



Dagster Deep Dives: Enable Data Mesh

Tim Castillo

Developer Advocate - Dagster Labs

The problem with centralizing and decentralizing

Fully Centralized Data Teams

- ✓ Centralized governance
Unified view across systems
Centralized performance management
Unified cost control
- ✗ Slow development cycle
Monolithic
Toolset compromises
Lacks specific business domain expertise

Fully Autonomous Data Teams

- ✓ Local autonomy and control
Dedicated toolsets
Faster innovation cycles
Local business domain expertise
- ✗ Duplication of efforts and tools
Hidden costs
Data silos
Poor cross-team collaboration
Data governance challenges
No centralized observability
Limited performance management



Data mesh is an organizational and technical paradigm for making high-quality data by domain experts

The Four Principles of Data Mesh are:

- Domain-driven data ownership
 - Data-as-a-product
- Federated computational governance
 - A self-serve data platform



Today, we'll talk about how to enable
a *data mesh* paradigm with Dagster.

We'll cover the **Four Principles** of data mesh and how
Dagster's building blocks fit well with them

Domain-Driven Data Ownership

- **Domain:** a group of people focused on the same goals
 - Ex. Sales, Marketing, HR
- Data for a specific purpose is made and managed by those who know it
 - ex. The marketing domain is responsible for the marketing data, and the Sales domain is responsible for the sales data

Data-as-a-Product

- Every domain builds, uses, and shares *data products*
- A data product is high-quality export of data with strong metadata
- Data products are written and accessed in persistent storage
 - Tables, files, machine learning models, streams, etc.
- Data products have metadata, such as:
 - SLAs for when the data is updated
 - Where the data is located/who is allowed to access it
 - Who owns and is responsible for the data
 - Observability and data quality
- Everything should be easily accessible for dependent teams

Federated computational governance

- Governance
 - Standardized ways for how people *build and share* data products
 - *Built*: Must be documented and easily accessible by every team
 - Update frequency SLAs, data quality, constraints
 - *Shared*: Often involves privacy, security, and compliance decisions
 - Access, maintenance, ownership, expectations and rollout of changes
- Computational
 - Enforcement should be automated by code, if possible
- Federated
 - Not wholly owned by a part of the organization
 - Everyone gets a seat at the table

A self-serve data platform

- A data platform is a foundation of shared resources, data products, and tools that empower domains to be part of the data mesh
- Using data products on a data platform
 - As a downstream team, you should be able to easily access those data products without additional work from the domains that produce them
 - Able to understand the upstream dependencies and implications
 - Self-discovery of source data upstream
- Producing data products in a data platform
 - A strong central data platform provides tools and processes to help domains produce their own data products
 - Federated computational governance should be easy for domains to implement
- Maintained and enabled by a data platform team
- A self-serve data platform empowers teams to access and produce data products independently, supported by a robust central infrastructure and the dedicated efforts of the data platform team

Governance Topologies : Different Approaches

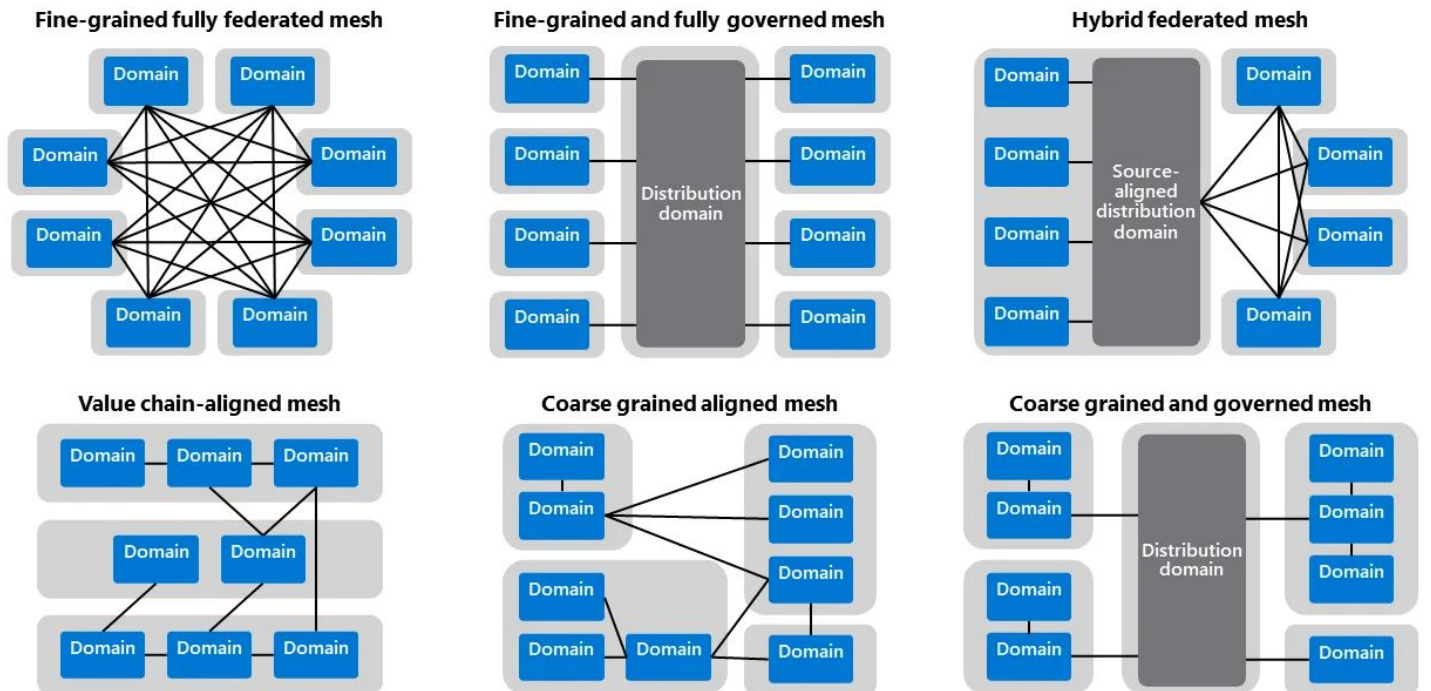


Image from [Data Mesh: Topologies and domain granularity](#)



The Four Principles of Data Mesh are:

- Domain-driven data ownership
 - Data-as-a-product
- Federated computational governance
 - A self-serve data platform

Everyone's data mesh looks different



About Dagster and Data Mesh

What's Dagster?

- Dagster is a framework for orchestrating data pipelines
- You have the flexibility to build pipelines outright or create a platform to enable others to build their own
- As a framework, Dagster is grounded in having strong core building blocks



Dagster's framework of building blocks can help enable the technical aspects of a data mesh

Notably, **Assets** and **Code Locations** are foundational in building a data platform and domains

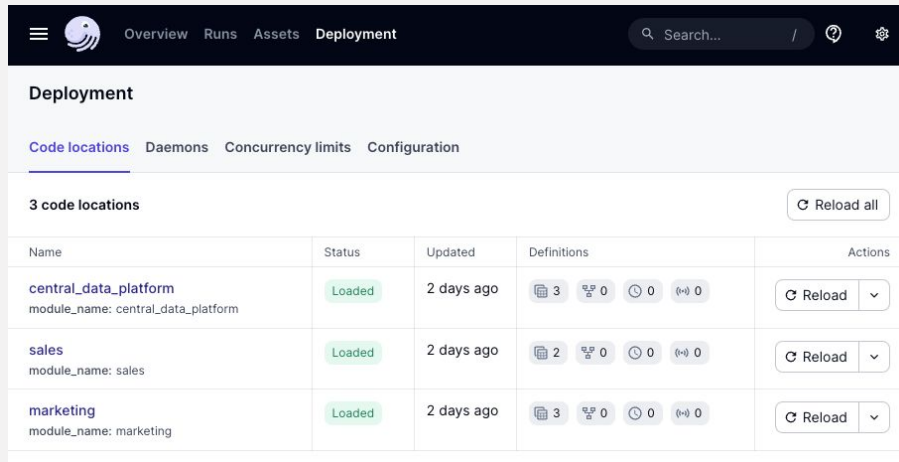
What's an Asset?

- **Assets are data products**
- Definition and data lineage
- Documentation
 - In the asset catalog
- Metadata about each run
- Observability
- Contracts
 - Data Quality with Asset Checks

The screenshot shows the Dagster web interface for an asset named 'dagster_pypi_downloads'. The top navigation bar includes links for Overview, Runs, Assets, Deployment, and Insights, along with a search bar and a 'prod' environment indicator. The main content area is titled 'Assets > dagster_pypi_downloads' and features a 'Materialize...' button. Below this, there are tabs for Overview, Partitions, Events, Checks, Plots, Lineage, Automation, and Insights. The 'Overview' tab is selected, displaying a 'Status' section with a 'Latest materialization' of 'Mar 30, 8:08 PM' and 'Check results' showing '2' checks passed. A 'Description' section provides context: 'A table containing the number of PyPi downloads for each package in the Dagster ecosystem, aggregated at the weekly grain. This data is fetched from the public BigQuery dataset bigquery-public-data.pypi.file_downloads.' The 'Metadata' section is partially visible. On the right, the 'Definition' section shows the 'Key' as 'dagster_pypi_downloads', the 'Group' as 'oss_analytics', the 'Code location' as 'dagster_open_platform' (loaded 2 days ago), and the 'Compute kind' as 'Snowflake'.

What's a Code Location?

- Part of the core architecture of Dagster
- **Code location:** a group of assets and other definitions
 - Independent from other code locations
 - Separated Python environment and dependencies
 - Can be deployed separately from each other
 - One deployment can have many code locations
- Assets can depend on assets in other code locations



The screenshot shows the Dagster web interface's 'Deployment' tab. It features a navigation bar with 'Overview', 'Runs', 'Assets', and 'Deployment'. Below the navigation bar, there's a 'Deployment' section with tabs for 'Code locations', 'Daemons', 'Concurrency limits', and 'Configuration'. The 'Code locations' tab is active, displaying a table with 3 code locations. Each row includes the name, status (all 'Loaded'), updated time (all '2 days ago'), definitions (code files, Python modules, and SQL queries), and actions (a 'Reload' button and a dropdown menu).

Name	Status	Updated	Definitions	Actions
central_data_platform module_name: central_data_platform	Loaded	2 days ago	3 code files, 0 Python modules, 0 SQL queries	Reload ⌵
sales module_name: sales	Loaded	2 days ago	2 code files, 0 Python modules, 0 SQL queries	Reload ⌵
marketing module_name: marketing	Loaded	2 days ago	3 code files, 0 Python modules, 0 SQL queries	Reload ⌵



Each domain can own a code location of their own respective assets, and domains can depend on the data products (assets) of other domains



Dagster has more features that can help make creating a data mesh even better

Domain-Driven Data Ownership

- Each domain gets their own code location
- Access controls can be applied (easily in Dagster Cloud)
- Domains can depend on assets from other domains



Data-as-a-product

- Assets can be treated as data products
- Metadata embedded into code when the asset is defined or when it is built
- Special metadata (ex. Owners) exist with enriched functionality
- Asset Checks communicate the assumptions that can be made with the data

Create alert policy

Policy details

Alert policy type

Asset alert

Alert policy name

data_platform_domain_alert

Description

Alerts the owners of the Core assets if they are out-of-date and miss their SLAs

Target

Group

Asset group

core in dagster_open_platform

Events

Materializations

☐ Success ☐ Failure

Asset checks

☐ Passed ☐ Failed (WARN) ☐ Failed (ERROR) ☐ Execution failed

Freshness

☒ Overdue

Notification service

☐ Slack

☐ Email

☐ Microsoft Teams

☒ Email asset owners (experimental)

Cancel

Send sample alert

Save policy

A self-serve data platform


- Code locations are Python modules
 - Use shared data platform code to promote quality
 - Python modules can be used to abstract and standardize governance
- Branch deployments can deploy every code location and test that everything works
 - Allow people to verify that access and usage of data products still works

Search filters...

f

☐ Select all

☐  Code Version

☐  Inputs

☐  New

☐  Partitions Definition

Federated computational governance

- SLAs - How frequently is it updated and when it is considered out-of-date?
 - Dagster can understand and report on “freshness” of your assets
 - Auto-Materialize Policies
- Data Quality - Users should be able to trust that the data is accurate and true
 - Asset checks
 - Alerting
- Owners and other asset metadata can be programmatically enforced

Create alert policy

Policy details

Alert policy type

Asset alert

Alert policy name

data_platform_domain_alert

Description

Alerts the owners of the Core assets if they are out-of-date and miss their SLAs

Target

Group

Asset group

core in dagster_open_platform

Events

Materializations

☐ Success ☐ Failure

Asset checks

☐ Passed ☐ Failed (WARN) ☐ Failed (ERROR) ☐ Execution failed

Freshness

☒ Overdue

Notification service

☐ Slack

☐ Email

☐ Microsoft Teams

☒ Email asset owners (experimental)

Cancel

Send sample alert

Save policy

Demo



Data mesh is an organizational and technical paradigm
for making high-quality data by domain experts

It focuses on the four principles we talked about today

Dagster's framework of building blocks can help enable
the technical aspects of a data mesh

Next steps & resources



Join us in Slack

Connect with other data practitioners. Share knowledge or find help

bit.ly/DagSlack



Sign up for Dagster Cloud

Sign up for Dagster Cloud and get started with a free 30 day trial

bit.ly/DagCloud



Get the Newsletter

Stay up-to-date on the latest events and news

bit.ly/DagNews



Star the GitHub Repo

Let us know you enjoyed this presentation!

bit.ly/DagDemo