Building a true data platform

Beyond the Modern Data Stack

Pedram Navid

Head of Data Engineering + DevRel

Join us on Slack: dagster.io/slack in the #dagster-deep-dives channel

Agenda

The Unkept Promise of the Modern Data Stack

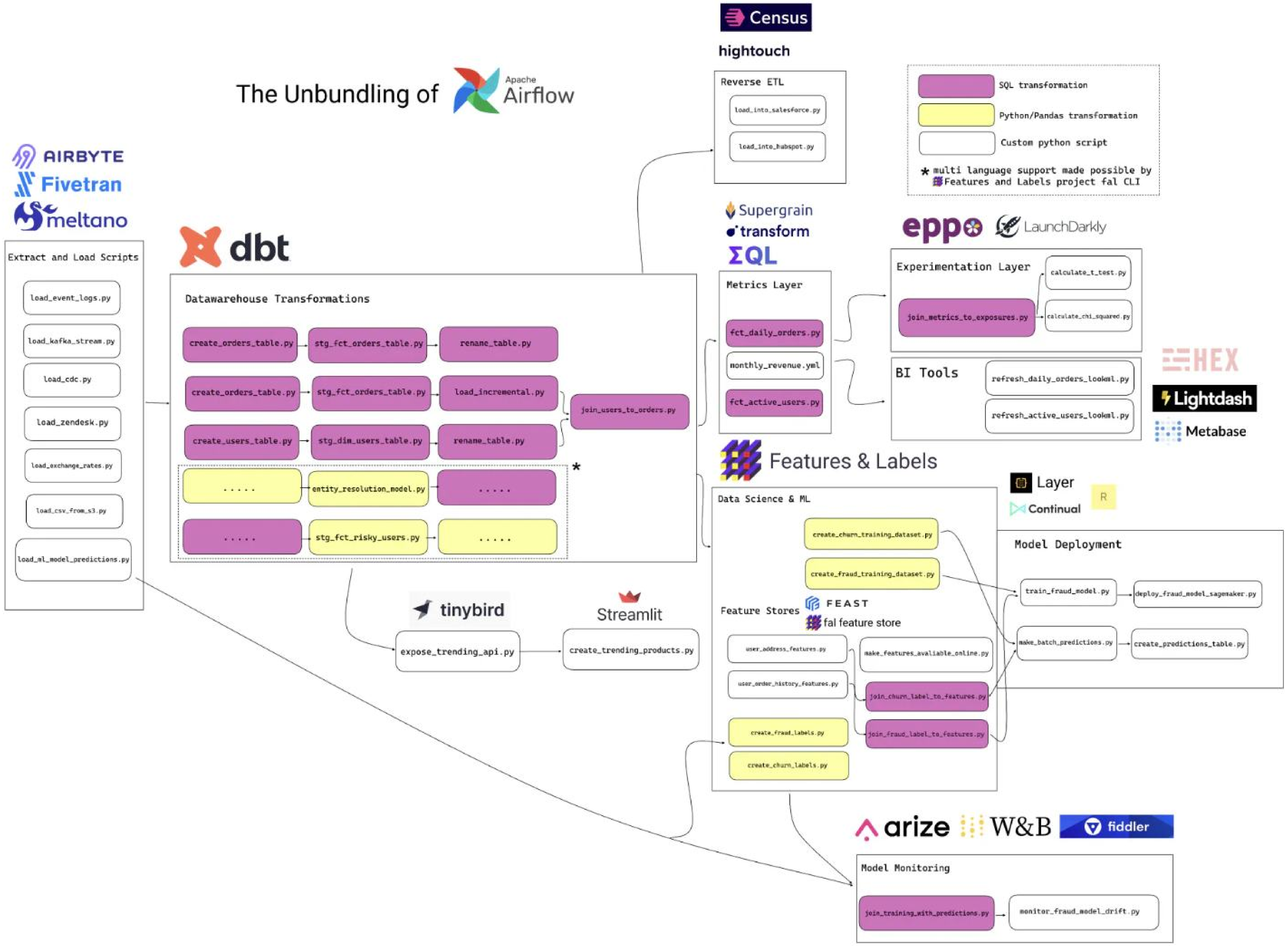
The Rise of the Data Platform Engineer

Building a Unified Data Platform

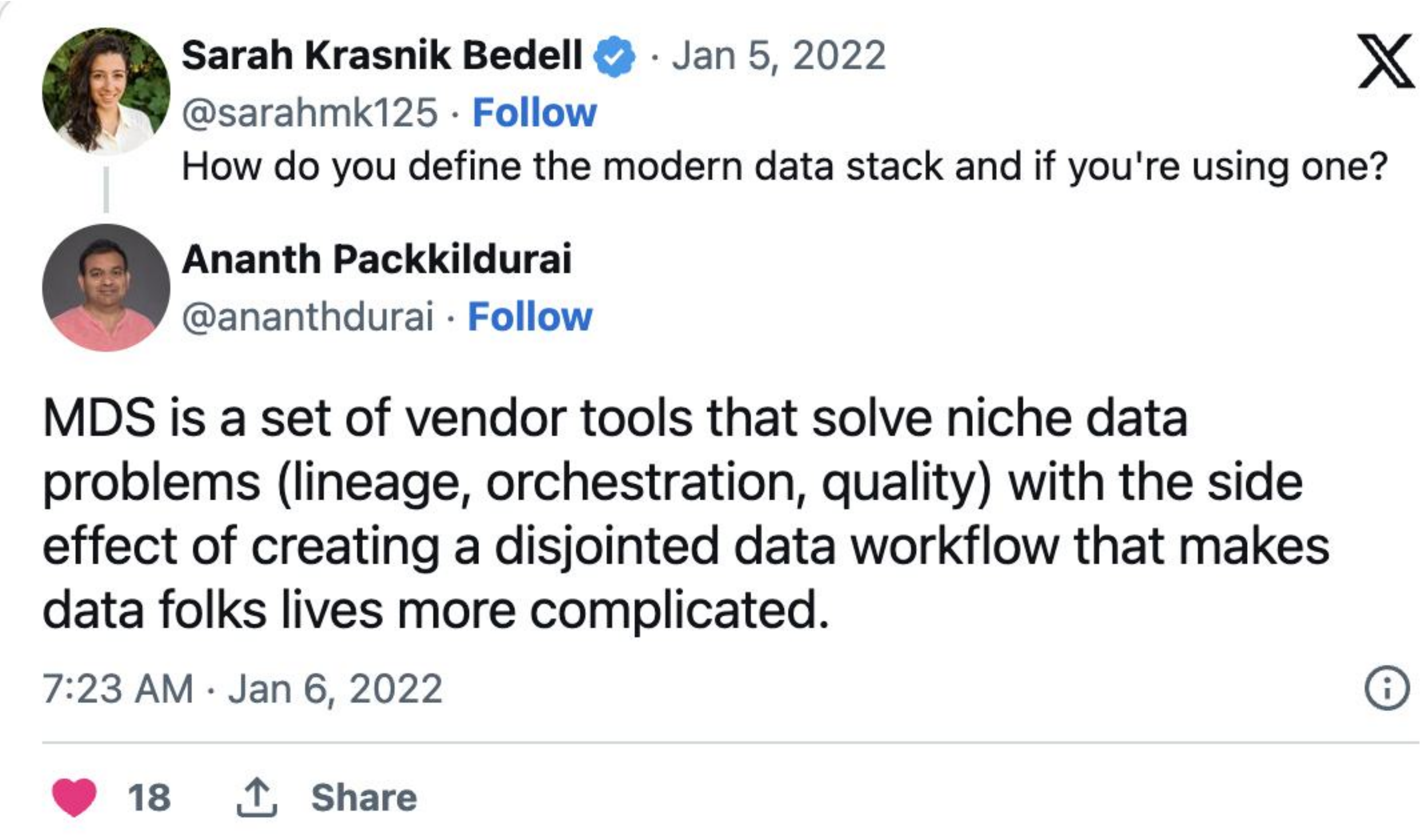
The Unkept Promise of the Modern Data Stack

"The modern data stack saves time, money and effort...leaving your analysts, data scientists and data engineers free to pursue higher-value analytics and data science projects." -Fivetran Blog

The Unbundling of Airflow



“Benefits” of the unbundled data stack



No Observability

To know the state of your data platform, you need to login to every application, click through a UI, and hope you can find what you need

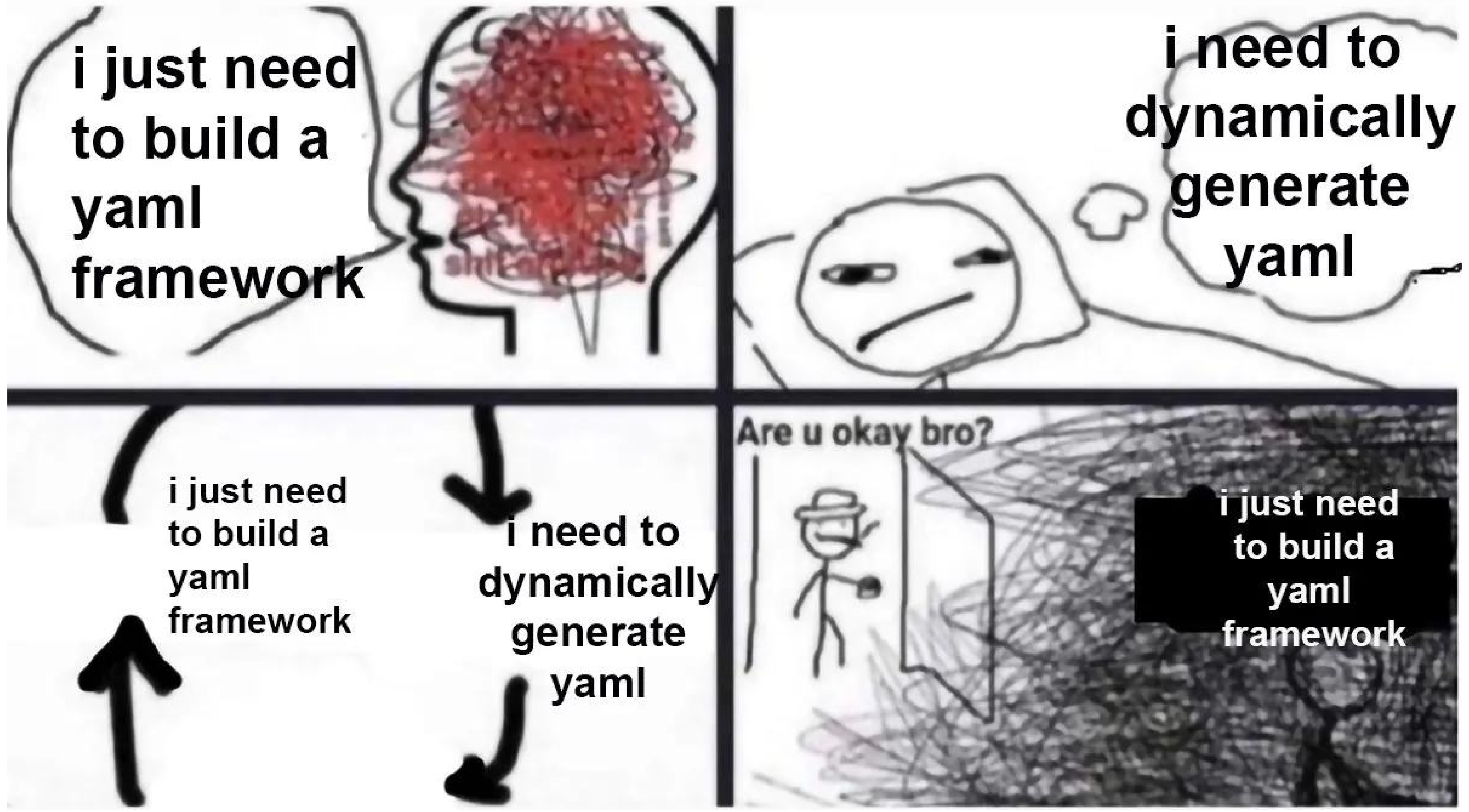
No orchestration

Insufficient integrations between the stack means you rely on cron-jobs with buffer time that you only hope will complete in time

Expensive and sticky

Difficult to migrate from one solution to another, meaning vendor lock-in, tough negotiations, and skyrocketing bills

The Rise of the Data Platform Engineer



The Unkept Promise of not writing ETL

“Data Engineers are coming back to the original sin of Data Engineering: building bespoke custom pipelines for your downstream consumers, and they’re solving it the same way we were trying to solve it 10 years ago: building platforms, frameworks, and services.”

https://databased.pedramnavid.com/p/the-rise-of-the-data-platform-engineer

The Data Platform Engineer

“The next evolution of the role is more akin to a Data Platform Engineer: someone who is tasked not with building ETL pipelines, but with making it possible for their various consumers to build any pipeline they need without having to resort to a complex higher language.”

Every organization that has data has a data platform.

Three Goals of a Data Platform

Scalabilty + Maintainability

A Data Platform must scale with your data maturity, while remaining maintainable, reliable, and a pleasure to work with. Not just scaling data, but scaling teams

Data Quality + Governance

Your data platform must be testable, alertable, and part of a software development lifecycle

Data Observability + Insights

Your Data Platform must allow you to observe the state of your pipelines across tools, while also introspecting the underlying data and metadata and enabling discoverability

Qualities of a Data Platform

Encompass the Software Development lifecycle

Testing, version control, branching, and local development

Support heterogeneous use cases

A data platform needs to accommodate a variety of storage and compute tools

Monolithic, unified data plane

Must not be siloed singular platform per team

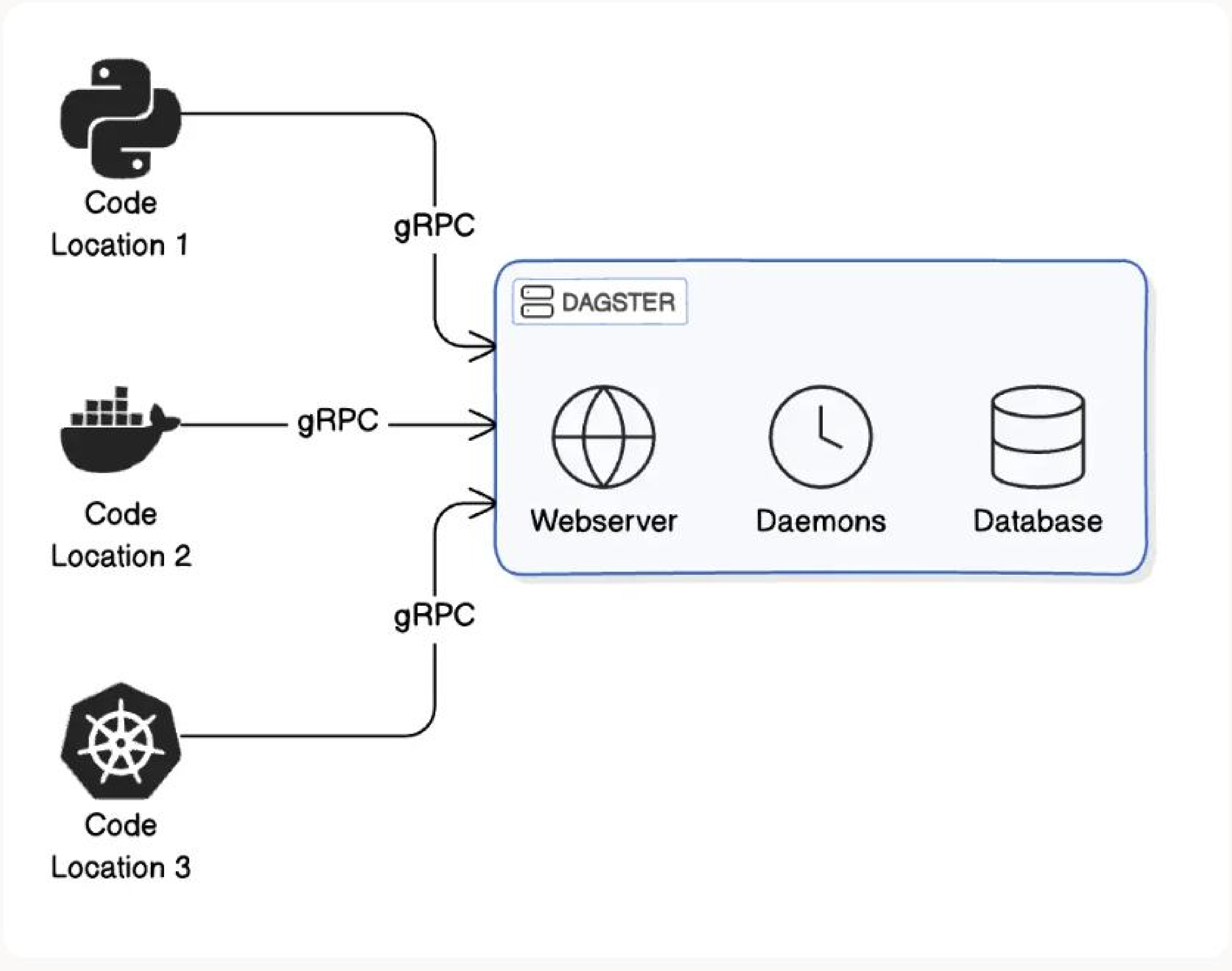
Declarative workflows

Must allow for both imperative and declarative workflows to simplify maintenance

Building a Data Platform with Dagster

Scalability + Maintainability

Code Locations



A code location helps in organizing and managing the code that defines your data assets and pipelines. It encapsulates your data assets, jobs, schedules, and sensors.

Scalability + Maintainability

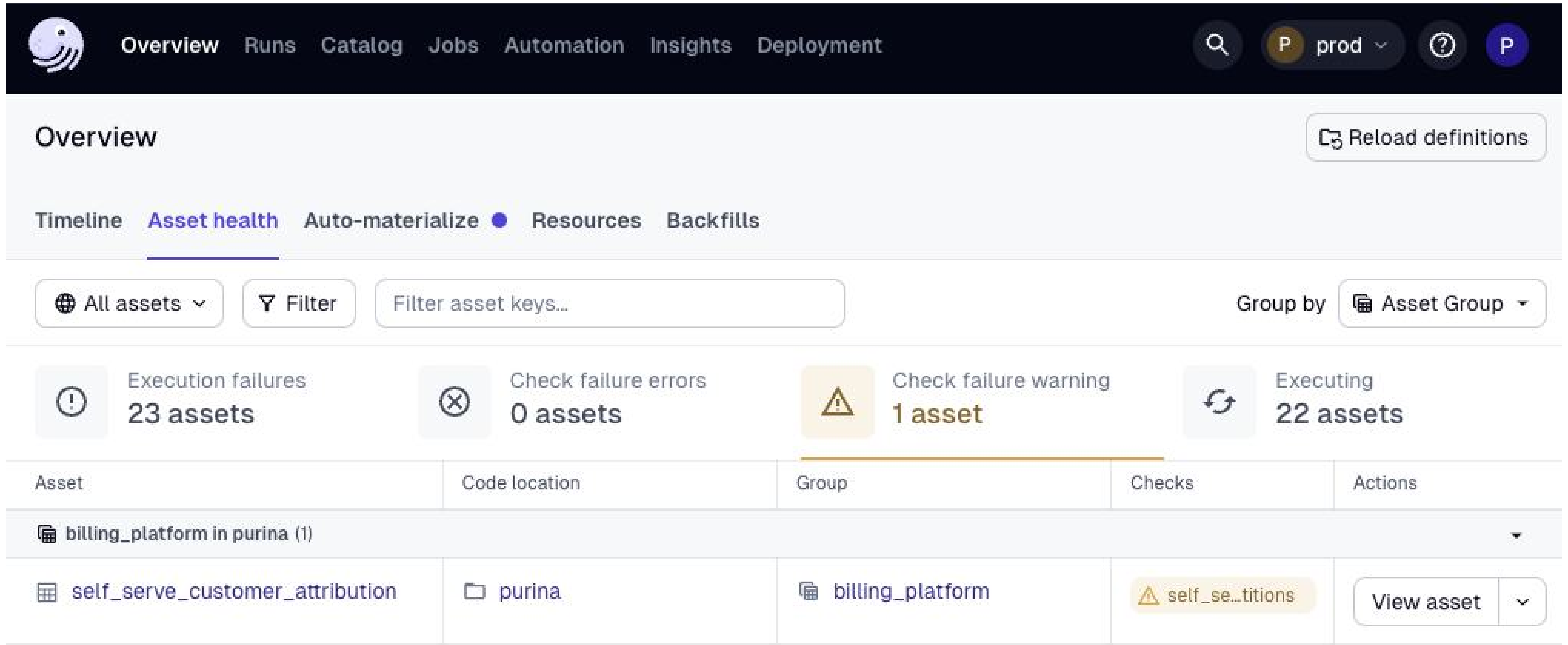
Pipelines as Code

Data Engineering is Software Engineering. Data pipelines are expressed as code, and checked into a shared version control system like git.

```python
@asset(
    compute_kind="github",
    group_name="support_bot",
    partitions_def=daily_partition,
    op_tags={"team": "devrel"},
    backfill_policy=BackfillPolicy.single_run(),
)
def github_issues(
    context: AssetExecutionContext, github: GithubResource, scoutos: ScoutosResource
) -> MaterializeResult:
    """Fetch Github Issues and Discussions and feed into Scout Support Bot.

    Since the Github API limits search results to 1000, we partition by updated at
    month, which should be enough to get all the issues and discussions. We use
    updated_at to ensure we don't miss any issues that are updated after the
    partition month. The underlying auto-materialize policy runs this asset every
    day to refresh all data for the current month.
    """
    start, end = context.partition_time_window
    context.log.info(f"Finding issues from {start} to {end}")

    issues = github.get_issues(
        start_date=start.strftime("%Y-%m-%d"), end_date=end.strftime("%Y-%m-%d")
    )
    context.log.info(f"Found {len(issues)} issues")

    parsed_issues = [parse_issue(i) for i in issues]
    context.log.info(f"Found {len(parsed_issues)} parsed issues")

    discussions = github.get_discussions(
        start_date=start.strftime("%Y-%m-%d"), end_date=end.strftime("%Y-%m-%d")
    )
    context.log.info(f"Found {len(discussions)} discussions")
    parsed_discussions = [parse_discussion(d) for d in discussions]
    context.log.info(f"Found {len(parsed_discussions)} parsed discussions")
    collection = os.getenv("SCOUTOS_COLLECTION_ID", "")
    context.log.info(f"Using Collection ID: {collection}")
    scoutos.write_documents(collection, parsed_issues)
    scoutos.write_documents(collection, parsed_discussions)
    return MaterializeResult()
```

Data Quality + Governance

Asset Checks + Health



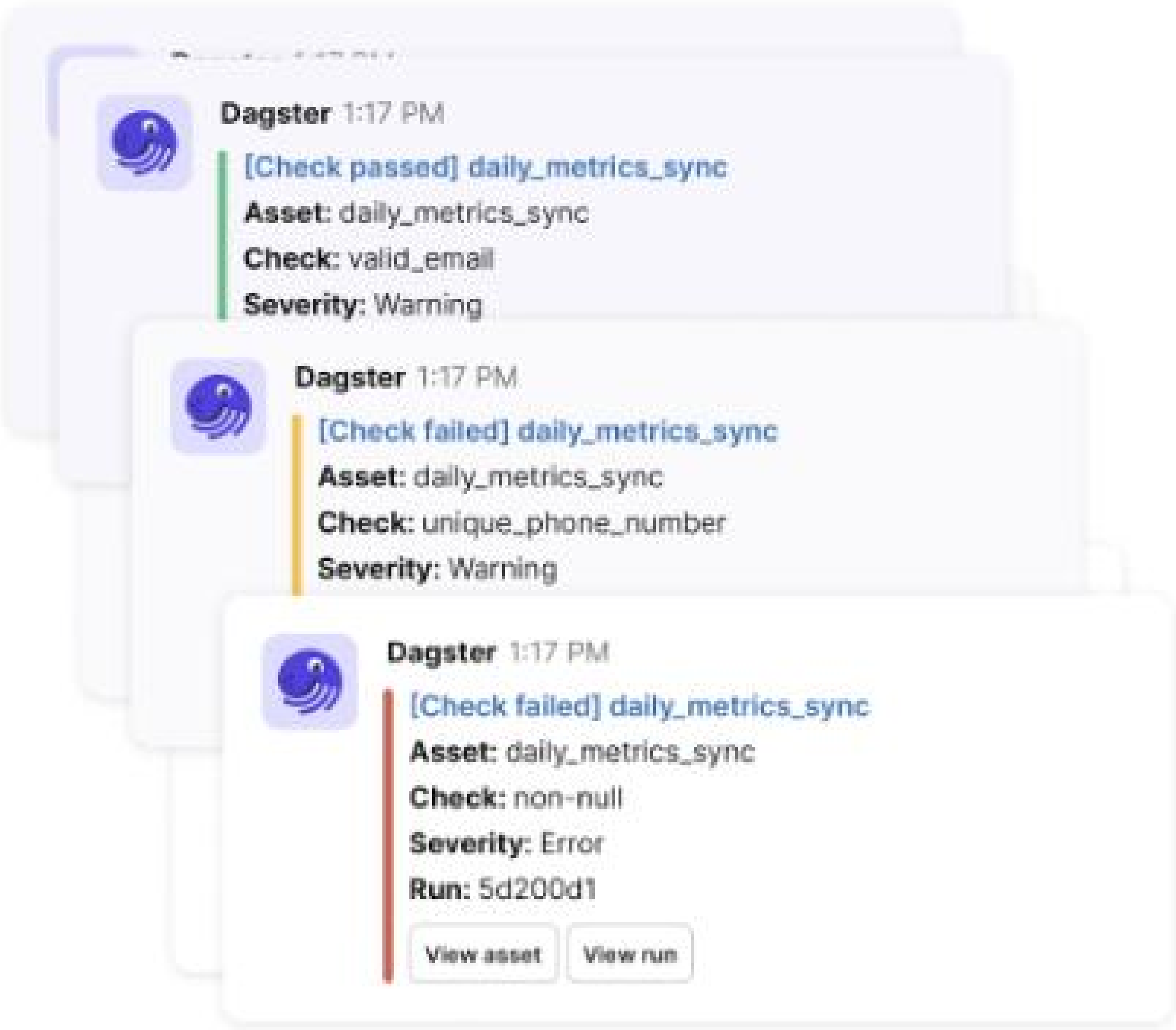
Catch data quality issues and schema changes early using Python-based logic.

dagster+

Data Quality + Governance

Data Freshness

```python
@dg.asset
def hourly_sales(context: dg.AssetExecutionContext):
    context.log.info("Fetching and emitting hourly sales data")
    ...


hourly_sales_freshness_check = dg.build_last_update_freshness_checks(
    assets=[hourly_sales], lower_bound_delta=timedelta(hours=1)
)
freshness_checks_sensor = dg.build_sensor_for_freshness_checks(
    freshness_checks=hourly_sales_freshness_check
)
```

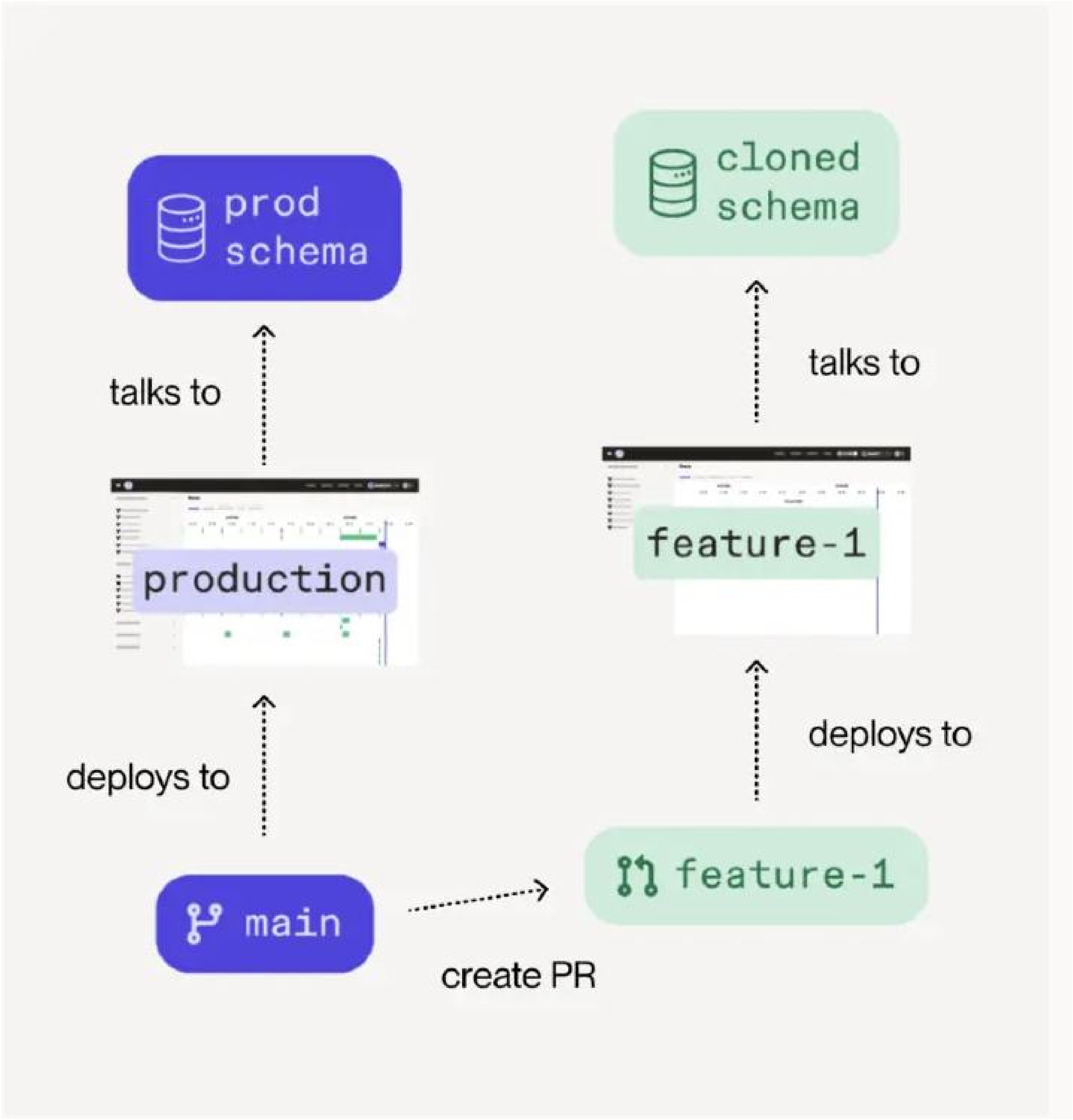


Freshness checks provide a way to identify data assets that are overdue for an update.

Data Quality + Governance

Change Management

Branch Deployments automatically create staging environments of your Dagster code, right in Dagster+. For every push to a branch in your git repository, Dagster+ will create a unique deployment, allowing you to preview the changes in the branch in real-time.



dagster+

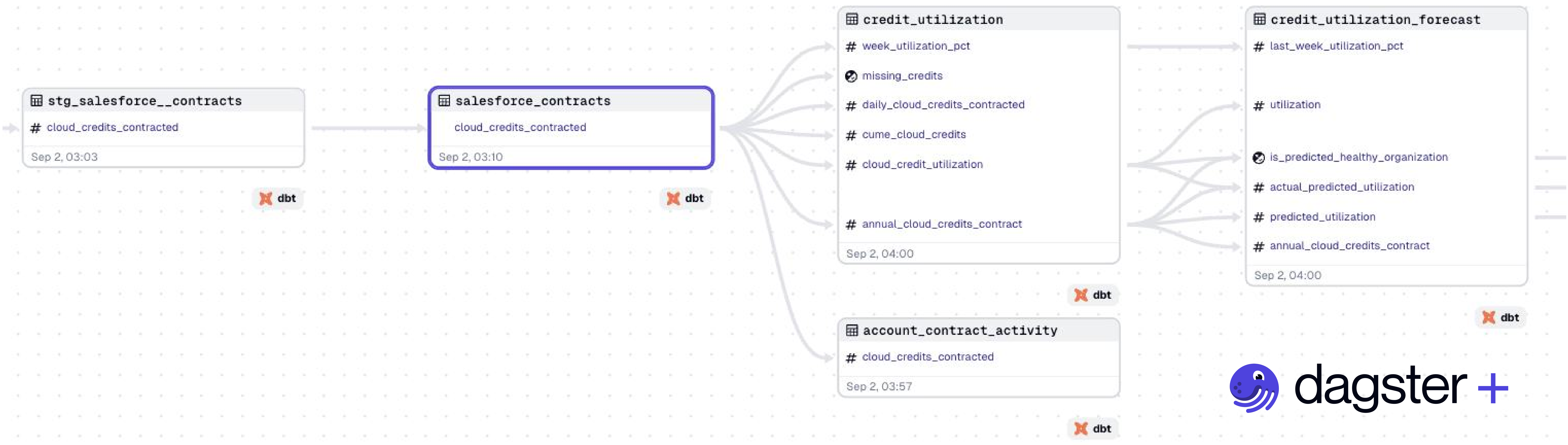
Data Observability + Insights

Asset-based pipelines

The Asset Graph is a **map of your data platform.** Dagster is the only data orchestrator that centers your pipelines on the assets you create rather than the tasks that create them

Data Observability + Insights

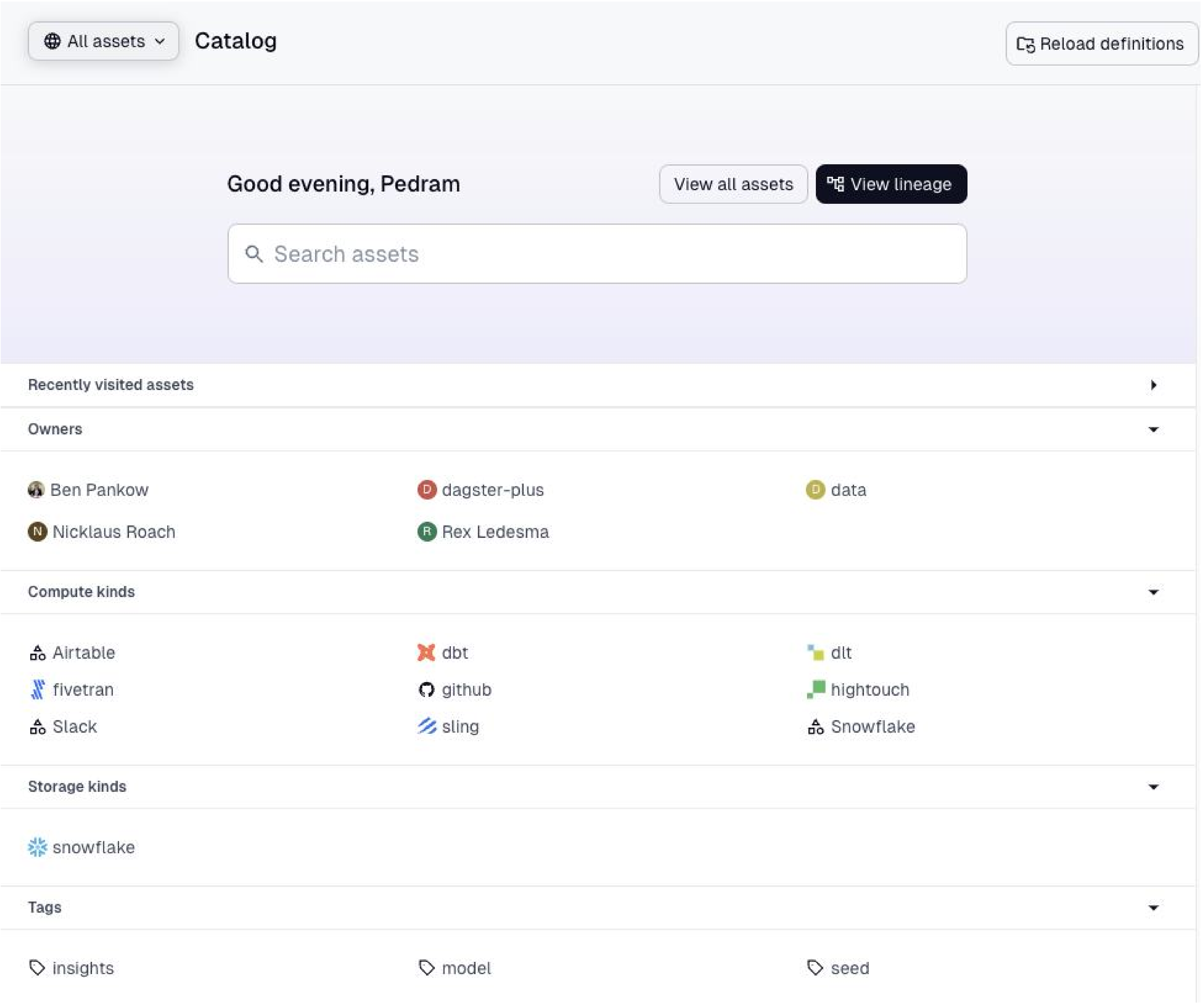
Column Level Lineage



Data Engineering is Software Engineering. Data pipelines are expressed as code, and checked into a shared version control system like git.

Data Observability + Insights

Data Catalog



Search by asset name, compute kind, group, owner, tag and more.

Wrapping Up

The Unkept Promise

The Modern Data Stack promised scalability, flexibility, and cost savings, but in reality, it often leaves us with a fragmented and complex ecosystem. This unkept promise has led to operational fragility, lack of observability, and expensive vendor lock-ins.

What We Believe

At Dagster, we believe in building a monolithic, unified data plane that supports heterogeneous use cases and declarative workflows. Our approach ensures that your data platform is not just a collection of disjointed tools but a cohesive, scalable, and maintainable system.

What You Can Accomplish

By leveraging Dagster, you can transform your modern data mess into a streamlined, efficient, and reliable data platform. Remember, the goal is not just to have a data platform but to have one that is truly effective and aligned with your organizational needs.

Q&A