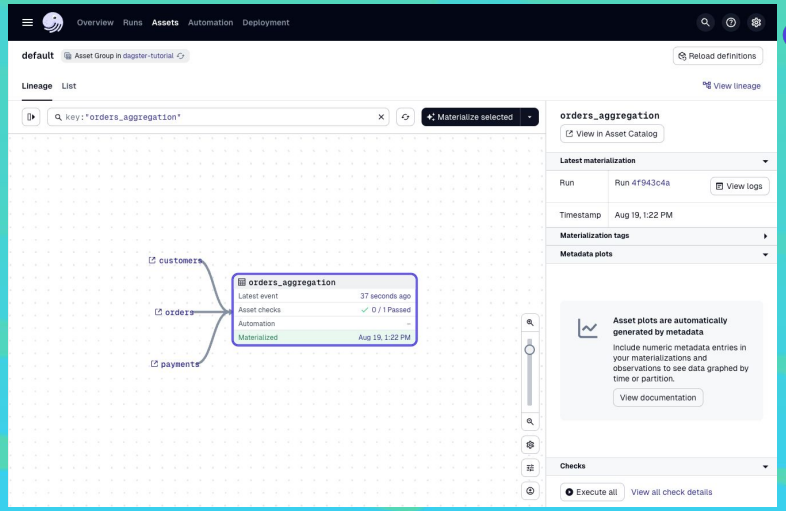
Dennis Hume

Data Engineer / DevRel / Education

Dagster

2025 DATA SF

Dagster is an open-source data orchestration platform designed to build, run, and observe data pipelines with reliability and structure



Dagster
Data Platform

Dagster

DATA SF 2025

Prerequisites

- Assumes no knowledge of Dagster
- Basic understanding of DuckDB
- Basic Python

What are we building?

In this workshop are building a system to identify duplicate issues for Github repositories that costs $0.

- No external services
- No authentication
- No cost

And we will learn Dagster along the way.

Support specifying configuration for backfills #28615

Closed as duplicate of #10079

Github Archive

GH Archive

Star Post

Error loading chart.

Open-source developers all over the world are working on millions of projects: writing code & documentation, fixing & submitting bugs, and so forth. GH Archive is a project to **record** the public GitHub timeline, **archive it**, and **make it easily accessible** for further analysis.

GitHub provides 15+ event types, which range from new commits and fork events, to opening new tickets, commenting, and adding members to a project. These events are aggregated into hourly archives, which you can access with any HTTP client:

Query	Command
Activity for 1/1/2015 @ 3PM UTC	`wget https://data.gharchive.org/2015-01-01-15.json.gz`
Activity for 1/1/2015	`wget https://data.gharchive.org/2015-01-01-{0..23}.json.gz`
Activity for all of January 2015	`wget https://data.gharchive.org/2015-01-{01..31}-{0..23}.json.gz`

What are our steps

1. Ingest Github Data
2. Model Github Issues
3. Monitor Data Quality
4. Generate Embeddings
5. Duplication Search
6. Enable Automation

0. Project Setup

Project setup local

https://github.com/dehume/small-data-sf

git clone
git@github.com:dehume/small-data-sf.git

- **uv**

```
uv sync
source .venv/bin/activate
```

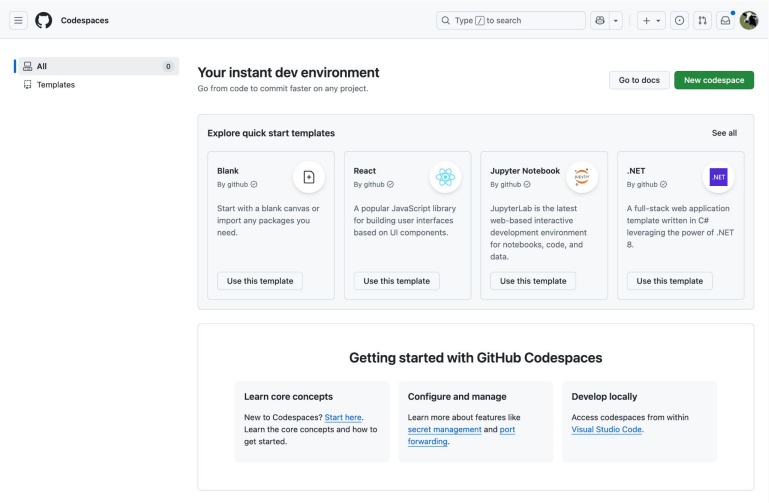
- **pip**

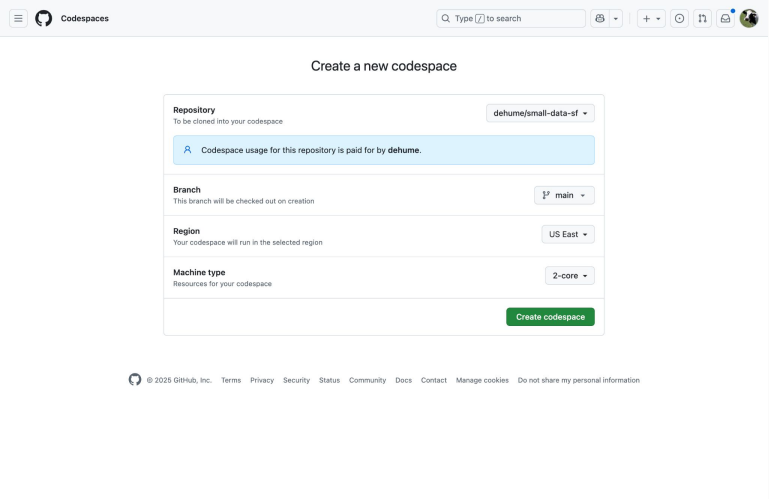
```
python -m venv .venv
source .venv/bin/activate
pip install -e .
```

Project setup Codespace

You can follow this entire project within a [Github Codespace](). This will handle the

https://github.com/dehume/small-data-sf





Setting up your Github Codespace

1. Go to https://github.com/codespaces
2. Click “New Codespace”
3. Select repository “dehume/small-data-sf”
4. **Select 8 Cores**
5. Click “Create Codespace”
6. Take ~a minute to build (Prebuild)

Github Codespace

small-data-sf [Codespaces: repulsive spooky crematorium]

EXPLORER

SMALL-DATA-SF [CODESPACES: REPULSIVE SP...

- _completed
- .devcontainer
- data
- models
- src
- tests
- .gitignore
- dagster.yaml
- pyproject.toml
- README.md
- uv.lock

[Preview] README.md

Small Data SF

Installing dependencies

Ensure `uv` is installed following their official documentation.

Create a virtual environment, and install the required dependencies using *sync*:

```
uv sync
```

Then, activate the virtual environment:

OS	Command
MacOS	`source .venv/bin/activate`
Windows	`.venv\Scripts\activate`

Running Dagster

Start the Dagster UI web server:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

bash

```
@dehume → /workspaces/small-data-sf (main) $ dg
```

DuckDB CLI

https://duckdb.org/install/?platform=macos&environment=cli

or...

```
1 brew install duckdb
```

DEMO

```
> dg check defs
> dg dev
```

Completed Code

```
_completed
└── small_data_workshop
    ├── __init__.py
    ├── constants.py
    ├── definitions.py
    └── defs
        ├── __init__.py
        ├── assets.py
        ├── jobs.py
        ├── resources.py
        ├── schedules.py
        └── sensors.py
```

```python
@dg.sensor(
    job_name="similarity_search_job",
    minimum_interval_seconds=60,
)
def new_issues_sensor(
    context: dg.SensorEvaluationContext,
    duckdb_storage: DuckDBResource,
):
    last_issue_id = context.cursor if context.cursor else "0"

    try:
        with duckdb_storage.get_connection() as conn:
            new_issues = conn.execute(f"""
                select id
                from {ISSUES_TABLE_NAME}
                where org = '{ORG_NAME}'
                and id > '{last_issue_id}'
            """).fetchall()

            if len(new_issues) > 0:
                context.log.info(f"Found {len(new_issues)} new issues")
            else:
                context.log.info("No new issues found")

        for issue in new_issues:
            issue_id = issue[0]

            # Update cursor to the highest issue ID
            if int(issue_id) > int(last_issue_id):
                last_issue_id = issue_id

            context.update_cursor(str(last_issue_id))

            yield dg.RunRequest(
                run_key=str(issue_id),
                run_config={
                    "ops": {
                        "similarity_search": {
                            "config": {
                                "id": issue_id,
                                "similarity_score": 0.1,
                                "num_results": 3,
                            }
                        }
                    }
                },
            )

    except Exception as e:
        return dg.SkipReason(f"Sensor failed with error: {str(e)}")
```

1. Ingest Github Data

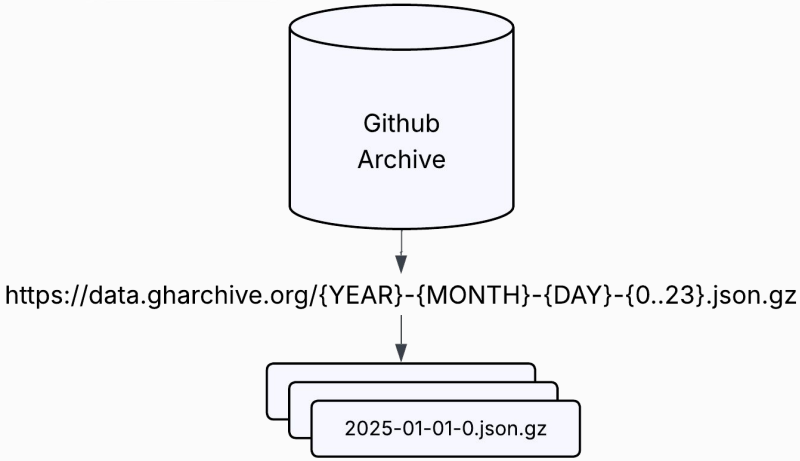
What is an asset

A Dagster **asset** is an object in persistent storage such as a database table, file or persisted machine learning model. An asset definition is a description, in code, of an asset that should exist and how to produce and update the asset.

```
@dg.asset
def weekly_sales_report(context: dg.AssetExecutionContext):
    ...
```

The data under our asset

The Github Archive consists of individual files for each hours. This data is retrieved from unique URLs for each JSON file.



Our first asset

```python
@dg.asset
def github_archive_files(
    context: dg.AssetExecutionContext,
) -> str:
    return "https://data.gharchive.org/2025-10-01-0.json.gz"
```

DEMO

```
> dg check defs
> dg dev
```

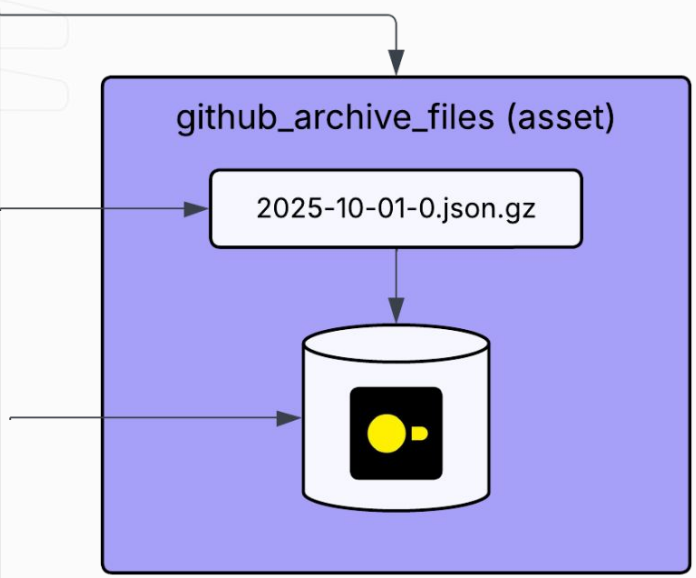
Our first `resource`

```python
database_resource_storage = DuckDBResource(database="workshop.duckdb")


@dg.definitions
def resources():
    return dg.Definitions(
        resources={
            "duckdb_storage": database_resource_storage,
        }
    )
```

assets and resources

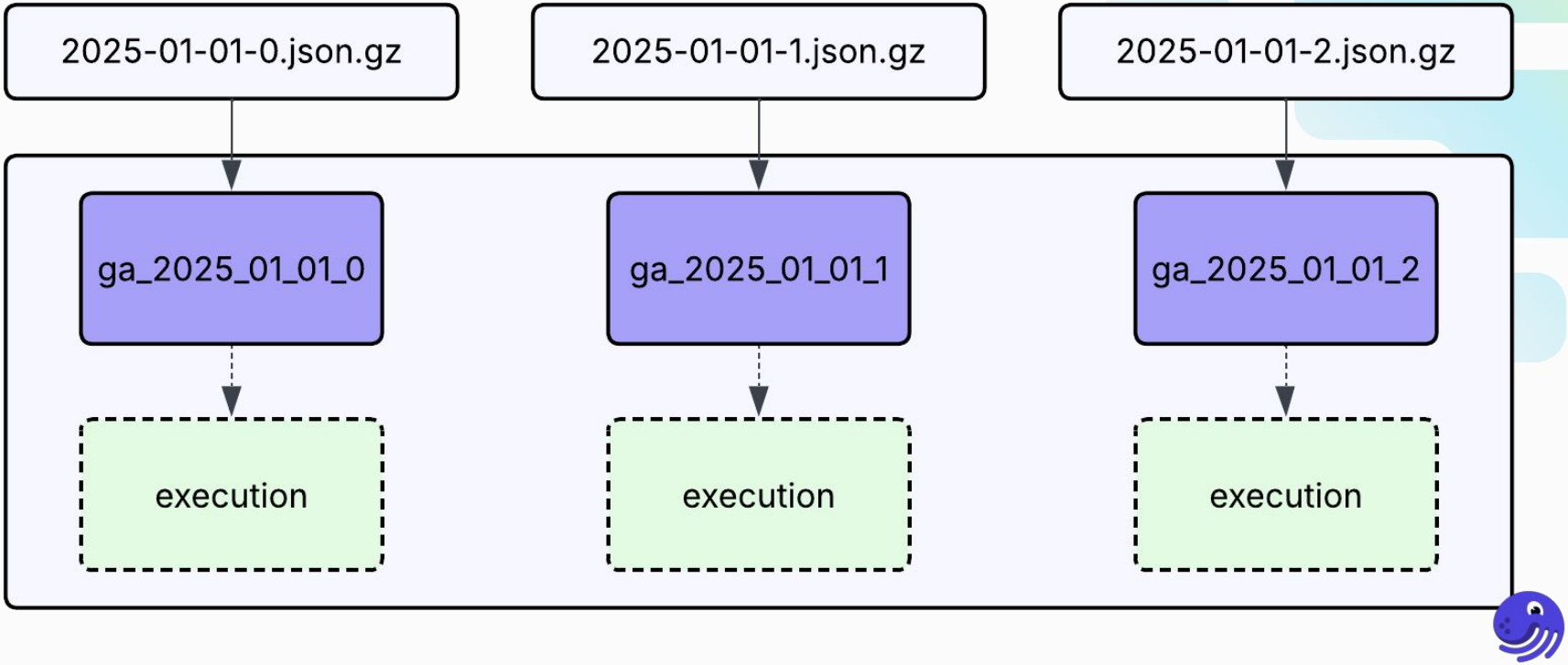
Our first asset

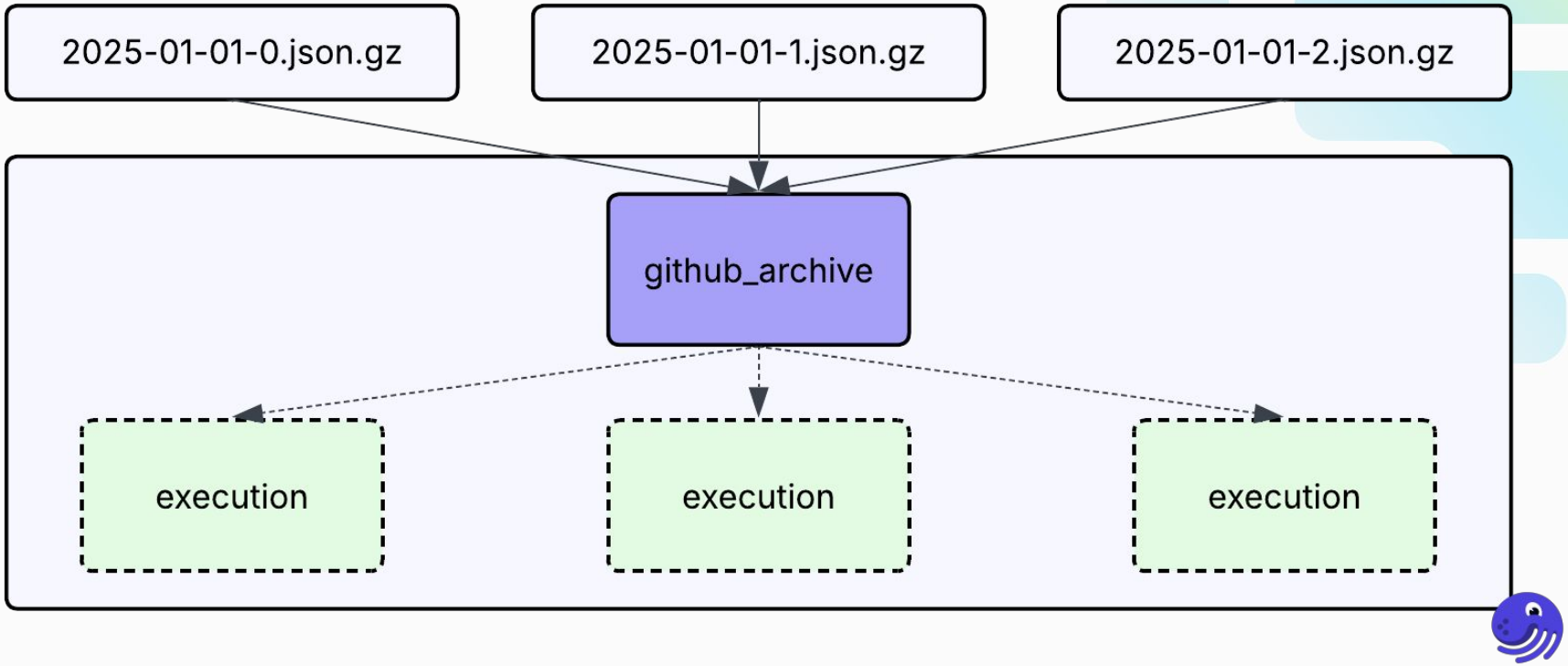
```python
@dg.asset
def github_archive_files(
    context: dg.AssetExecutionContext,
    duckdb_storage: DuckDBResource,
) -> str:
    with duckdb_storage.get_connection() as conn:
        file_name = f"https://data.gharchive.org/2025-10-01-0.json.gz"
        conn.execute(
            f"""
            copy (
                select * from read_json_auto('{file_name}',
                    ignore_errors=true,
                    union_by_name=true,
                    sample_size=-1
                )
                where type = 'IssuesEvent'
                and payload.action = 'opened'
            ) to 'data/raw/{file_name}.parquet' (format parquet)
            """
        )
    context.log.info(f"Downloaded - {file_name}")
```



DEMO

```
> dg check defs
```


2025-01-01-0.json.gz
2025-01-01-1.json.gz
2025-01-01-2.json.gz
ga_2025_01_01_0
ga_2025_01_01_1
ga_2025_01_01_2
execution
execution
execution


2025-01-01-0.json.gz
2025-01-01-1.json.gz
2025-01-01-2.json.gz
github_archive
execution
execution
execution

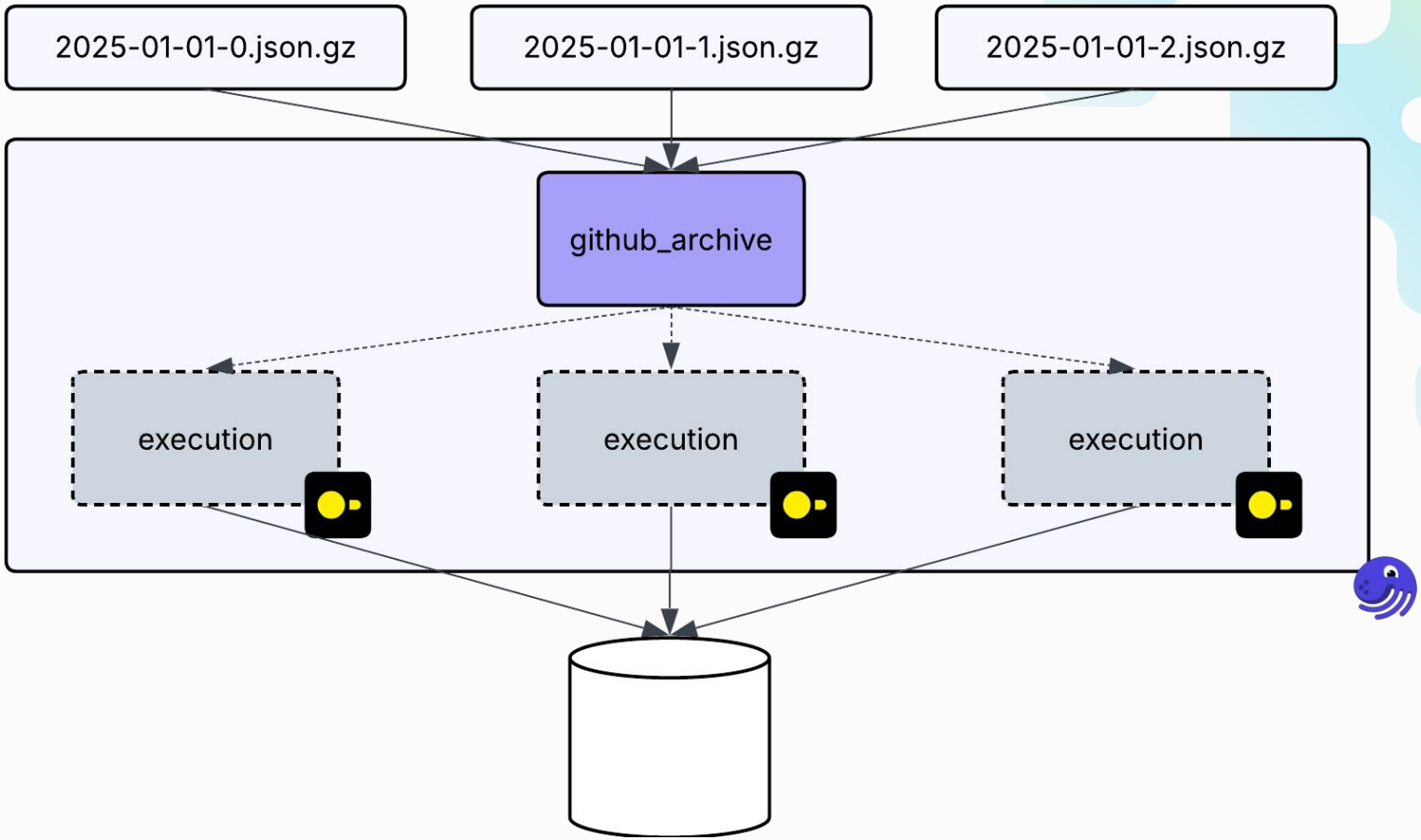
What is a partition

A Dagster **partition** allows you to divide your assets into subsets based on:

- time
- categories
- other dimensions

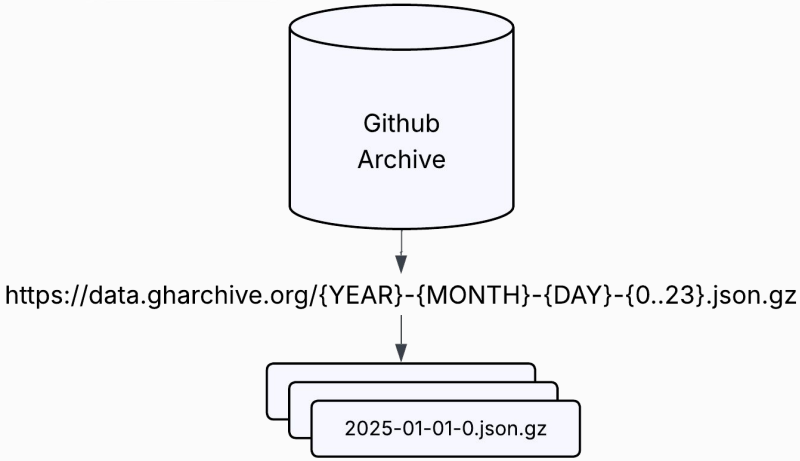
When an asset is partitioned, it is still represented by a single asset in the asset graph but each partition is maintained and tracked independently.

```
daily_partitions = dg.DailyPartitionsDefinition(
    start_date="2024-01-01"
)
```


2025-01-01-0.json.gz
2025-01-01-1.json.gz
2025-01-01-2.json.gz
github_archive
execution
execution
execution

Partitioning our data

The Github Archive data is already partitioned on their end at the hourly level. So we can use the Dagster **HourlyPartitionsDefinition** to match that partitioning strategy.



Our partition

```
1 daily_partitions_def = dg.HourlyPartitionsDefinition(
2     start_date="2025-10-01-00:00+0000",
3     end_date="2025-10-02-00:00+0000",
4 )
```

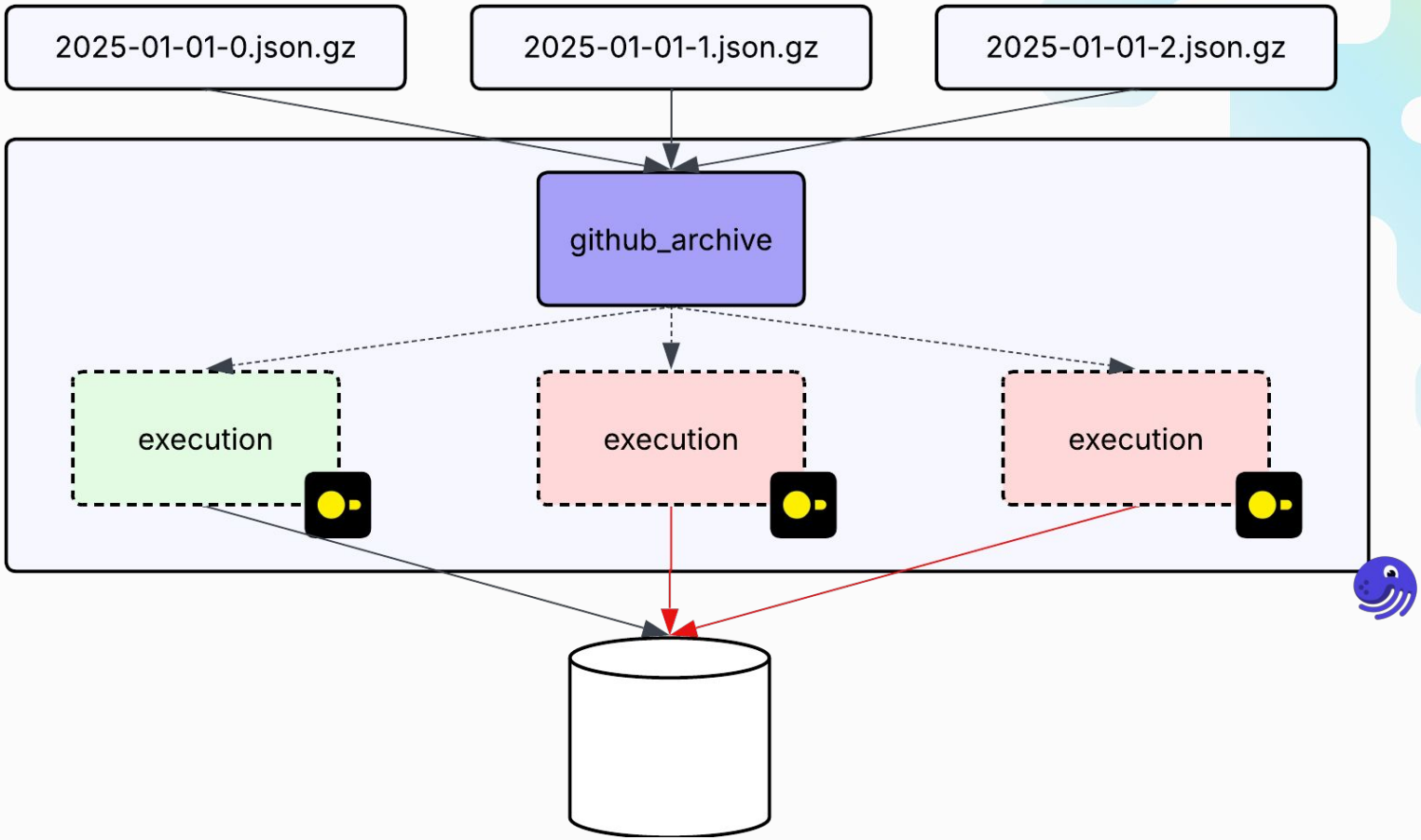
* This variable should really be called `hourly_partitions_defs`

Our partition asset

```python
@dg.asset(
    partitions_def=daily_partitions_def,
)
def github_archive_files(
    context: dg.AssetExecutionContext,
    duckdb_storage: DuckDBResource,
) -> str:
    partition_date = datetime.strptime(context.partition_key, "%Y-%m-%d-%H:%M")
    url_partition = partition_date.strftime("%Y-%m-%d") + f"-{partition_date.hour}"

    if os.path.exists(f"data/raw/{url_partition}.parquet"):
        context.log.info(f"File already exists - {url_partition}")
    else:
        with duckdb_storage.get_connection() as conn:
            gharchive_url = f"https://data.gharchive.org/{url_partition}.json.gz"
            conn.execute(
                f"""
                copy (
                    select * from read_json_auto('{gharchive_url}',
                        ignore_errors=true,
                        union_by_name=true,
                        sample_size=-1
                    )
                    where type = 'IssuesEvent'
                    and payload.action = 'opened'
                ) to 'data/raw/{url_partition}.parquet' (format parquet)
                """
            )
        context.log.info(f"Downloaded - {url_partition}")

    return url_partition
```

Our partition asset

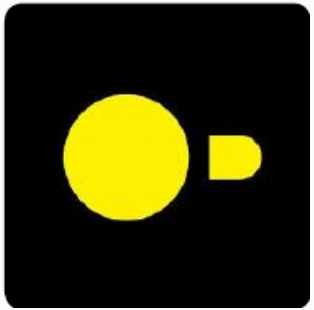
```python
@dg.asset(
    partitions_def=daily_partitions_def,
)
def github_archive_files(
    context: dg.AssetExecutionContext,
    duckdb_storage: DuckDBResource,
) -> str:
    partition_date = datetime.strptime(context.partition_key, "%Y-%m-%d-%H:%M")
    url_partition = partition_date.strftime("%Y-%m-%d") + f"-{partition_date.hour}"

    if os.path.exists(f"data/raw/{url_partition}.parquet"):
        context.log.info(f"File already exists - {url_partition}")
    else:
        with duckdb_storage.get_connection() as conn:
            gharchive_url = f"https://data.gharchive.org/{url_partition}.json.gz"
            conn.execute(
                f"""
                copy (
                    select * from read_json_auto('{gharchive_url}',
                        ignore_errors=true,
                        union_by_name=true,
                        sample_size=-1
                    )
                    where type = 'IssuesEvent'
                    and payload.action = 'opened'
                ) to 'data/raw/{url_partition}.parquet' (format parquet)
                """
            )
        context.log.info(f"Downloaded - {url_partition}")

    return url_partition
```


2025-01-01-0.json.gz
2025-01-01-1.json.gz
2025-01-01-2.json.gz
github_archive
execution
execution
execution

Why Dagster loves DuckDB

Both Dagster and DuckDB are designed for analytical workloads and modern data engineering.

They both can run almost anywhere and are meant to run in parallel.

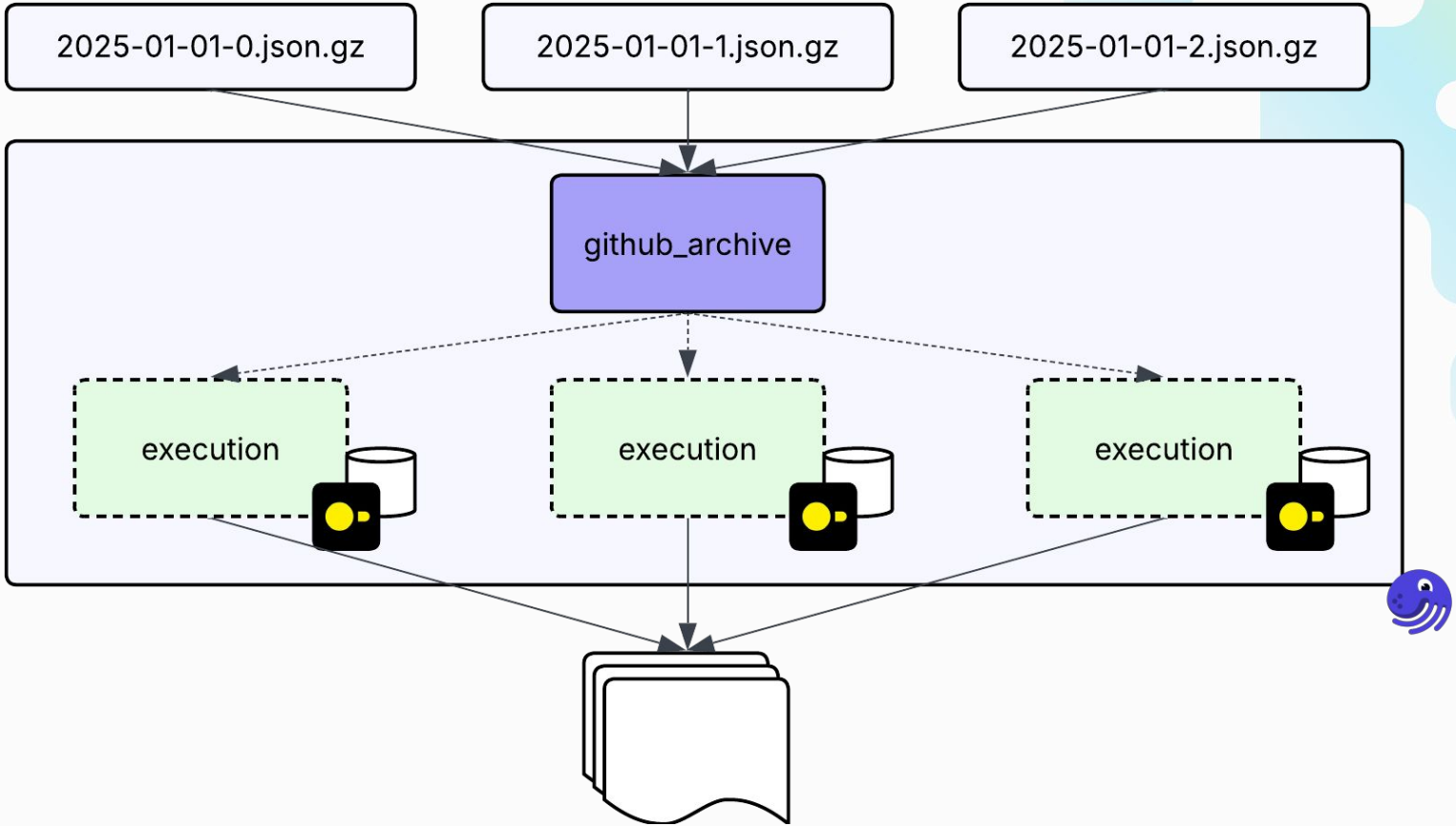


Additional resources

```python
database_resource_memory = DuckDBResource(database=":memory:")
database_resource_storage = DuckDBResource(database="workshop.duckdb")


@dg.definitions
def resources():
    return dg.Definitions(
        resources={
            "duckdb_memory": database_resource_memory,
            "duckdb_storage": database_resource_storage,
        }
    )
```

Update our asset

```python
@dg.asset(
    partitions_def=daily_partitions_def,
    kinds={"duckdb"},
    group_name="duckdb_memory",
)
def github_archive_files(
    context: dg.AssetExecutionContext,
    duckdb_memory: DuckDBResource,
) -> str:
    partition_date = datetime.strptime(context.partition_key, "%Y-%m-%d-%H:%M")
    url_partition = partition_date.strftime("%Y-%m-%d") + f"-{partition_date.hour}"

    if os.path.exists(f"data/raw/{url_partition}.parquet"):
        context.log.info(f"File already exists - {url_partition}")
    else:
        with duckdb_memory.get_connection() as conn:
            gharchive_url = f"https://data.gharchive.org/{url_partition}.json.gz"
            conn.execute(
                f"""
                copy (
                    select * from read_json_auto('{gharchive_url}',
                        ignore_errors=true,
                        union_by_name=true,
                        sample_size=-1
                    )
                    where type = 'IssuesEvent'
                    and payload.action = 'opened'
                ) to 'data/raw/{url_partition}.parquet' (format parquet)
                """
            )
        context.log.info(f"Downloaded - {url_partition}")

    return url_partition
```

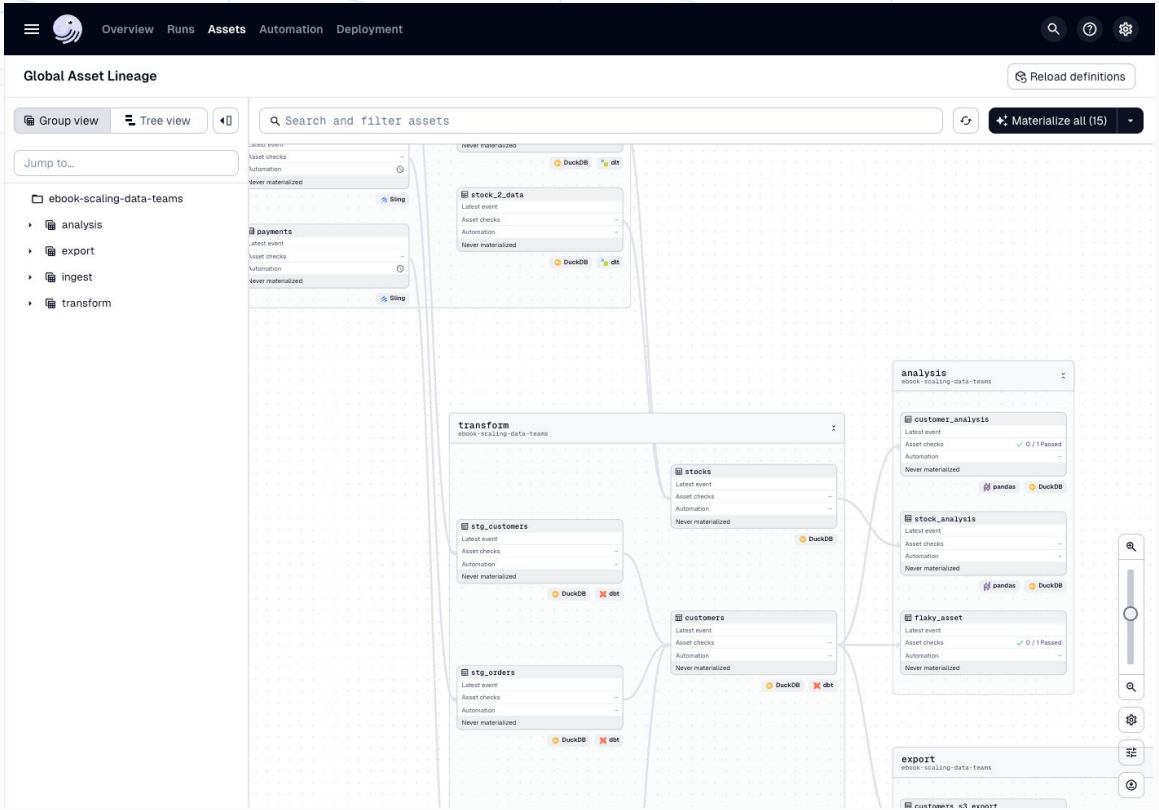

2025-01-01-0.json.gz
2025-01-01-1.json.gz
2025-01-01-2.json.gz
github_archive
execution
execution
execution

DEMO

> dg dev (if not already running)

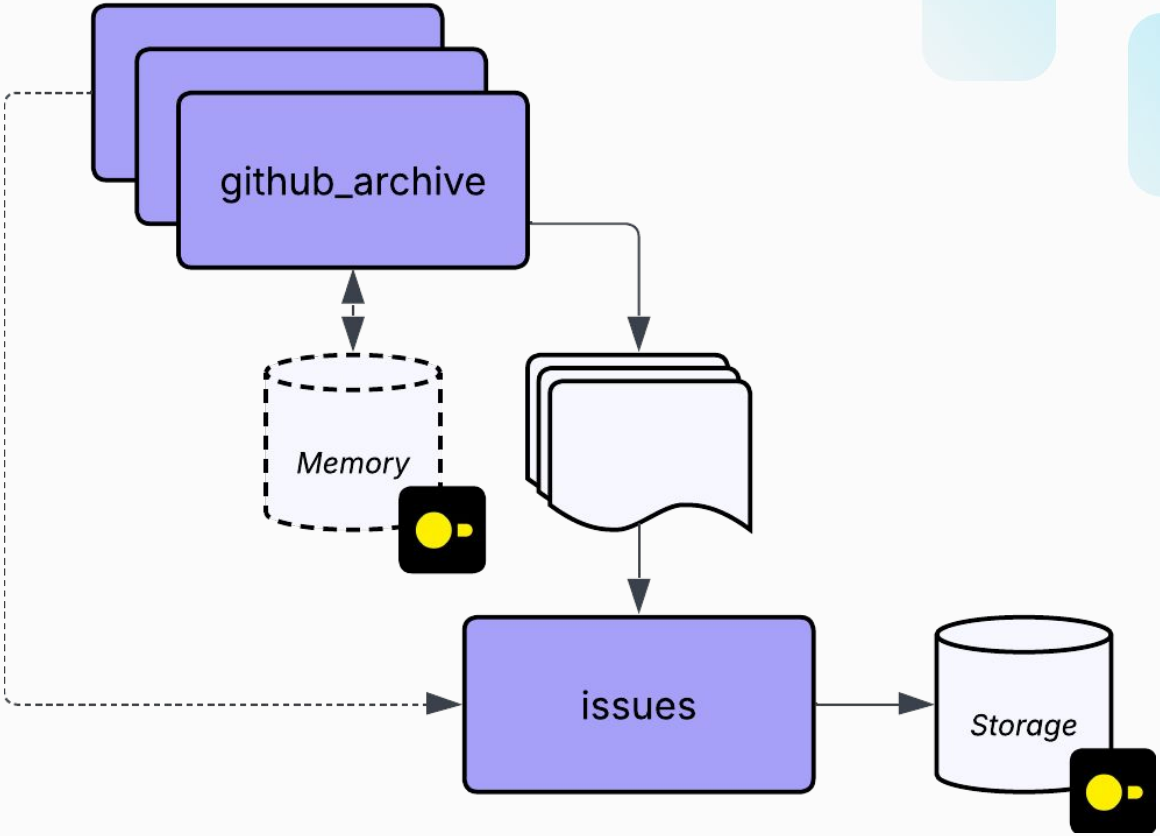
2. Model Github Issues

Downstream assets



Our downstream asset

```python
@dg.asset(
    kinds={"duckdb"},
    deps=[github_archive_files],
    group_name="duckdb_storage",
    backfill_policy=dg.BackfillPolicy.single_run(),
)
def issues(
    context: dg.AssetExecutionContext,
    duckdb_storage: DuckDBResource,
):
    with duckdb_storage.get_connection() as conn:
        conn.execute(
            f"""
                create or replace table {ISSUES_TABLE_NAME} as
                select
                    id,
                    org.login as org,
                    payload.issue.html_url as url,
                    payload.issue.user.login as author,
                    payload.issue.created_at as created_at,
                    payload.issue.title as title,
                    payload.issue.body as body
                from 'data/raw/*.parquet'
            """
        )
```


github_archive
Memory
issues
Storage

DEMO

Dagster
> dg dev (if not already running)

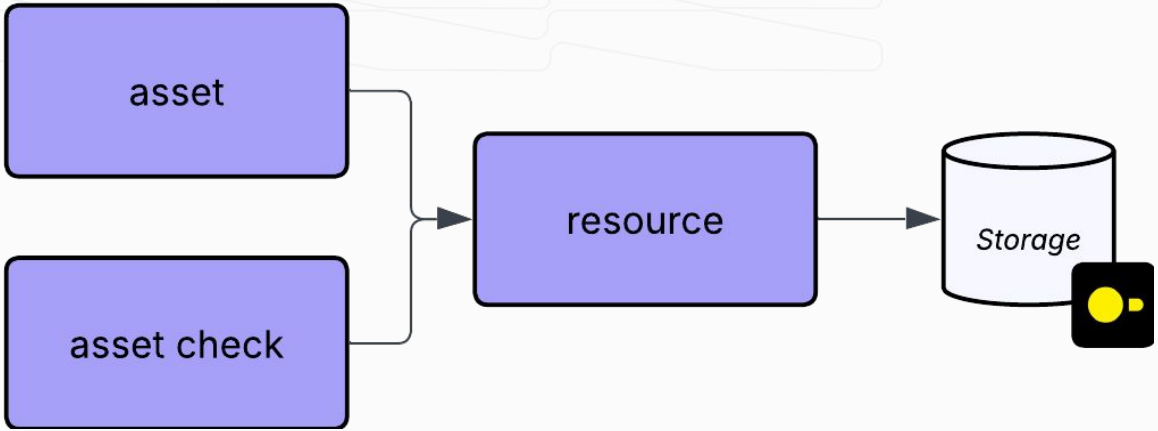
DuckDB
> duckdb workshop.duckdb
> SELECT * FROM github_issues LIMIT 10;

3. Monitor Data Quality

Our asset check

```python
@dg.asset_check(
    asset=issues,
)
def github_events_check(
    duckdb_storage: DuckDBResource,
):
    with duckdb_storage.get_connection() as conn:
        events = conn.execute(
            f"""select count(*) from {ISSUES_TABLE_NAME}"""
        ).fetchall()
    return dg.AssetCheckResult(passed=len(events) > 0)
```

checks and resources

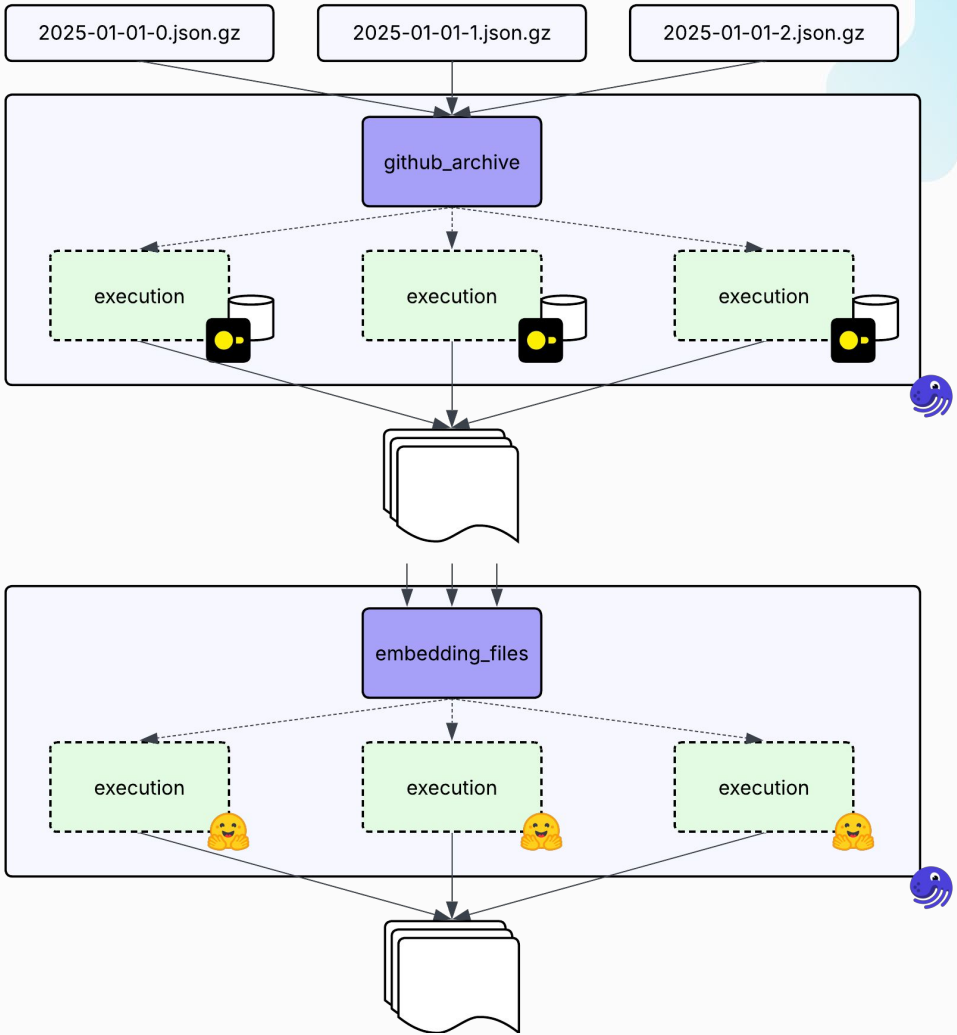


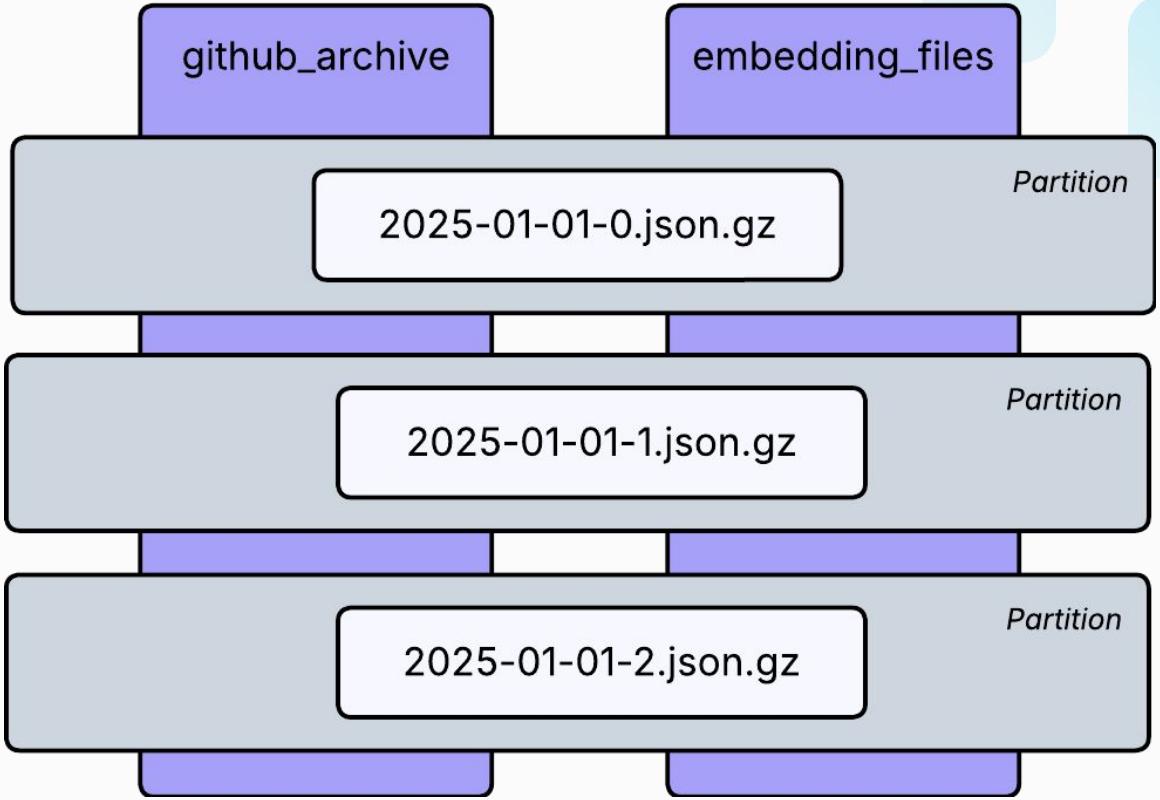
DEMO

Dagster

> dg dev (if not already running)

4. Generate Embeddings


2025-01-01-0.json.gz
2025-01-01-1.json.gz
2025-01-01-2.json.gz
github_archive
execution
execution
execution
embedding_files
execution
execution
execution


github_archive
embedding_files
Partition
2025-01-01-0.json.gz
Partition
2025-01-01-1.json.gz
Partition
2025-01-01-2.json.gz

```python
@dg.asset(
    kinds={"duckdb", "huggingface"},
    deps=[github_archive_files],
    group_name="duckdb_memory",
    partitions_def=daily_partitions_def,
)
def embedding_files(
    context: dg.AssetExecutionContext,
    duckdb_memory: DuckDBResource,
    github_archive_files: str,
):
    if os.path.exists(f"data/embeddings/{github_archive_files}.parquet"):
        context.log.info(f"File already exists - {github_archive_files}.parquet")
    else:
        model = SentenceTransformer(MODEL_NAME, cache_folder="models")
        context.log.info("Loaded model")

        with duckdb_memory.get_connection() as conn:
            df = conn.execute(f"""
                select
                    id,
                    'Title: ' || payload.issue.title || ' Body: ' || coalesce(payload.issue.body, '') as combined_text
                from 'data/raw/{github_archive_files}.parquet'
            """).df()

            embeddings_list = model.encode(
                df["combined_text"].tolist(), show_progress_bar=True
            )

            df["embedding"] = [embedding.tolist() for embedding in embeddings_list]
            context.log.info(
                f"Generated embeddings with dimension: {len(embeddings_list[0])}"
            )

            conn.execute(f"""
                copy (
                    select
                        id,
                        embedding::FLOAT[{EMBEDDING_DIMENSION}] as embedding
                    from df
                ) to 'data/embeddings/{github_archive_files}.parquet' (format parquet)
            """)
```

Creating embeddings

Combined text	Title: Login button throws 500 Body: Clicking the login button on mobile returns a 500 error after OAuth redirect.
Turn text to list of strings	["Title:", "Login", "button", "throws", "500", "Body:", "Clicking", "the", "login",...]
Token ids	[102, 394, 341, 233, 844, 932, 754, 324, 235, ...]
Embeddings	[.91, -.322, .223, .256, -.425, .975, .564, -.424, .809, ..., n383]

DEMO

Dagster

> dg dev (if not already running)

```python
@dg.asset(
    kinds={"duckdb"},
    deps=[embedding_files],
    group_name="duckdb_storage",
    backfill_policy=dg.BackfillPolicy.single_run(),
)
def embeddings(
    context: dg.AssetExecutionContext,
    duckdb_storage: DuckDBResource,
):
    with duckdb_storage.get_connection() as conn:
        conn.execute(
            f"""
                create or replace table {EMBEDDINGS_TABLE_NAME} as
                select
                    id,
                    embedding::FLOAT[{EMBEDDING_DIMENSION}] as embedding
                from 'data/embeddings/*.parquet'
            """
        )
```

DuckDB isn't a vector database. But...

DuckDB Docs

⌘+k 1.4 (stable)

Installation
Documentation
- Getting Started
- Connect
- Data Import and Export
- Lakehouse Formats
- Client APIs
- SQL
- Configuration
- Extensions
- Core Extensions
  - Overview
  - AutoComplete
  - Avro
  - AWS
  - Azure

DOCUMENTATION / CORE EXTENSIONS

Vector Similarity Search Extension

The `vss` extension is an experimental extension for DuckDB that adds indexing support to accelerate vector similarity search queries using DuckDB's new fixed-size `ARRAY` type.

See the announcement blog post and the "What's New in the Vector Similarity Search Extension?" post.

Usage

To create a new HNSW (Hierarchical Navigable Small Worlds) index on a table with an `ARRAY` column, use the `CREATE INDEX` statement with the `USING HNSW` clause. For example:

```
INSTALL vss;
LOAD vss;

CREATE TABLE my_vector_table (vec FLOAT[3]);
INSERT INTO my_vector_table
    SELECT array_value(a, b, c)
```

IN THIS ARTICLE

Usage
Index Options
Persistence
Inserts, Updates, Deletes a...
Bonus: Vector Similarity Se...
Limitations

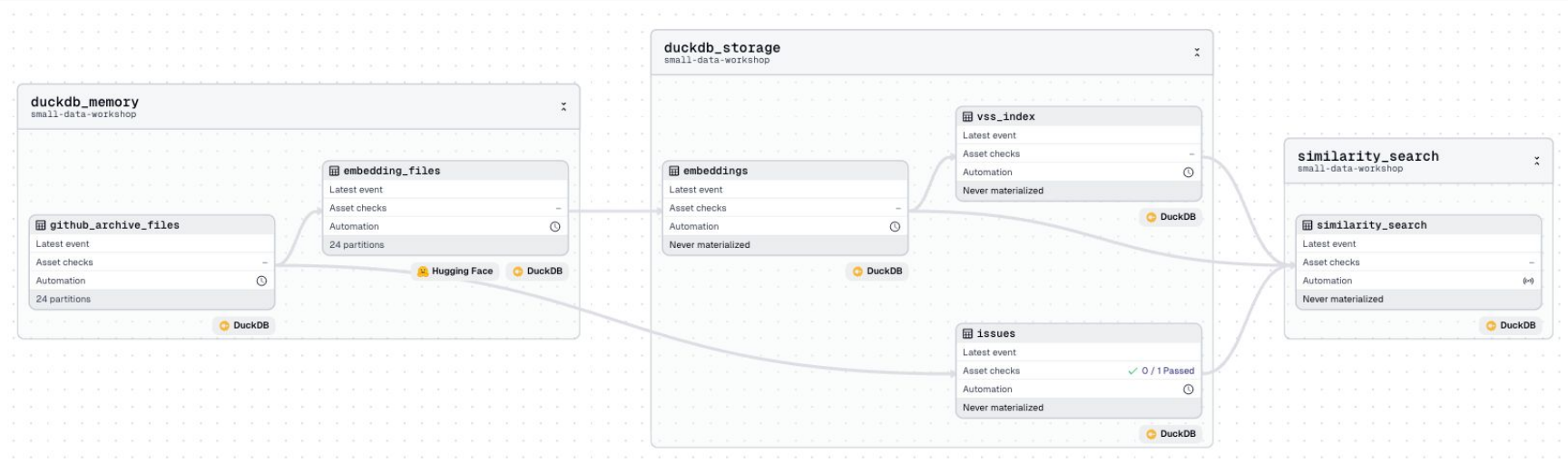
```python
@dg.asset(
    kinds={"duckdb"},
    deps=[embeddings],
    group_name="duckdb_storage",
)
def vss_index(
    context: dg.AssetExecutionContext,
    duckdb_storage: DuckDBResource,
):
    with duckdb_storage.get_connection() as conn:
        # Install and load the VSS extension
        conn.execute("INSTALL vss")
        conn.execute("LOAD vss")

        # Enable experimental persistence for the HNSW index
        conn.execute("SET hnsw_enable_experimental_persistence = true")

        # Create HNSW index on the embedding column
        conn.execute(f"""drop index if exists {INDEX_NAME}""")

        context.log.info("Creating HNSW index")
        conn.execute(f"""
            create index if not exists {INDEX_NAME}
                on {EMBEDDINGS_TABLE_NAME}
                using hnsw (embedding)
                with (m = 16, ef_construction = 200)
        """)
```

DEMO

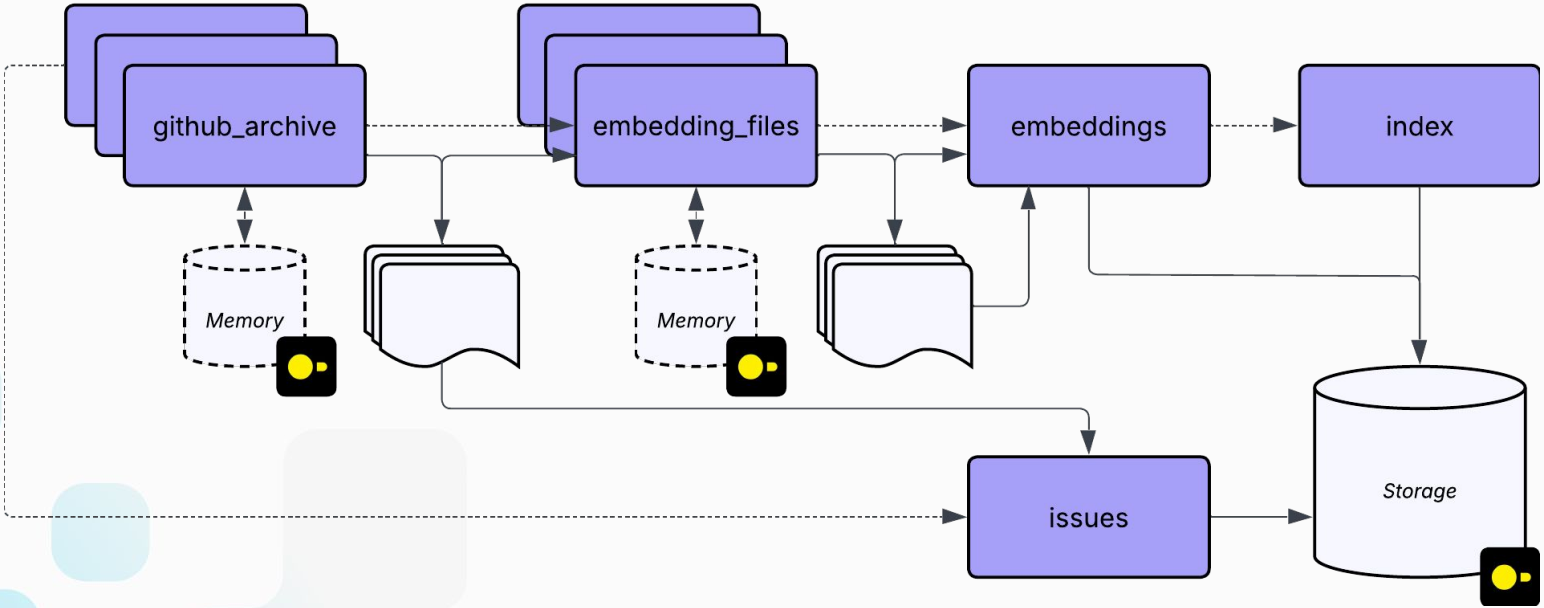
Dagster

> dg dev (if not already running)

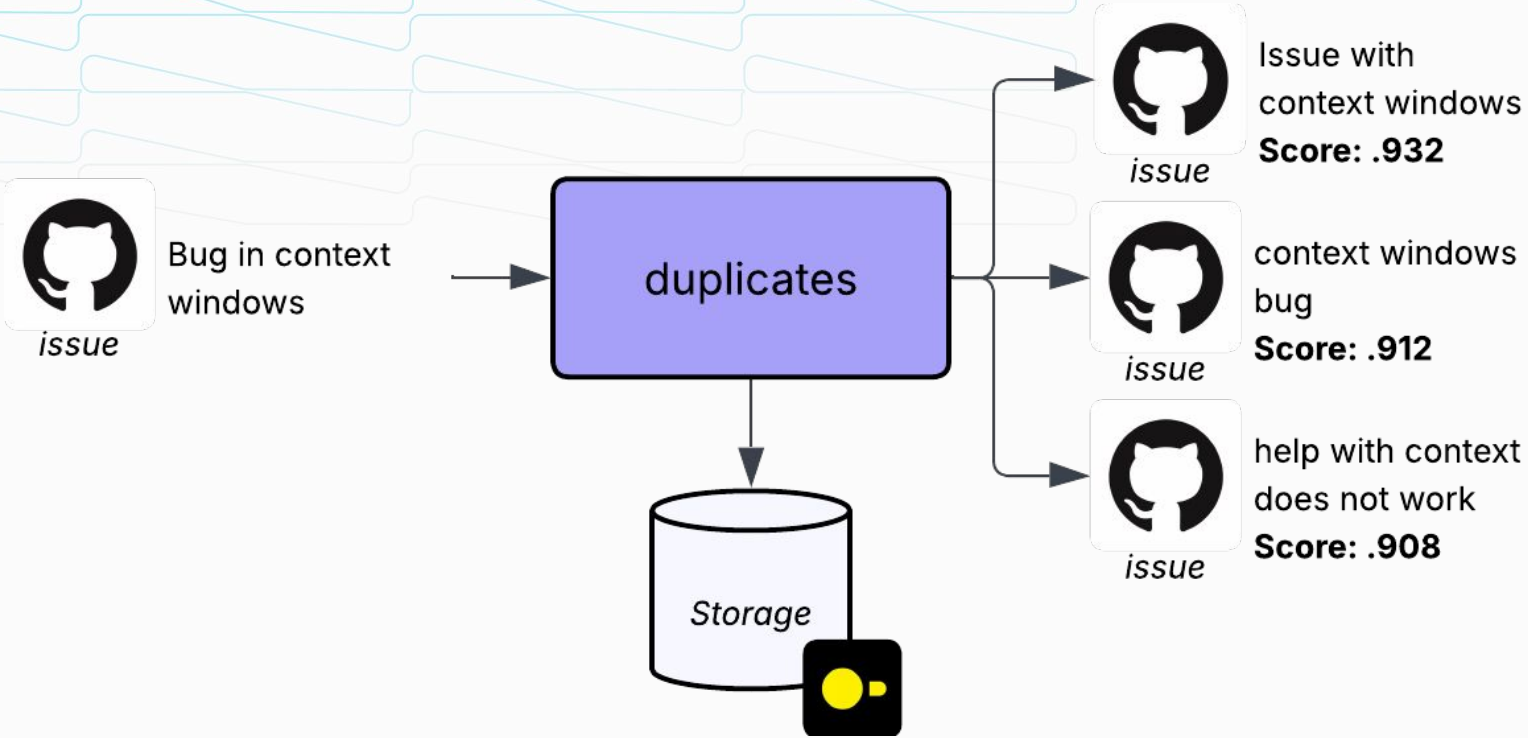
Hiding Complexity



Hiding Complexity



5. Duplication Search


issue
Bug in context windows
duplicates
Storage
issue
Issue with context windows
Score: .932
issue
context windows bug
Score: .912
issue
help with context does not work
Score: .908

Parameterizing execution

Configs allow you to parameterize assets at runtime, making them flexible and reusable without changing the core code.

```
class GithubIssue(dg.Config):
    id: int
    similarity_score: float = .75
    num_results: int = 5
```

Query issue data

Query the issues table based on the configuration value of the id for our asset.

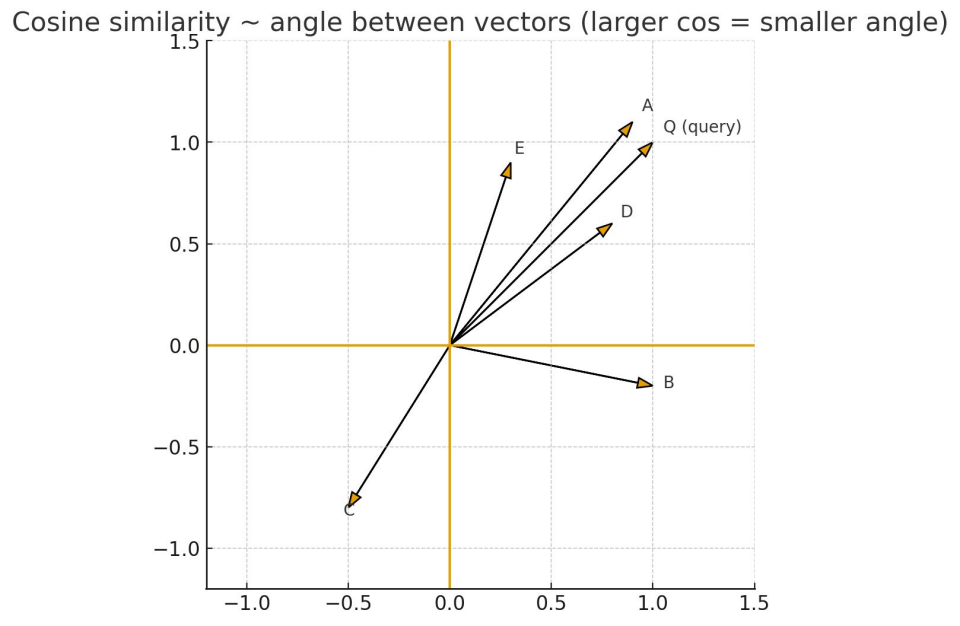
```
1 select
2     title,
3     author,
4     url,
5 from {ISSUES_TABLE_NAME}
6 where id = {config.id}
```

Query similar issues

Using the DuckDB function **array_cosine_similarity**. Find the n nearest issues based on the similarity of the embeddings.

```
select
    ie.id,
    i.url,
    array_cosine_similarity(ie.embedding, q.query_vec) as similarity
from
    {EMBEDDINGS_TABLE_NAME} as ie,
    (
        select
            embedding as query_vec
        from {EMBEDDINGS_TABLE_NAME}
        where id = {config.id}
    ) as q
join {ISSUES_TABLE_NAME} as i
    on ie.id = i.id
where ie.id != {config.id}
and array_cosine_similarity(ie.embedding, q.query_vec) > {config.similarity_score}
order by array_cosine_similarity(ie.embedding, q.query_vec) desc
limit {config.num_results}
```

Query similar issues



```python
@dg.asset(
    kinds={"duckdb"},
    deps=[embeddings, vss_index, issues],
    group_name="similarity_search",
)
def similarity_search(
    context: dg.AssetExecutionContext,
    config: GithubIssue,
    duckdb_storage: DuckDBResource,
) -> dg.MaterializeResult:
    with duckdb_storage.get_connection() as conn:
        # Issue data
        issue_data = conn.execute(f"""
            select
                title,
                author,
                url,
            from {ISSUES_TABLE_NAME}
            where id = {config.id}
        """).fetchone()

        # Similar issues
        similar_issues = conn.execute(f"""
            select
                ie.id,
                i.url,
                array_cosine_similarity(ie.embedding, q.query_vec) as similarity
            from
                {EMBEDDINGS_TABLE_NAME} as ie,
                (
                    select
                        embedding as query_vec
                    from {EMBEDDINGS_TABLE_NAME}
                    where id = {config.id}
                ) as q
            join {ISSUES_TABLE_NAME} as i
                on ie.id = i.id
            where ie.id != {config.id}
            and array_cosine_similarity(ie.embedding, q.query_vec) > {config.similarity_score}
            order by array_cosine_similarity(ie.embedding, q.query_vec) desc
            limit {config.num_results};
        """).fetchall()

        return dg.MaterializeResult(
            metadata={
                "id": config.id,
                "title": issue_data[0],
                "author": issue_data[1],
                "url": dg.MetadataValue.url(issue_data[2]),
                "similar_issues": similar_issues,
            }
        )
```

```python
@dg.asset(
    kinds={"duckdb"},
    deps=[embeddings, vss_index, issues],
    group_name="similarity_search",
)
def similarity_search(
    context: dg.AssetExecutionContext,
    config: GithubIssue,
    duckdb_storage: DuckDBResource,
) -> dg.MaterializeResult:
    with duckdb_storage.get_connection() as conn:
        # Issue data
        issue_data = conn.execute(f"""
            select
                title,
                author,
                url,
            from {ISSUES_TABLE_NAME}
            where id = {config.id}
        """).fetchone()

        # Similar issues
        similar_issues = conn.execute(f"""
            select
                ie.id,
                i.url,
                array_cosine_similarity(ie.embedding, q.query_vec) as similarity
            from
                {EMBEDDINGS_TABLE_NAME} as ie,
                (
                    select
                        embedding as query_vec
                    from {EMBEDDINGS_TABLE_NAME}
                    where id = {config.id}
                ) as q
            join {ISSUES_TABLE_NAME} as i
                on ie.id = i.id
            where ie.id != {config.id}
            and array_cosine_similarity(ie.embedding, q.query_vec) > {config.similarity_score}
            order by array_cosine_similarity(ie.embedding, q.query_vec) desc
            limit {config.num_results};
        """).fetchall()

        return dg.MaterializeResult(
            metadata={
                "id": config.id,
                "title": issue_data[0],
                "author": issue_data[1],
                "url": dg.MetadataValue.url(issue_data[2]),
                "similar_issues": similar_issues,
            }
        )
```

Issue Query

Similarity Query

Dagster Metadata

In Dagster, metadata allows you to attach information to assets such as Markdown descriptions, table schemas, or time series data which be associated with the execution of an asset by returning **MaterializeResult**.

```
return dg.MaterializeResult(
    metadata={
        "id": config.id,
        "title": issue_data[0],
        "author": issue_data[1],
        "url": dg.MetadataValue.url(issue_data[2]),
        "similar_issues": similar_issues,
    }
)
```

DEMO

DuckDB

> duckdb workshop.duckdb
> SELECT * FROM github_issues WHERE org = '{ORG NAME}';

Suggestions

- Try your company
- anthropics
- openai
- google

Dagster

dg dev

6. Enable Automation

schedules

Schedules enable automated execution of jobs at specified intervals, ranging from common frequencies to more complex custom cron expressions.

```
1 @dg.schedule(cron_schedule="0 0 * * *", target="*")
2 def my_schedule(context: dg.ScheduleEvaluationContext):
3     pass
```

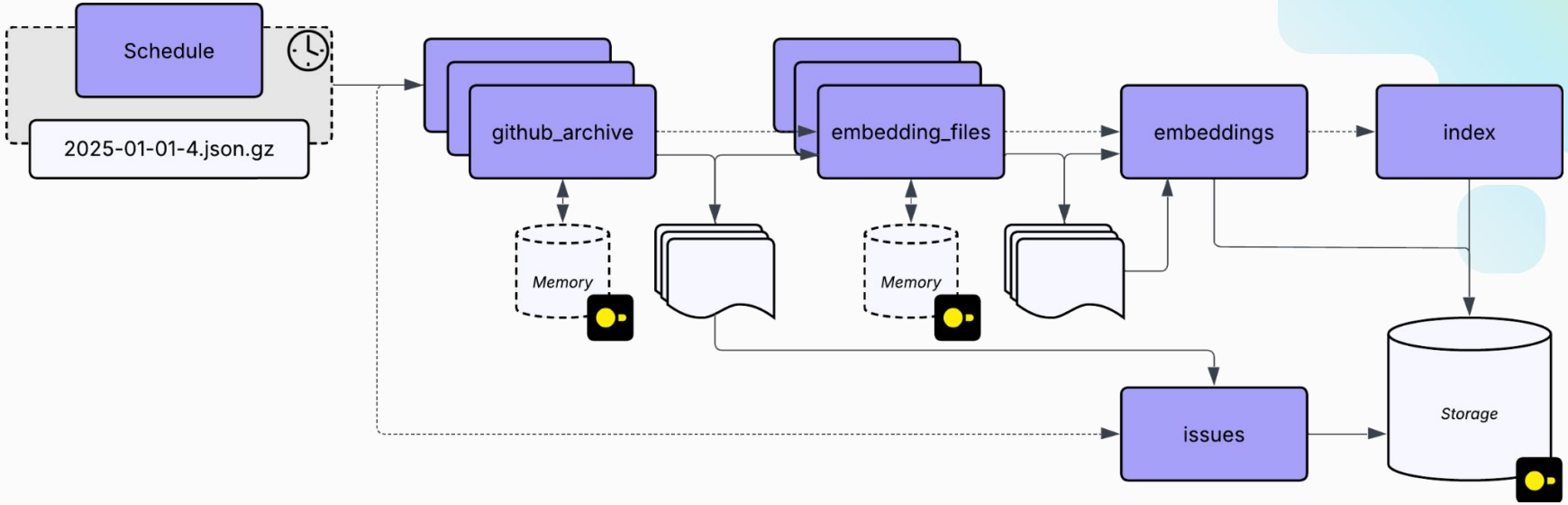
partition schedule

This requires a special schedule since it is scheduling assets with partitions.

We can automatically create the schedule based on the partition using **build_schedule_from_partitioned_job**.

```
ingestion_job = dg.define_asset_job(
    name="ingestion_job",
    selection=[
        github_archive_files,
        issues,
        embedding_files,
        embeddings,
        vss_index,
    ],
)
```

```
asset_partitioned_schedule = dg.build_schedule_from_partitioned_job(
    ingestion_job,
)
```


Schedule
2025-01-01-4.json.gz
github_archive
embedding_files
embeddings
index
Memory
Memory
issues
Storage

DEMO

Dagster

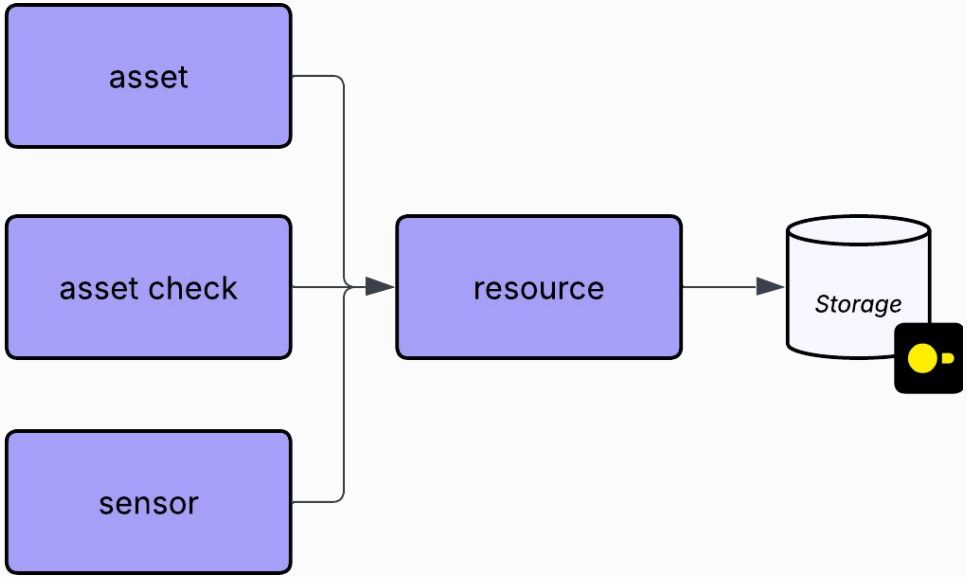
> dg dev (if not already running)

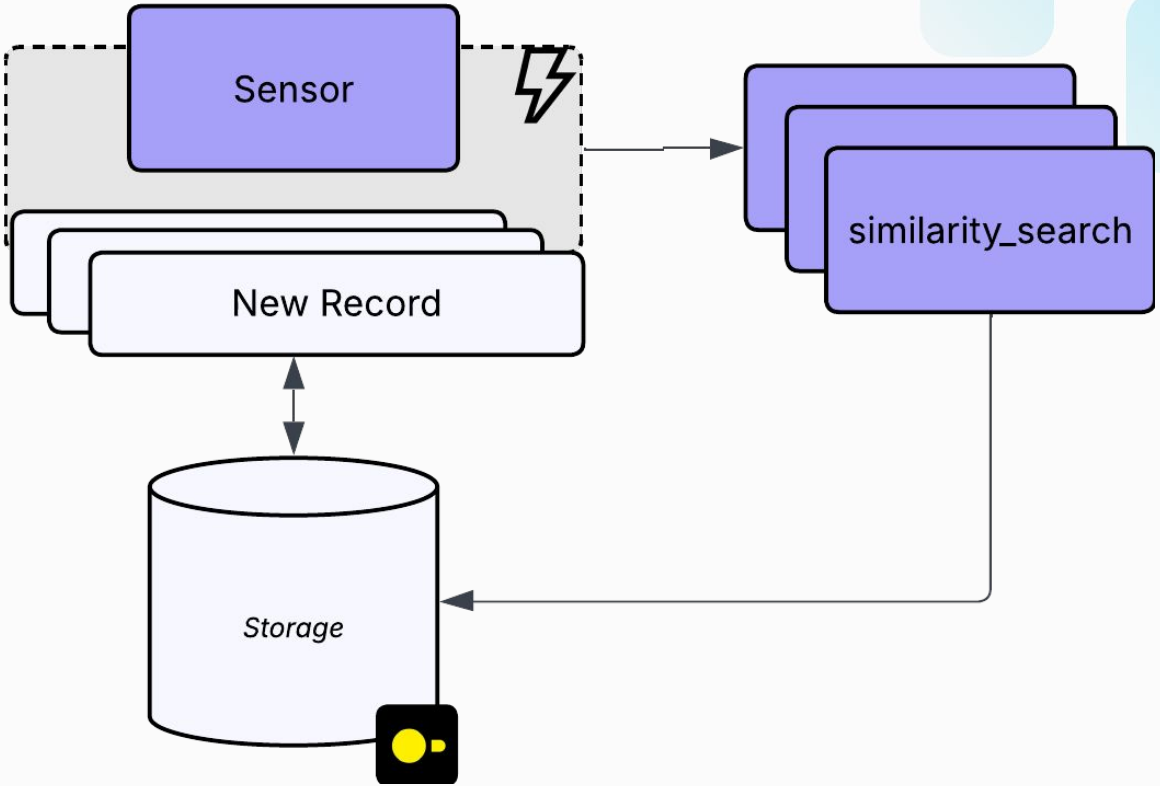
Sensors

Sensors enable you to trigger executions in response to events that occur either within Dagster or an external system.

```python
@dg.sensor(
    job=my_job,
    minimum_interval_seconds=5,
    default_status=dg.DefaultSensorStatus.RUNNING,  # Sensor is turned on by default
)
def new_file_sensor():
    new_files = check_for_new_files()
    # New files, run `my_job`
    if new_files:
        for filename in new_files:
            yield dg.RunRequest(run_key=filename)
    # No new files, skip the run and log the reason
    else:
        yield dg.SkipReason("No new files found")
```

sensors and resources



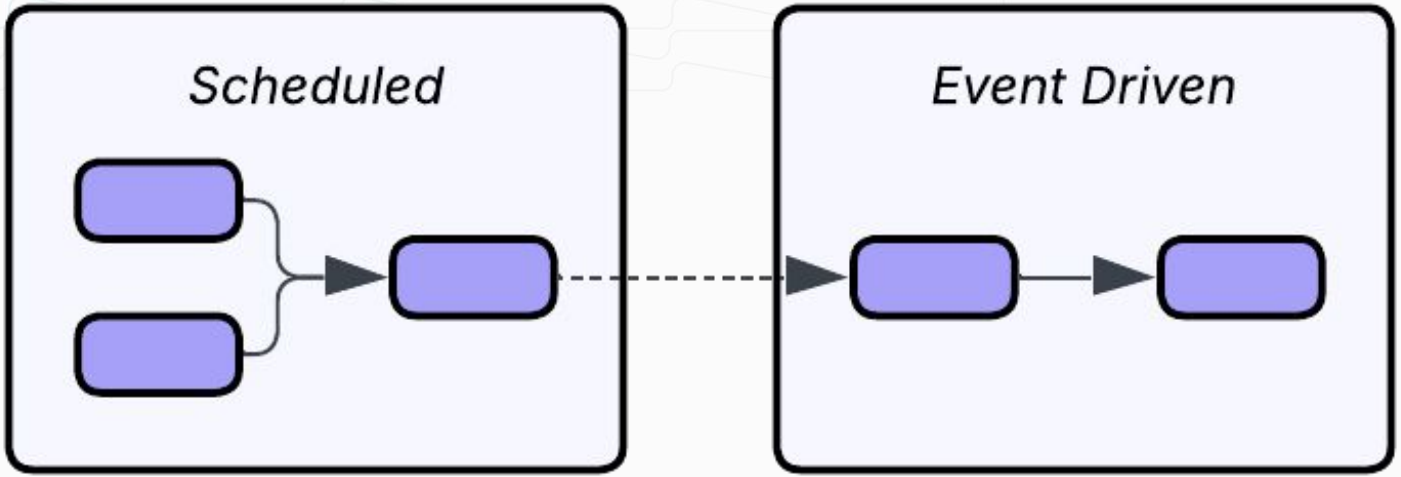

Sensor
New Record
similarity_search
Storage

```python
@dg.sensor(
    job_name="similarity_search_job",
    minimum_interval_seconds=60,
)
def new_issues_sensor(
    context: dg.SensorEvaluationContext,
    duckdb_storage: DuckDBResource,
):
    last_issue_id = context.cursor if context.cursor else "0"

    try:
        with duckdb_storage.get_connection() as conn:
            new_issues = conn.execute(f"""
                select id
                from {ISSUES_TABLE_NAME}
                where org = '{ORG_NAME}'
                and id > '{last_issue_id}'
            """).fetchall()

            if len(new_issues) > 0:
                context.log.info(f"Found {len(new_issues)} new issues")
            else:
                context.log.info("No new issues found")

        for issue in new_issues:
            issue_id = issue[0]

            # Update cursor to the highest issue ID
            if int(issue_id) > int(last_issue_id):
                last_issue_id = issue_id

            context.update_cursor(str(last_issue_id))

            yield dg.RunRequest(
                run_key=str(issue_id),
                run_config={
                    "ops": {
                        "similarity_search": {
                            "config": {
                                "id": issue_id,
                                "similarity_score": 0.1,
                                "num_results": 3,
                            }
                        }
                    }
                },
            )

    except Exception as e:
        return dg.SkipReason(f"Sensor failed with error: {str(e)}")
```

DEMO

Dagster

> dg dev (if not already running)


Scheduled
Event Driven

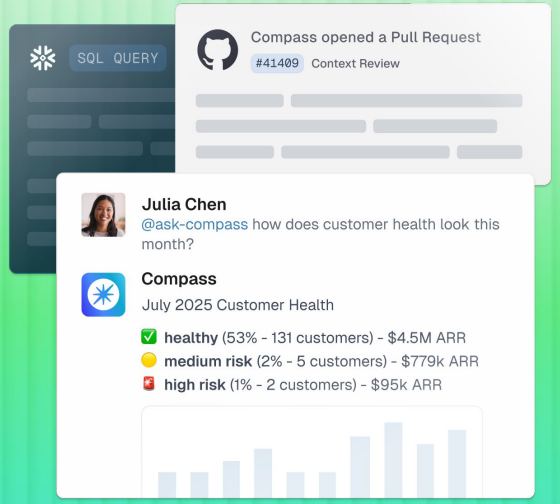
Possible Improvements

- Data Cleaning
- Deploy to Dagster+
- Full backfill
- Filtering rules
- Include AI for better deduplication
- Use personal Github feed
- Different Storage layers (like S3 instead of local storage)

Want more Dagster?

- Dagster University
- Dagster Examples
- Dagster+

compass



Join the
Dagster community Slack
to try it out!

Thanks