



DAGSTER

Dagster

The Data Orchestrator

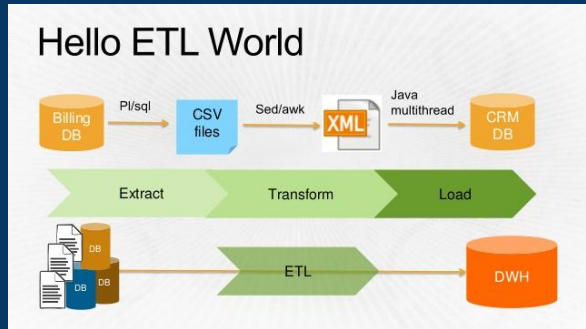


Budapest Data Forum 2020
Max Gasner
@gasnerpants

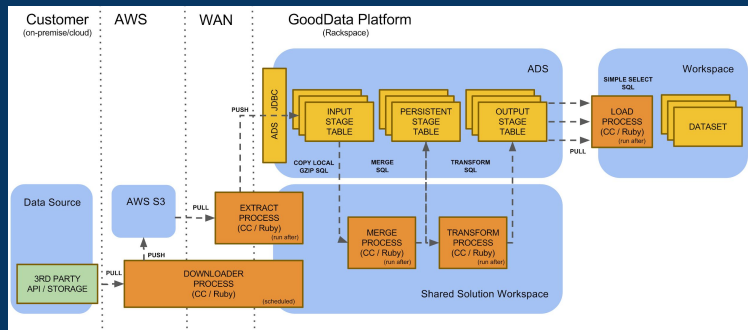
Data Applications

*graphs of computations
that consume and produce
data assets*

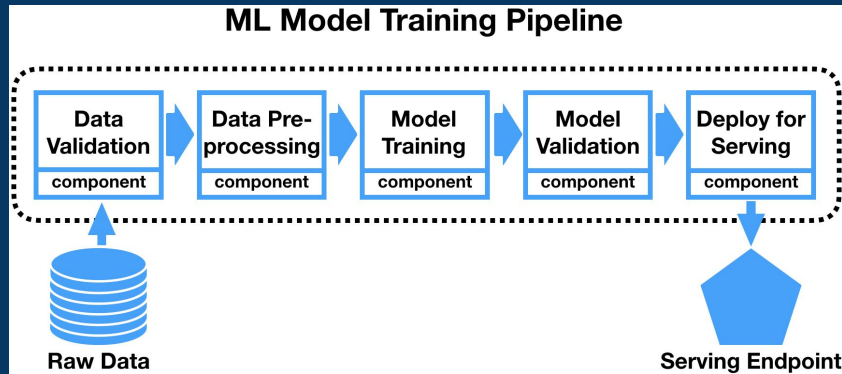
ETL



ELT



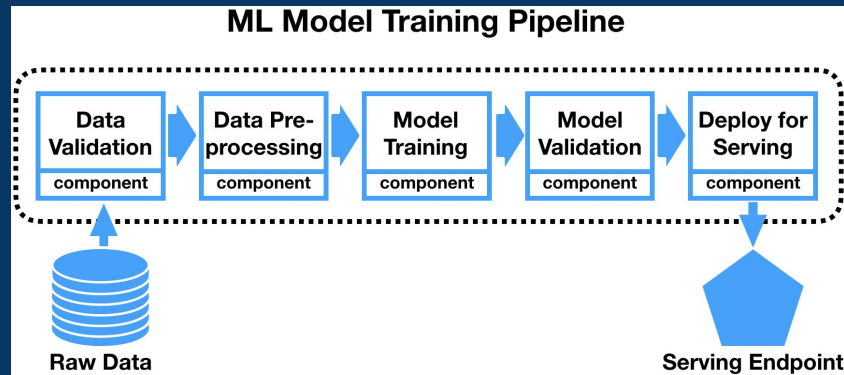
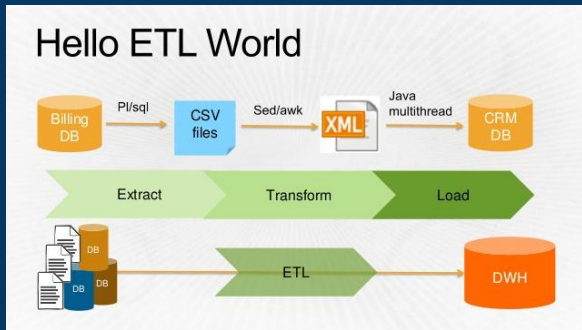
ML Pipeline



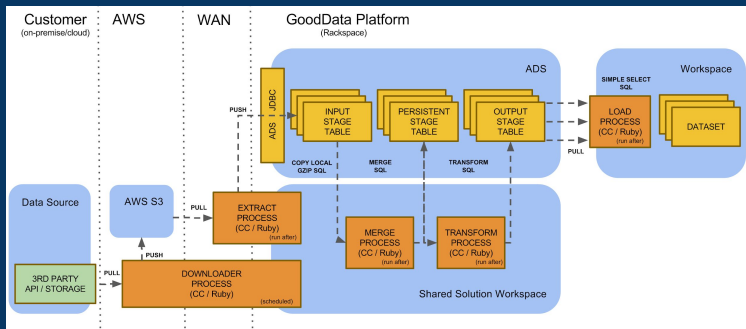
All are data applications

ETL

ML Pipeline



ELT

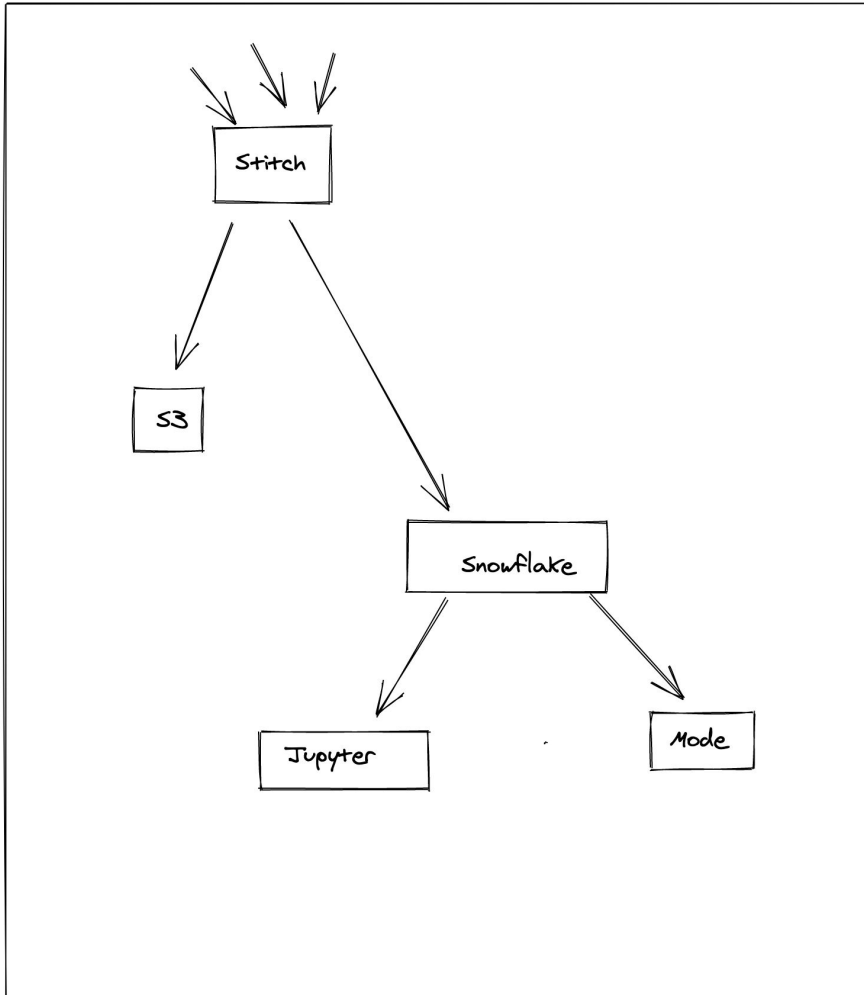


In fact, they could be three components of a broader, single data application

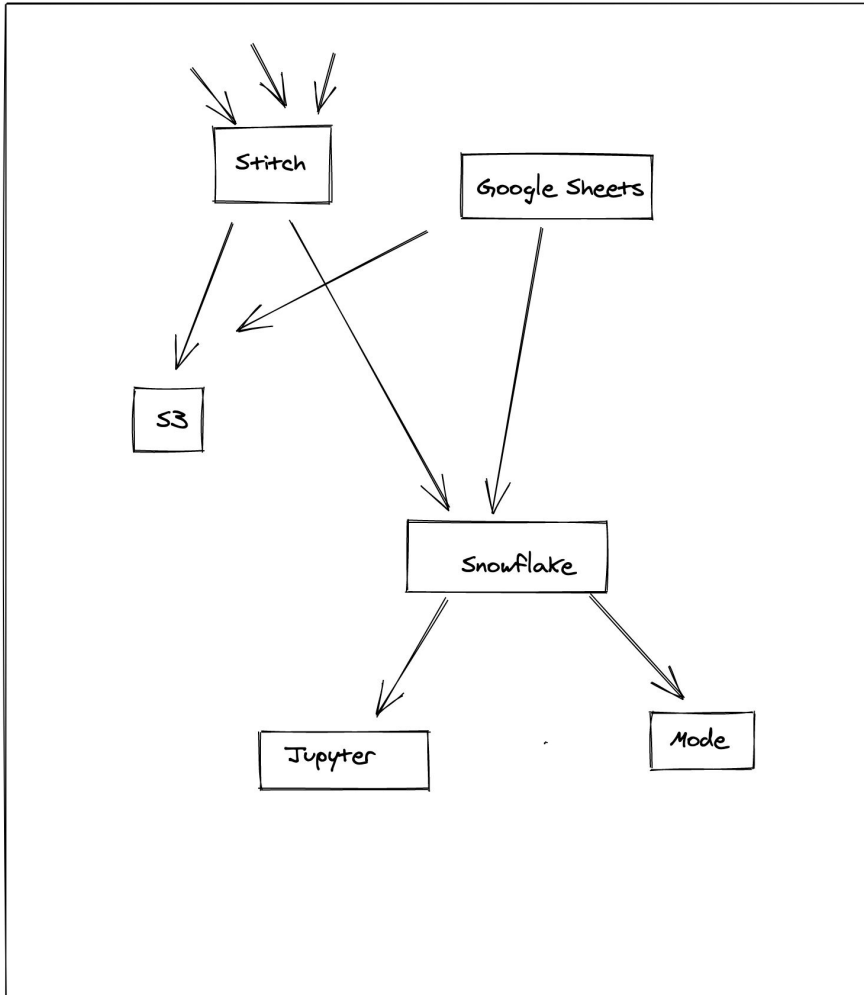
Data Applications

- Complex and heterogeneous
(personas, tools, teams, environments)

It starts simple...

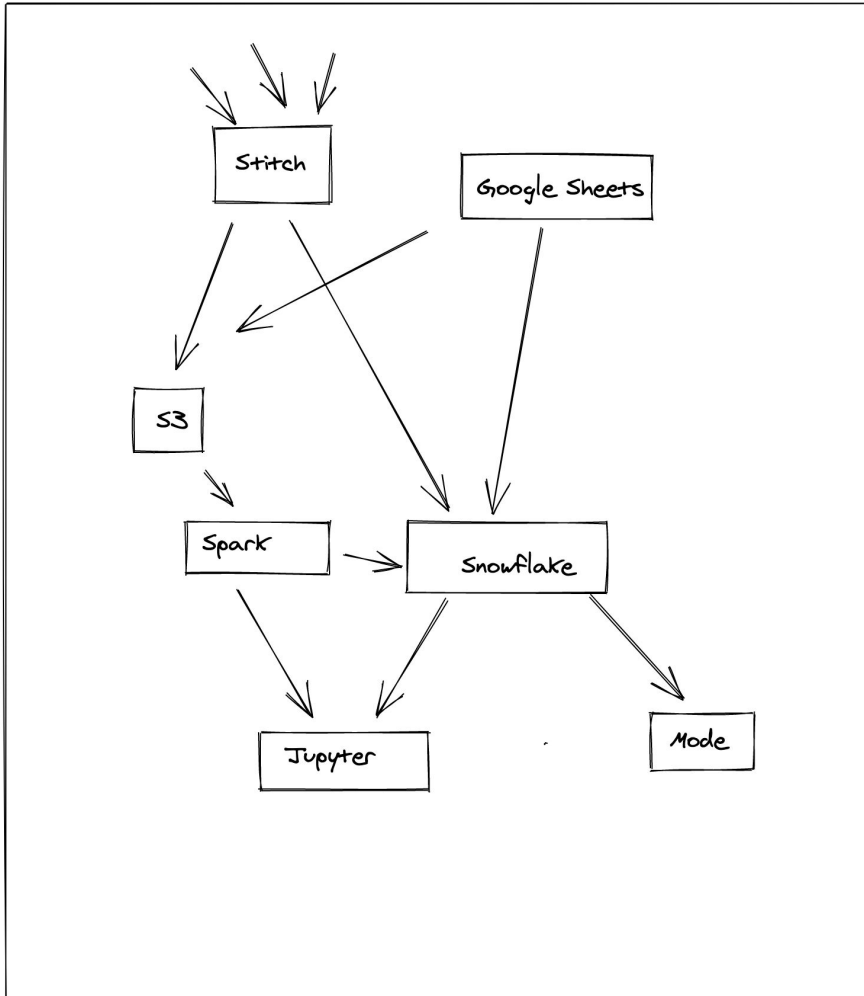


It starts simple...



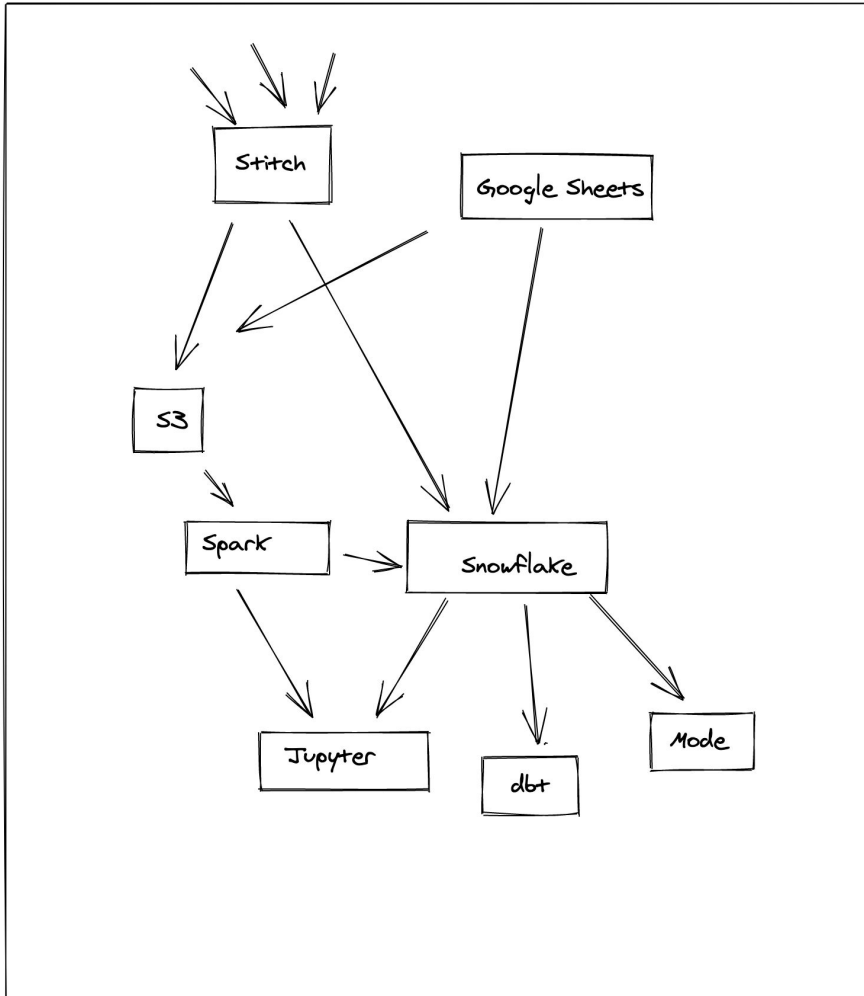
It starts simple...

Then the toolset grows...



It starts simple...

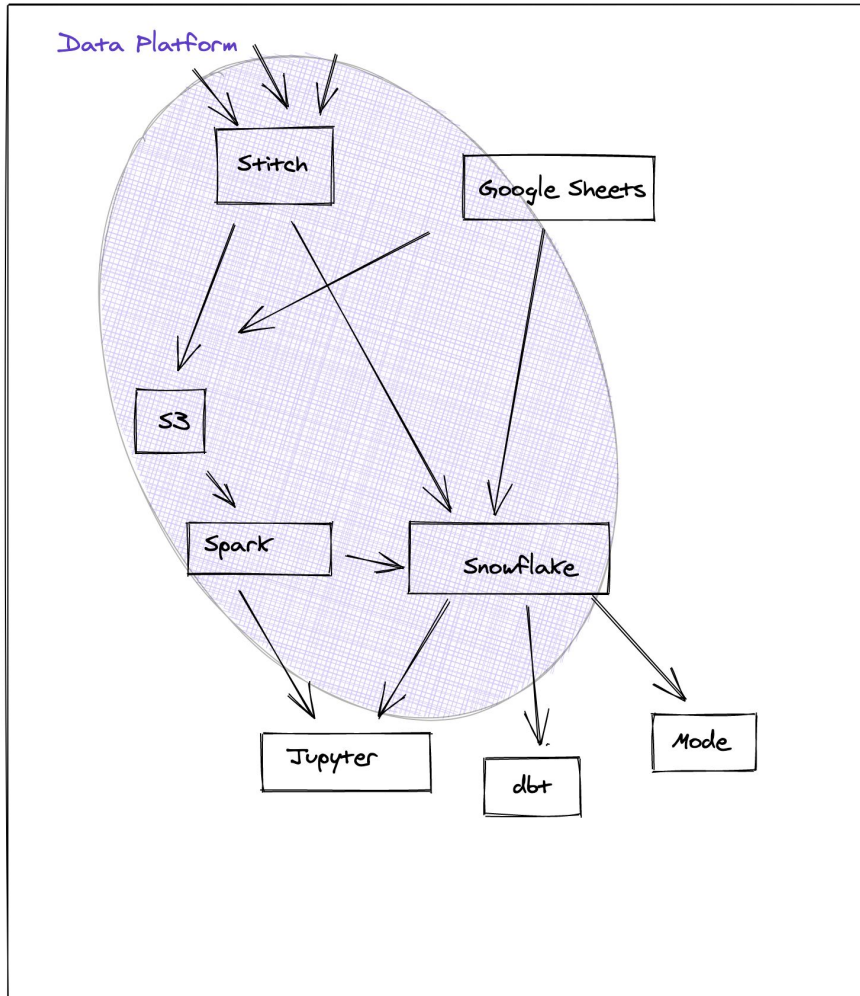
Then the toolset grows...



It starts simple...

Then the toolset grows...

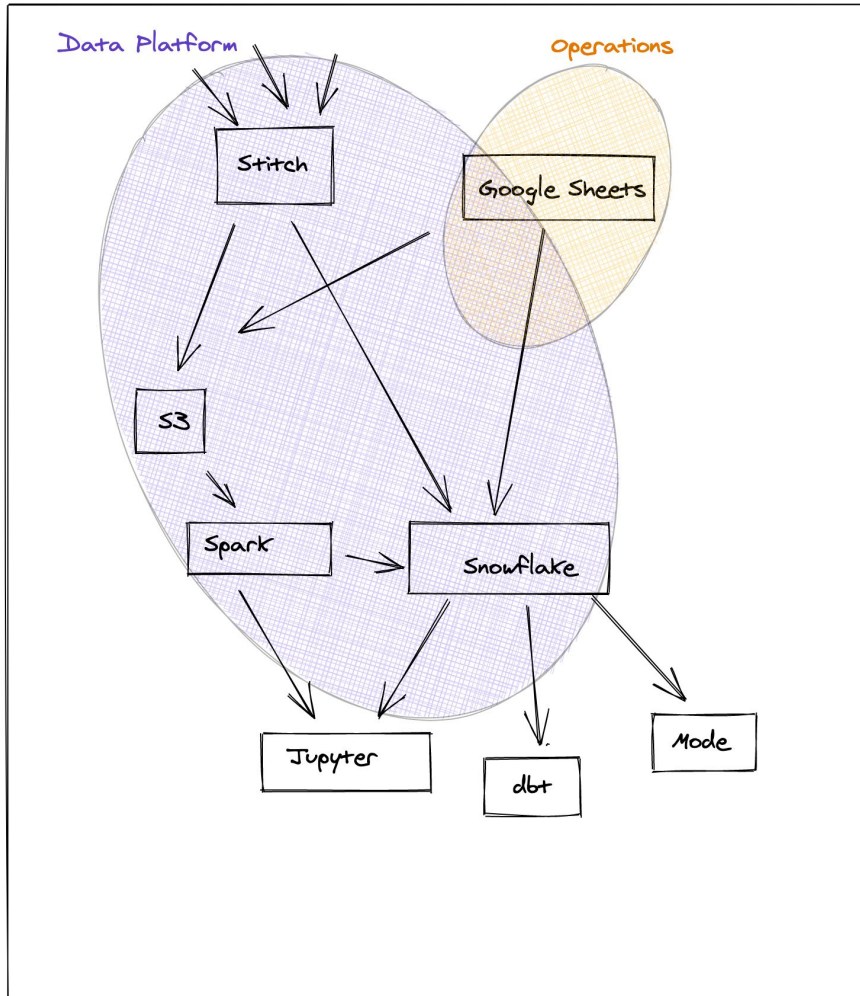
Along with teams and personas



It starts simple...

Then the toolset grows...

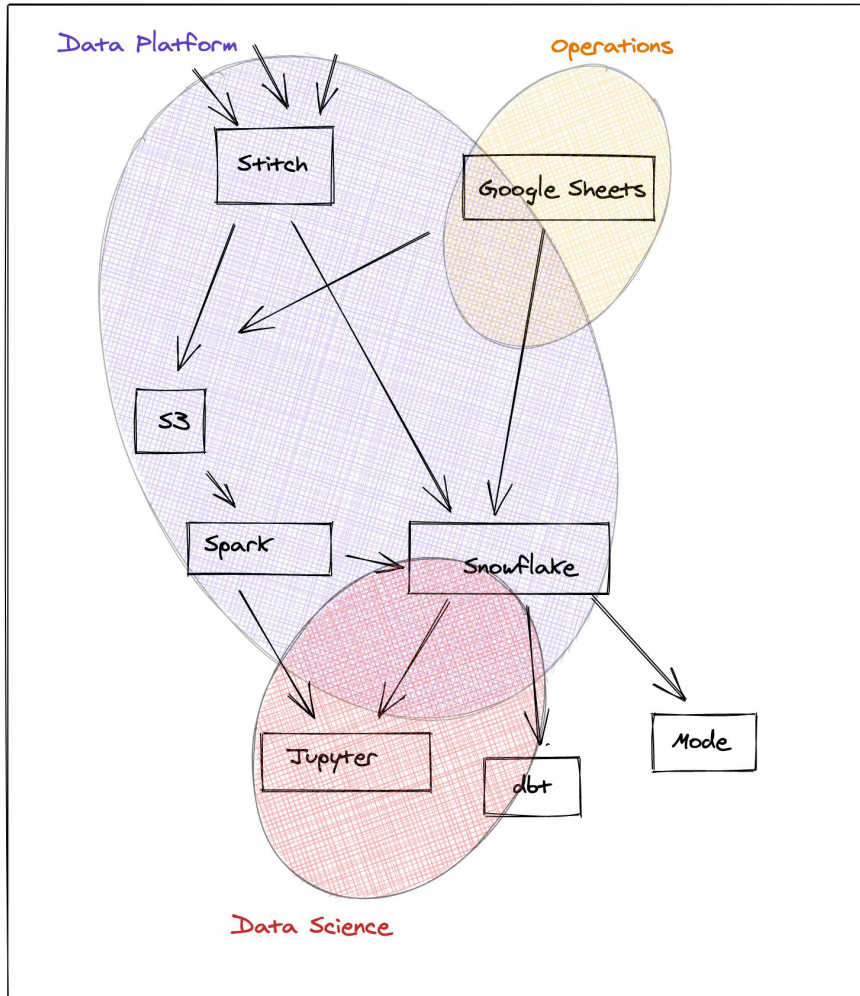
Along with teams and personas



It starts simple...

Then the toolset grows...

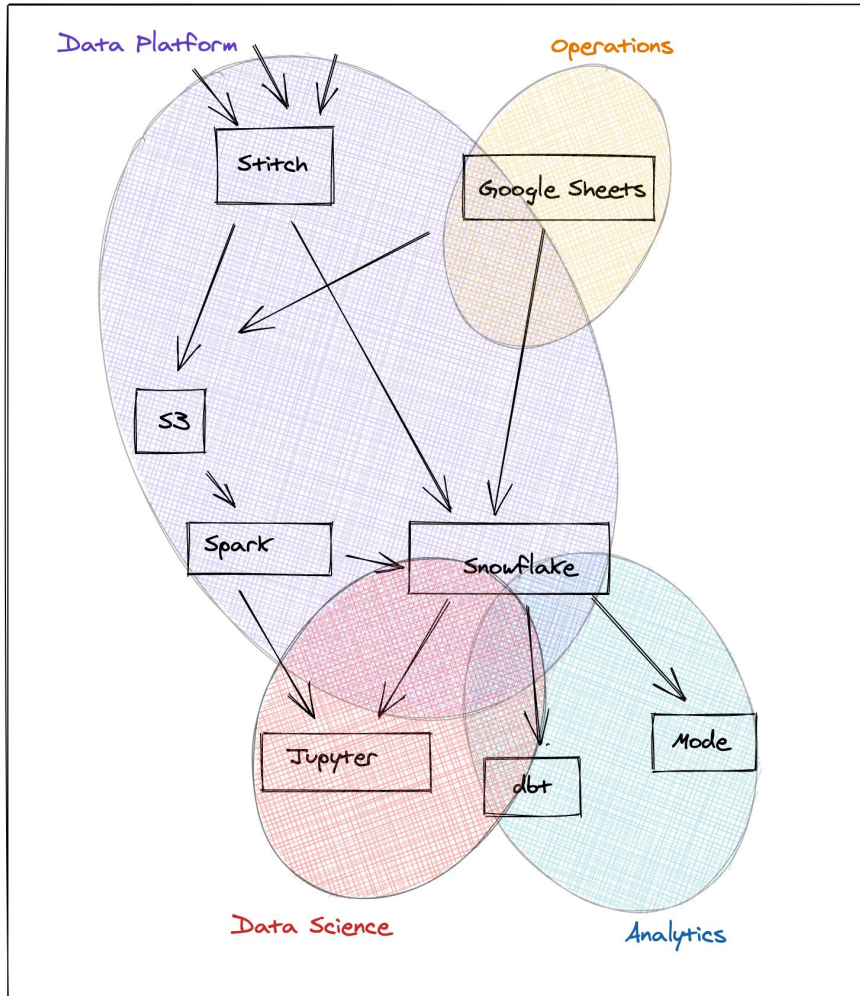
Along with teams and personas



It starts simple...

Then the toolset grows...

Along with teams and personas



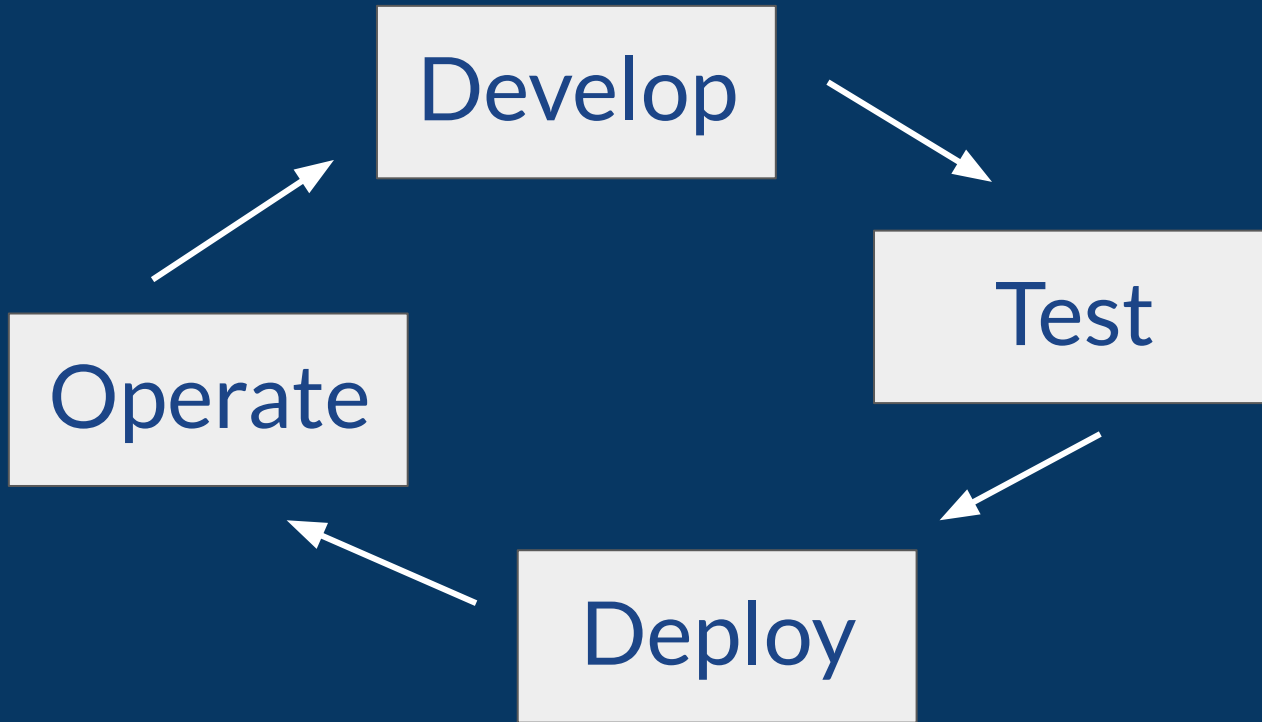
Data Applications

- Complex and heterogeneous
(personas, tools, teams, environments)

Data Applications

- Complex and heterogeneous
(personas, tools, teams, environments)

Everything is hard



Data Applications

- Complex and heterogeneous
(personas, tools, teams, environments)
- Hard to develop
- Hard to test
- Hard to deploy
- Hard to operate

Software engineering

Data Orchestrator

Combines ideas from several software lineages:

- Workflow engines (Luigi, Airflow): focus on ops
- Dataflow programming: design principles
- ETL environments (Informatica): user focus
- DevOps: full dev cycle

Data Orchestrator

- View data apps as a graph of functional computations
(isolate external state)
- Nodes are computations, edges are data-aware
(connect data to computations)
- Computations produce stream of structured metadata
(platform for tooling)

Programming model



DAGSTER

Tooling

Platform



Dagster

- Programming model
 - > `pip install dagster`

```
@solid(
```

```
    description="A solid that posts a message to Slack.",  
    config_schema={"channel": Field(str)},  
    required_resource_keys={"slack"},  
)
```

```
def post_slack_message(context, text: str) → List[str]:  
    channel = context.solid_config["channel"]  
    slack = context.resources.slack  
    resp = slack.chat_postMessage(channel=channel, text=text)  
    yield AssetMaterialization(  
        description="A message posted to Slack.",  
        metadata_entries=[  
            EventMetadataEntry.json(  
                data=resp.data, label="slack_response"  
            )  
        ],  
        asset_key="slack.message",  
    )  
    yield Output(resp["ts"])
```

@solid: a functional unit of computation in the orchestration graph

```
@solid(
```

```
    description="A solid that posts a message to Slack.",  
    config_schema={"channel": Field(str)},  
    required_resource_keys={"slack"},  
)
```

```
def post_slack_message(context, text: str) → List[str]:  
    channel = context.solid_config["channel"]  
    slack = context.resources.slack  
    resp = slack.chat_postMessage(channel=channel, text=text)  
    yield AssetMaterialization(  
        description="A message posted to Slack.",  
        metadata_entries=[  
            EventMetadataEntry.json(  
                data=resp.data, label="slack_response"  
            )  
        ],  
        asset_key="slack.message",  
    )  
    yield Output(resp["ts"])
```

@solid: a functional unit of computation in the orchestration graph

(Annotated Python function)

```
@solid(
    description="A solid that posts a message to Slack.",
    config_schema={"channel": Field(str)},
    required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
    channel = context.solid_config["channel"]
    slack = context.resources.slack
    resp = slack.chat_postMessage(channel=channel, text=text)
    yield AssetMaterialization(
        description="A message posted to Slack.",
        metadata_entries=[
            EventMetadataEntry.json(
                data=resp.data, label="slack_response"
            )
        ],
        asset_key="slack.message",
    )
    yield Output(resp["ts"])
```

@solid: a functional unit of computation in the orchestration graph

(Annotated Python function)

Self-documenting

Injected context
parameter isolates
access to external
state

```
@solid(
    description="A solid that posts a message to Slack.",
    config_schema={"channel": Field(str)},
    required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
    channel = context.solid_config["channel"]
    slack = context.resources.slack
    resp = slack.chat_postMessage(channel=channel, text=text)
    yield AssetMaterialization(
        description="A message posted to Slack.",
        metadata_entries=[
            EventMetadataEntry.json(
                data=resp.data, label="slack_response"
            )
        ],
        asset_key="slack.message",
    )
    yield Output(resp["ts"])
```

Injected context
parameter isolates
access to external
state

Solids declare their
resource requirements

```
@solid(
    description="A solid that posts a message to Slack.",
    config_schema={"channel": Field(str)},
    required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
    channel = context.solid_config["channel"]
    slack = context.resources.slack
    resp = slack.chat_postMessage(channel=channel, text=text)
    yield AssetMaterialization(
        description="A message posted to Slack.",
        metadata_entries=[
            EventMetadataEntry.json(
                data=resp.data, label="slack_response"
            )
        ],
        asset_key="slack.message",
    )
    yield Output(resp["ts"])
```

```
@solid(
  description="A solid that posts a message to Slack.",
  config_schema={"channel": Field(str)},
  required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
  channel = context.solid_config["channel"]
  slack = context.resources.slack
  resp = slack.chat_postMessage(channel=channel, text=text)
  yield AssetMaterialization(
    description="A message posted to Slack.",
    metadata_entries=[
      EventMetadataEntry.json(
        data=resp.data, label="slack_response"
      )
    ],
    asset_key="slack.message",
  )
  yield Output(resp["ts"])
```

Injected context
parameter isolates
access to external
state

Solids declare their
resource requirements

External resources are
injected by framework
(so can be mocked)

Solids yield a stream of structured events

```
@solid(
  description="A solid that posts a message to Slack.",
  config_schema={"channel": Field(str)},
  required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
  channel = context.solid_config["channel"]
  slack = context.resources.slack
  resp = slack.chat_postMessage(channel=channel, text=text)
  yield AssetMaterialization(
    description="A message posted to Slack.",
    metadata_entries=[
      EventMetadataEntry.json(
        data=resp.data, label="slack_response"
      )
    ],
    asset_key="slack.message",
  )
  yield Output(resp["ts"])
```

Solids yield a stream of structured events

Metadata is stored and available to tooling

```
@solid(
    description="A solid that posts a message to Slack.",
    config_schema={"channel": Field(str)},
    required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
    channel = context.solid_config["channel"]
    slack = context.resources.slack
    resp = slack.chat_postMessage(channel=channel, text=text)
    yield AssetMaterialization(
        description="A message posted to Slack.",
        metadata_entries=[
            EventMetadataEntry.json(
                data=resp.data, label="slack_response"
            )
        ],
        asset_key="slack.message",
    )
    yield Output(resp["ts"])
```


Solids yield a stream of structured events

Metadata is stored and available to tooling

Meaningful side effects are represented as assets

```
@solid(
    description="A solid that posts a message to Slack.",
    config_schema={"channel": Field(str)},
    required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
    channel = context.solid_config["channel"]
    slack = context.resources.slack
    resp = slack.chat_postMessage(channel=channel, text=text)
    yield AssetMaterialization(
        description="A message posted to Slack.",
        metadata_entries=[
            EventMetadataEntry.json(
                data=resp.data, label="slack_response"
            )
        ],
        asset_key="slack.message",
    )
    yield Output(resp["ts"])
```

Inputs and outputs are typed

```
@solid(
    description="A solid that posts a message to Slack.",
    config_schema={"channel": Field(str)},
    required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
    channel = context.solid_config["channel"]
    slack = context.resources.slack
    resp = slack.chat_postMessage(channel=channel, text=text)
    yield AssetMaterialization(
        description="A message posted to Slack.",
        metadata_entries=[
            EventMetadataEntry.json(
                data=resp.data, label="slack_response"
            )
        ],
        asset_key="slack.message",
    )
    yield Output(resp["ts"])
```

Inputs and outputs are typed

Config schema is typed

```
@solid(
  description="A solid that posts a message to Slack.",
  config_schema={"channel": Field(str)},
  required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
    channel = context.solid_config["channel"]
    slack = context.resources.slack
    resp = slack.chat_postMessage(channel=channel, text=text)
    yield AssetMaterialization(
        description="A message posted to Slack.",
        metadata_entries=[
            EventMetadataEntry.json(
                data=resp.data, label="slack_response"
            )
        ],
        asset_key="slack.message",
    )
    yield Output(resp["ts"])
```

Comment on output type

Inputs and outputs are typed

Config schema is typed

Solids yield meaningful outputs to downstream computations

```
@solid(
    description="A solid that posts a message to Slack.",
    config_schema={"channel": Field(str)},
    required_resource_keys={"slack"},
)
def post_slack_message(context, text: str) → List[str]:
    channel = context.solid_config["channel"]
    slack = context.resources.slack
    resp = slack.chat_postMessage(channel=channel, text=text)
    yield AssetMaterialization(
        description="A message posted to Slack.",
        metadata_entries=[
            EventMetadataEntry.json(
                data=resp.data, label="slack_response"
            )
        ],
        asset_key="slack.message",
    )
    yield Output(resp["ts"])
```

Pipeline DSL mimics
function invocation

```
@pipeline(  
  ... ,  
  mode_defs=[  
    ModeDefinition(  
      ... ,  
      name="prod",  
      resource_defs={"slack": slack_resource, ... : ... },  
    ),  
    ... ,  
  ],  
)  
  
def ingest_transportation_temp_labor():  
  ...  
  ingest_result_message = build_ingest_slack_message(  
    load_s3_to_snowflake( ... )  
  )  
  post_slack_message(ingest_result_message)
```

Pipeline DSL mimics
function invocation

Solid inputs and
outputs are connected

```
@pipeline(  
    ... ,  
    mode_defs=[  
        ModeDefinition(  
            ... ,  
            name="prod",  
            resource_defs={"slack": slack_resource, ... : ... },  
        ),  
        ... ,  
    ],  
)  
  
def ingest_transportation_temp_labor():  
    ...  
    ingest_result_message = build_ingest_slack_message(  
        load_s3_to_snowflake( ... )  
    )  
    post_slack_message(ingest_result_message)
```

```
@pipeline(  
    ... ,  
    mode_defs=[  
        ModeDefinition(  
            ... ,  
            name="prod",  
            resource_defs={"slack": slack_resource, ... : ... },  
        ),  
        ... ,  
    ],  
)  
  
def ingest_transportation_temp_labor():  
    ...  
    ingest_result_message = build_ingest_slack_message(  
        load_s3_to_snowflake( ... )  
    )  
    post_slack_message(ingest_result_message)
```

Pipeline DSL mimics
function invocation

Solid inputs and
outputs are connected

Resource
implementations can
be swapped out

Custom types help to ensure runtime correctness of inputs and outputs

```
TransportationTempLaborDataFrame = create_dagster_pandas_dataframe_type(  
    name="TransportationTempLaborDataFrame",  
    columns=[  
        PandasColumn.datetime_column(  
            name="date",  
            min_datetime=datetime.datetime(2017, 10, 1),  
            max_datetime=datetime.datetime.utcnow(),  
            non_nullable=True,  
        ),  
        PandasColumn.string_column(name="team", non_nullable=True),  
        ... ,  
    ],  
    dataframe_constraints=[  
        UniqueColumnsConstraint(columns=["date", "location"]),  
        ... ,  
    ],  
)
```

@solid

Functional unit of computation

Stream of metadata

Basis for tooling

context

Access to external state

Custom Types

Correctness guarantees

@pipeline

Defines data dependencies

(and much more...)

@resource

Injected external state

Dagster

- Programming model
- Tooling
 - > `pip install dagit && dagit`

Tooling

- View of orchestration graph
- Playground with typed config editor
- Operational views: runs, schedules, assets,
longitudinal graphs

Dagit

View of orchestration graph

The screenshot displays the Dagit web interface for a pipeline named 'longitudinal_pipeline'. The interface is divided into a left sidebar and a main content area.

Left Sidebar:

- Header:** DAGSTER 0.9.1 Instance Details
- Navigation:** Runs, Assets, Scheduler (with expand/collapse icons).
- Repository:** toys_repository
- Pipelines & Solids:** Search pipelines, P then Up / Down, hammer_pipeline, log_spew, longitudinal_pipeline (selected), many_events, retry_pipeline, sleepy_pipeline.
- Schedules:** Search schedules, P then Up / Down, View All.
- Footer:** Automatic

Main Content Area:

- Navigation:** longitudinal_pipeline, Overview, Definition (selected), Playground, Runs, Partitions.
- Search:** Select a Solid..., Highlight...
- Orchestration Graph:** A directed graph showing the flow of data and tasks. The graph starts with two parallel ingestion tasks: 'ingest_costs' and 'ingest_traffic'. 'ingest_costs' leads to 'persist_costs', which then leads to 'build_cost_dashboa...'. 'ingest_traffic' leads to 'persist_traffic', which then leads to 'build_traffic_dash...'. Both 'build_cost_dashboa...' and 'build_traffic_dash...' lead to a central 'build_model' task. 'build_model' leads to 'train_model', which finally leads to 'persist_model'.
- Footer:** ✨ Type a Solid Subset (ex: persist_costs+)

Dagit

View of orchestration graph

Execution playground with
typed config editor

The screenshot displays the Dagster web interface for a pipeline named 'longitudinal_pipeline'. The interface is divided into several sections:

- Header:** Shows the pipeline name 'longitudinal_pipeline' and navigation tabs for Overview, Definition, Playground (active), Runs, and Partitions.
- Left Sidebar:** Contains navigation options for Runs, Assets, Scheduler, Repository (toys_repository), Pipelines & Solids (with a search bar and a list of pipeline names like branch_pipeline, composition, error_monster, etc.), and Schedules.
- Main Editor:** Features a 'New Run' button and a 'Partition Set' dropdown. The configuration editor shows a typed YAML configuration with auto-completion suggestions for 'config' (max_concurrent, retries). A JSON schema for 'retries' is visible on the right.
- Errors Section:** Displays an error message: 'Value at path root:execution:multiprocess:config must not be None.'.
- Runtime Section:** Includes a 'Launch Execution' button and a list of available solids such as 'execution', 'intermediate_storage', 'loggers', 'storage', 'build_cost_dashboard', 'build_model', etc.

Dagit

View of orchestration graph

Execution playground with
typed config editor

Operational views of
pipeline runs, schedules,
longitudinal views










The screenshot displays the Dagit web interface for a pipeline named 'longitudinal_pipeline'. The interface is divided into several sections:

- Left Sidebar:** Contains navigation options for 'Runs', 'Assets', and 'Scheduler'. Below this is the 'REPOSITORY' section, currently set to 'toys_repository'. The 'Pipelines & Solids' section lists various pipeline components like 'branch_pipeline', 'composition', 'error_monster', 'fan_in_fan_out_pipeline', 'hammer_pipeline', and 'log_spew'. At the bottom, there is a 'Schedules' section.
- Top Navigation:** Includes tabs for 'Overview', 'Definition', 'Playground', 'Runs', and 'Partitions'. The 'Runs' tab is currently active.
- Main View:** Shows an orchestration graph with green rectangular nodes representing pipeline steps. A 'Launch Re-execution' button is visible at the top left of the graph area. The graph shows a sequence of steps, with some steps branching out. A timeline at the top indicates durations of 5s and 10s.
- Right Panel:** Displays the execution status of the pipeline. It is currently in the 'PREPARING' state, with the message 'No steps are preparing to execute'. Below this, the 'EXECUTING' state is shown with 'No steps are executing'. The 'ERRORED' state is currently empty.
- Bottom Panel:** A log viewer showing a table of events. The table has columns for 'SOLID', 'EVENT TYPE', 'INFO', and 'TIMESTAMP'. The events include input/output for 'persist_model', step completion for 'persist_model.compute', pipeline completion, and engine events for process completion and exit.


SOLID	EVENT TYPE	INFO	TIMESTAMP
persist_model	Input	Got input "_" of type "Any". (Type check passed).	14:08:19.940
persist_model	Output	Yielded output "result" of type "Any". (Type check passed).	14:08:20.479
persist_model	Step Finished	Finished execution of step "persist_model.compute" in 522ms.	14:08:20.497
-	Pipeline Finis...	Finished execution of pipeline "longitudinal_pipeline".	14:08:20.545
-	Engine Event	Finished steps in process (pid: 32531) in 5.67s <pre>pid 32531 step_keys ['ingest_costs.compute', 'ingest_traffic.compute', 'persist_costs.compute', 'persist_traffic.compute', 'build_cost_dashboard.compute', 'build_model.compute', 'build_traffic_dashboard.compute', 'train_model.compute', 'persist_model.compute']</pre>	14:08:20.518
-	Engine Event	Process for pipeline exited (pid: 32531).	14:08:20.613

Schedules

8 loaded from `internal_dagit_repository`

	SCHEDULE NAME	PIPELINE	SCHEDULE	LAST TICK	LATEST RUNS	EXECUTION PARAMS
<input type="radio"/> off	backfill_unreliable_weekly Schedule ID: 9d98411bba30438ac2318449fd098f4...	 unreliable_pipeline	Every minute			Mode: default 
<input type="radio"/> off	daily_weather_ingest_schedule Schedule ID: 6ed57aa3f60a60c19808edf6f486088...	 generate_training_set...	At 02:41 PM			Mode: production 
<input type="radio"/> off	daily_weather_schedule Schedule ID: ede7c8e0f86675c5e6883a6bf3bbf36...	 daily_weather_pipeline	At 02:41 PM			Mode: production 
<input type="radio"/> off	longitudinal_demo Schedule ID: bc80de353bf1cc249da3bbcc409bf724...	 longitudinal_pipeline	Every 5 minutes	Success	 ...	Mode: default 

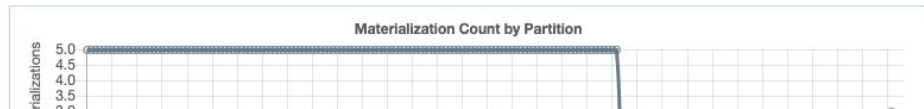
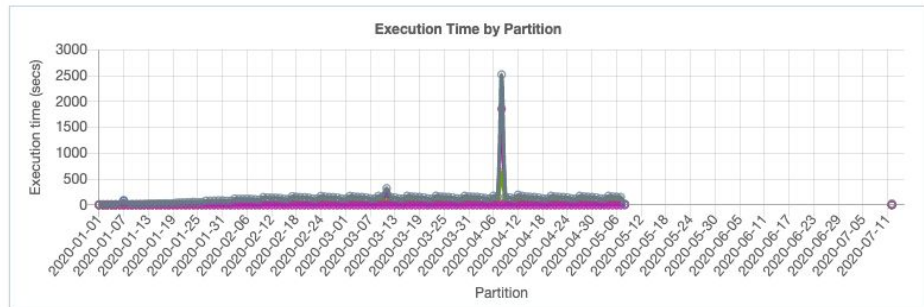
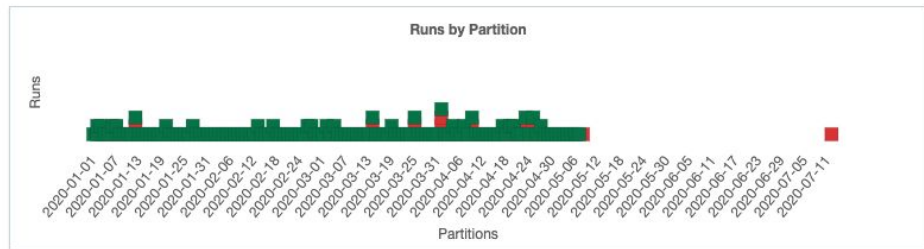
Schedules

SCHEDULE NAME	PIPELINE	SCHEDULE	LAST TICK	LATEST RUNS	EXECUTION PARAMS
<input type="checkbox"/> off longitudinal_demo	 longitudinal_pipeline	Every 5 minutes	Success	● ●●●●●●●● ...	Mode: default ▾

Partition Set: ingest_and_train

Last 7 Last 30 All

← Back Next →



Run filters

Filter...

Run steps

- Total pipeline
- build_cost_dashboard.compute
- build_model.compute
- ingest_costs.compute
- ingest_traffic.compute
- persist_costs.compute
- persist_traffic.compute

Dagster

- Programming model
- Tooling
- Platform for integrations

Built for interoperability



dbt —



Google Cloud Platform



AWS ECS

Deep integrations



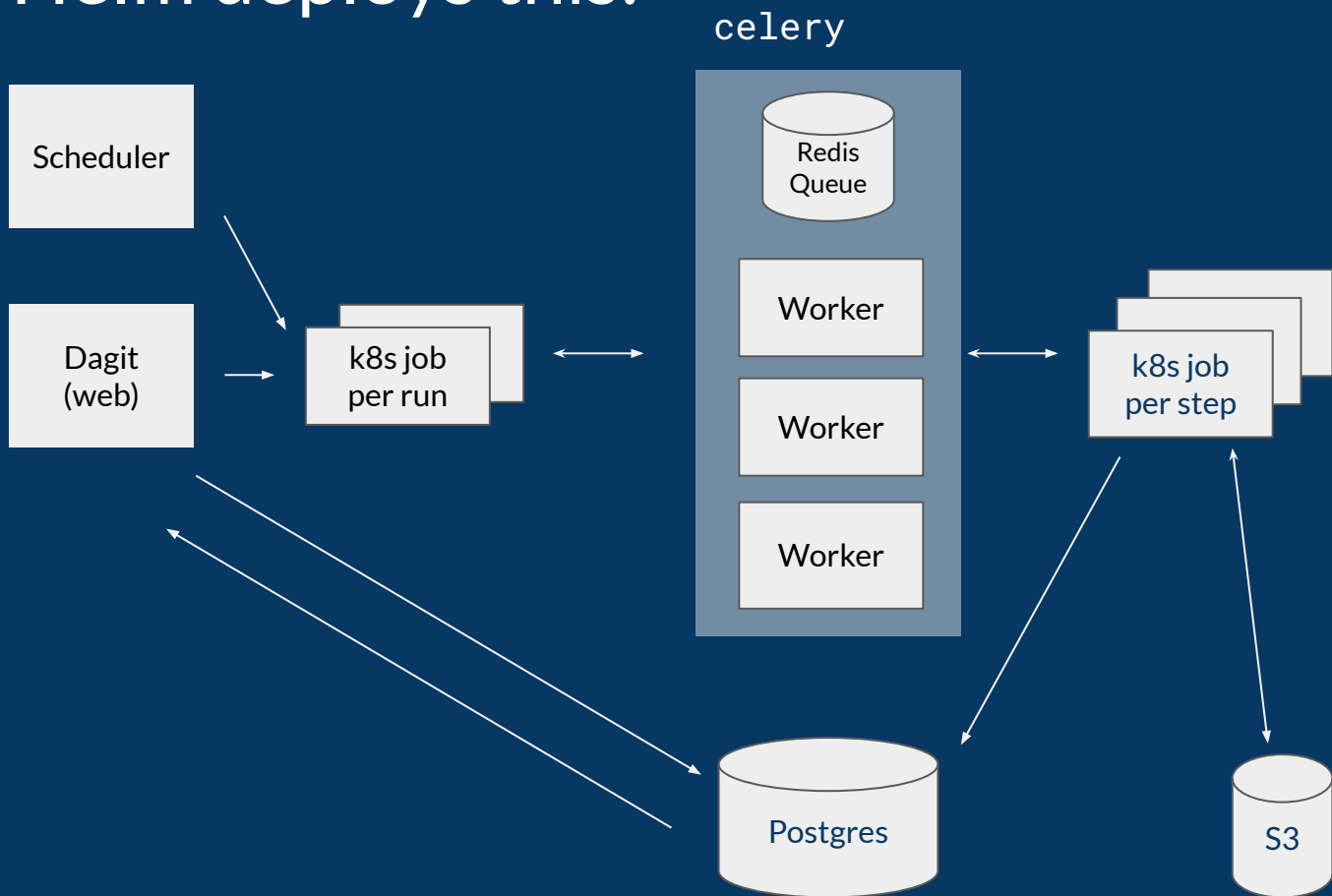
Easy to surface structured
metadata in Dagster

AssetMaterialization		A materialized node within the dbt graph.	
asset_key	model.dagster_dbt_test_project.sort_by_calories		
Node	[Show Metadata]		
Status	CREATE VIEW		
Execution Time	0.12479090690612793		
Materialization Strategy	view		
Database	test		
Schema	test-schema		
Alias	sort_by_calories		
Description	Sort the cereals table by calorie count		
Compilation Started At	2020-09-21T18:23:24+00:00		
Compilation Completed At	2020-09-21T18:23:24+00:00		
Compilation Duration	0:00:00.027323		
Execution Started At	2020-09-21T18:23:24+00:00		
Execution Completed At	2020-09-21T18:23:24+00:00		
Execution Duration	0:00:00.096249		

Pluggable infrastructure

- Deployment (local, cloud, k8s, PaaS, ...)
- Storage (local, s3, ...)
- Execution (multiprocess, celery, dask, ...)
- Scheduling (cron, Airflow,)
- Loggers (stdout, CloudWatch, Datadog, ...)

Or Helm deploys this:



Contribute!

- Open & active Slack
- Github Issues
- Github Discussions
- Welcoming & growing community

The screenshot shows a GitHub pull request interface for the pull request titled "Add Dask-Jobqueue clusters #2668". At the top, it indicates that the pull request has been merged by natekupp into the dagster-io:master branch from the DavidKatz-1l:add-dask-clusters branch on June 30. Below this, there are statistics for the pull request: 2 conversations, 1 commit, 5 checks, and 1 file changed. The main content of the pull request is a comment from DavidKatz-1l on June 30, stating "Add support for the following clusters: [moab , sge , tsf , slurm , oar].". This is followed by a review from natekupp on June 30, who left a comment asking if the user could remove some changes from the diff. DavidKatz-1l responded with a thumbs up and a thank you, stating they would make the requested change. Below the comments, the pull request details are shown, including the commit hash 93f5a88 and the fact that 5 checks passed. At the bottom, it shows that DavidKatz-1l deleted the branch on June 30.

Add Dask-Jobqueue clusters #2668

Merged natekupp merged 1 commit into dagster-io:master from DavidKatz-1l:add-dask-clusters on Jun 30

Conversation 2 Commits 1 Checks 5 Files changed 1

DavidKatz-1l commented on Jun 30 Contributor

Add support for the following clusters: [moab , sge , tsf , slurm , oar].

natekupp reviewed on Jun 30 View changes

natekupp left a comment Member

hey seems like these files in examples got changed inadvertently—can you remove these changes from the diff?

natekupp commented on Jun 30 Member

otherwise LGTM! if you can make that one change I'll approve and merge. Thanks!

1

Add dask_jobqueue clusters ✓ ca13493

DavidKatz-1l force-pushed the DavidKatz-1l:add-dask-clusters branch from 4638f79 to ca13493 on Jun 30

natekupp merged commit 93f5a88 into dagster-io:master on Jun 30 View details Revert

5 checks passed

DavidKatz-1l deleted the DavidKatz-1l:add-dask-clusters branch on Jun 30

Programming model



DAGSTER

Tooling

Platform



The Data Application Lifecycle

Develop

Test

Deploy

Operate



DAGSTER

Thank you!



Budapest Data Forum 2020
Max Gasner
@gasnerpants