DAGSTER

Building For The Data Application Lifecycle

Nick Schrock
Founder, Elementl

@schrockn

schrockn@elementl.com

Dagster is a Data Orchestrator

Orchestrator: Manages and orchestrates the graph of computations the comprise a data application.

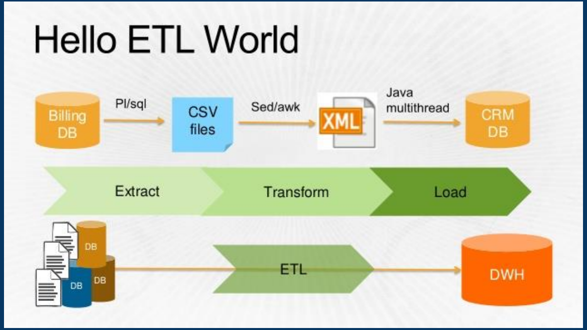
Data: Metadata- and Data-Aware.

Any runtime, any compute environment, any storage.

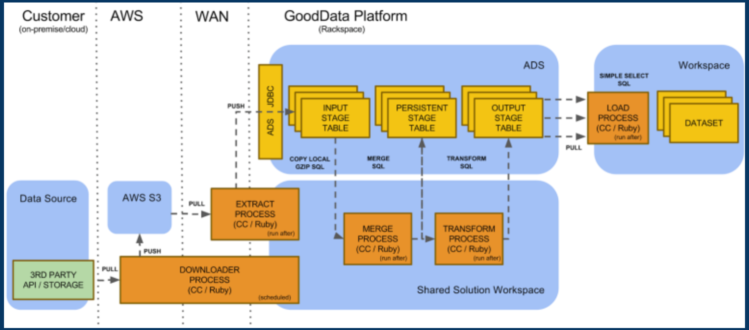
Data Application

Graph of Functional Computations
That Produce and Consume Data Assets

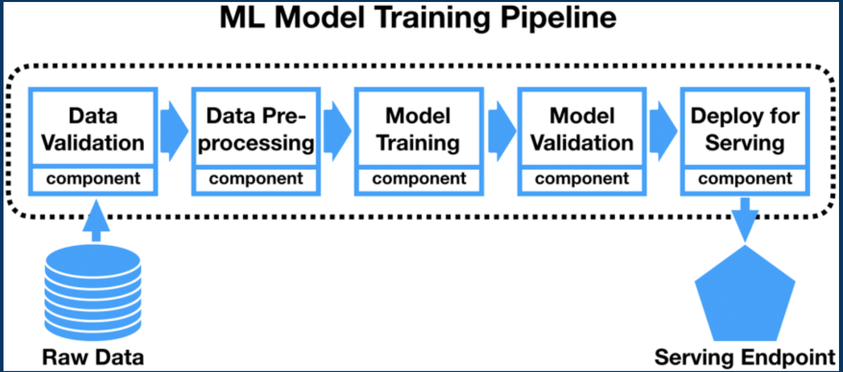
ETL



ELT



ML Pipeline


ML Model Training Pipeline
Data Validation component
Data Pre-processing component
Model Training component
Model Validation component
Deploy for Serving component
Raw Data
Serving Endpoint

All are graphs of computations that consume and produce data assets

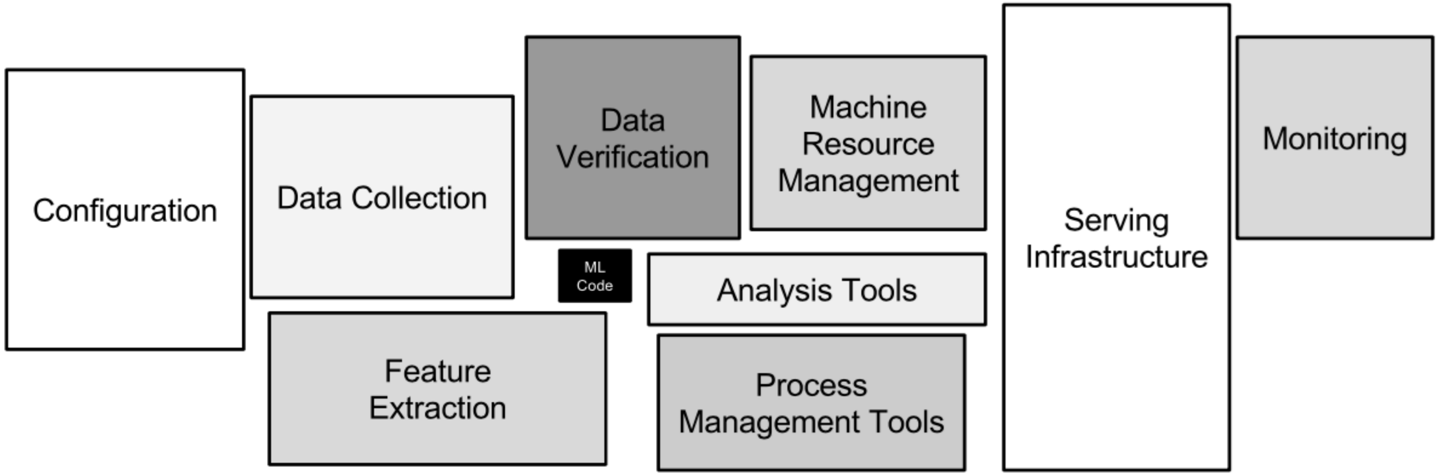
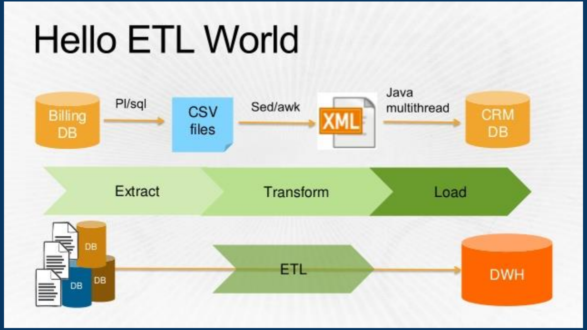


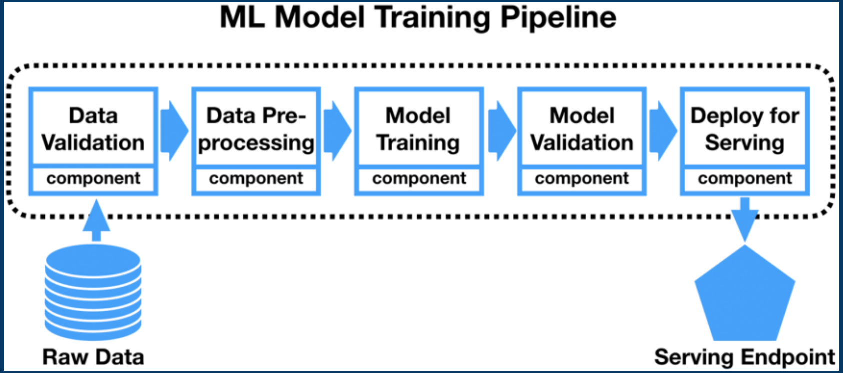
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

All components (except "ML Code" and "Serving") are common to *all* data applications

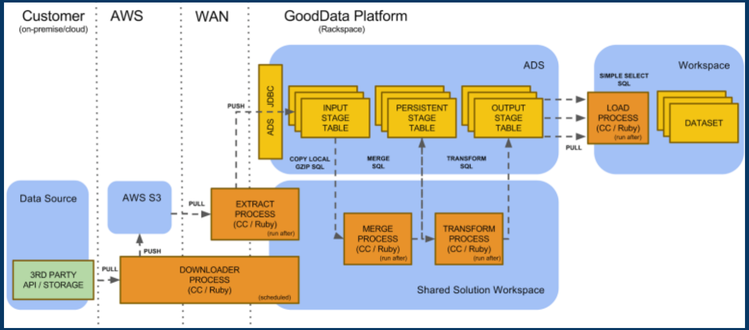
ETL



ML Pipeline



ELT



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

Data applications are:

Multi

persona

tool

team

environment

This is software engineering

The Data Application Lifecycle

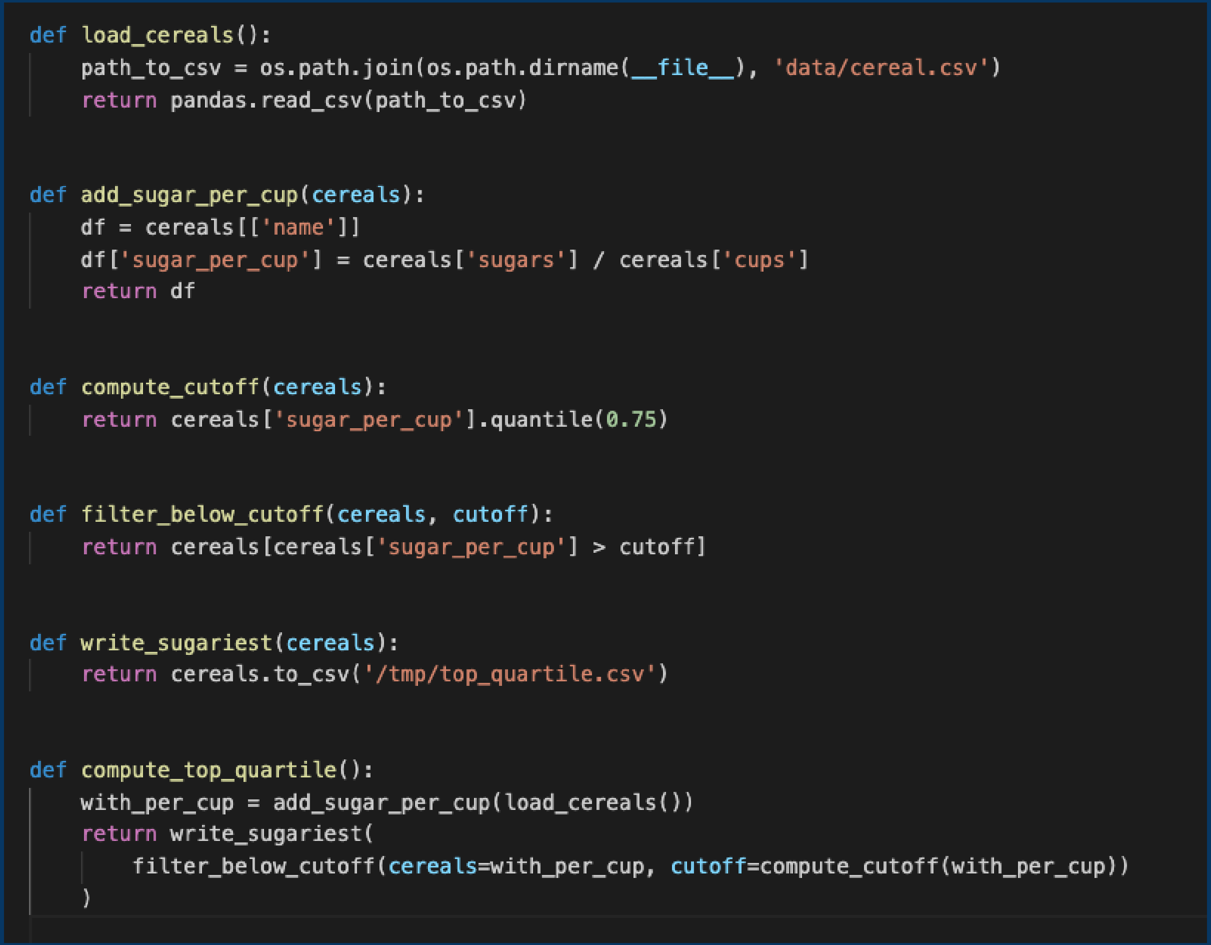
Develop

Test

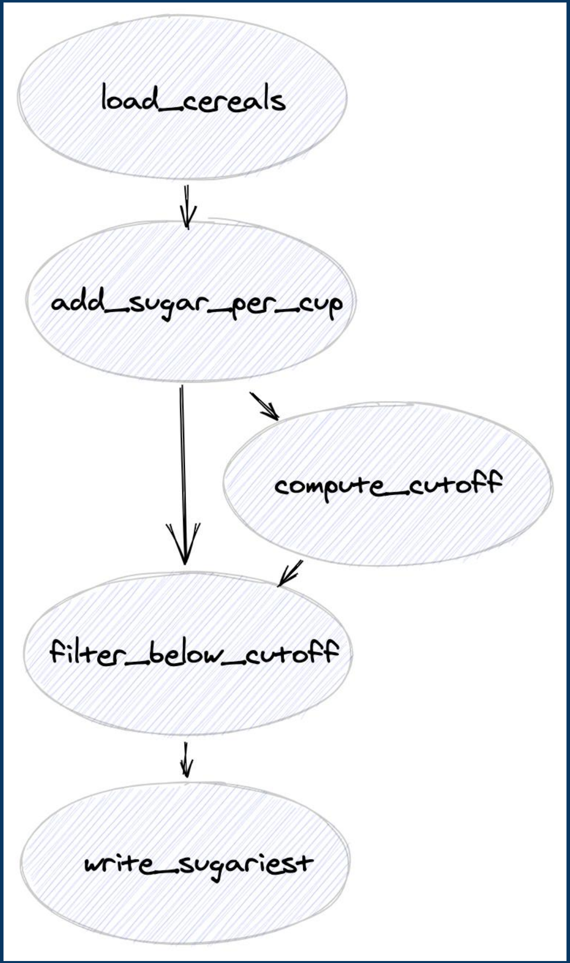
Deploy

Operate

Develop Test Deploy Operate

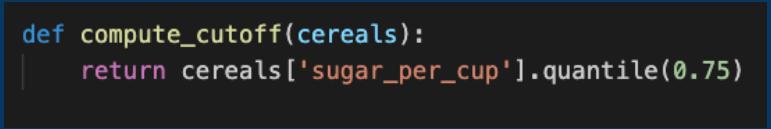


```python
def load_cereals():
    path_to_csv = os.path.join(os.path.dirname(__file__), 'data/cereal.csv')
    return pandas.read_csv(path_to_csv)


def add_sugar_per_cup(cereals):
    df = cereals[['name']]
    df['sugar_per_cup'] = cereals['sugars'] / cereals['cups']
    return df


def compute_cutoff(cereals):
    return cereals['sugar_per_cup'].quantile(0.75)


def filter_below_cutoff(cereals, cutoff):
    return cereals[cereals['sugar_per_cup'] > cutoff]


def write_sugariest(cereals):
    return cereals.to_csv('/tmp/top_quartile.csv')


def compute_top_quartile():
    with_per_cup = add_sugar_per_cup(load_cereals())
    return write_sugariest(
        filter_below_cutoff(cereals=with_per_cup, cutoff=compute_cutoff(with_per_cup))
    )
```

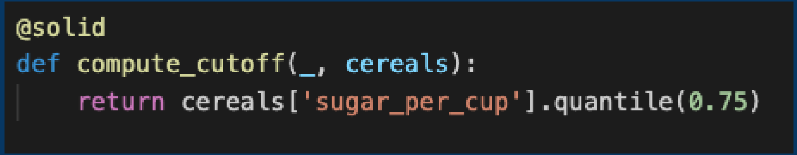


$$\begin{tikzcd}
\text{load\_cereals} \arrow[d] & \\
\text{add\_sugar\_per\_cup} \arrow[d] \arrow[r] & \text{compute\_cutoff} \arrow[dl] \\
\text{filter\_below\_cutoff} \arrow[d] & \\
\text{write\_sugariest} &
\end{tikzcd}$$

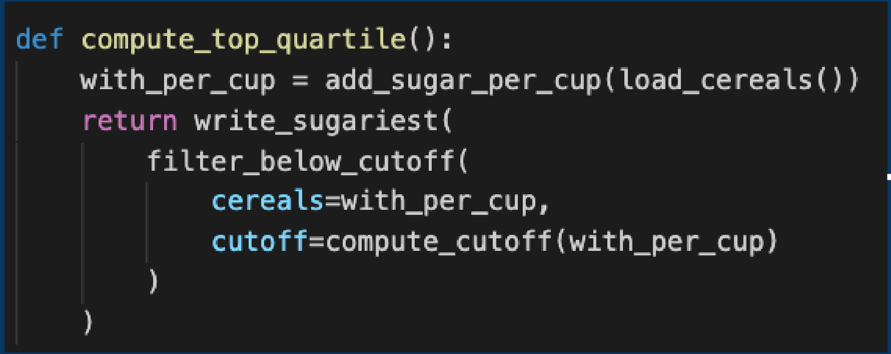
```
> pip install dagster
```



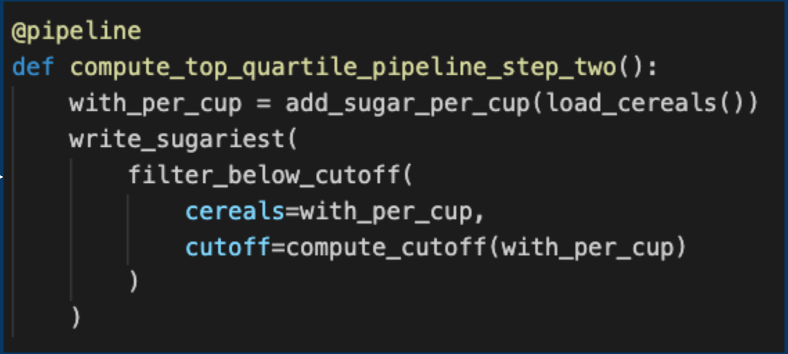
```python
def compute_cutoff(cereals):
    return cereals['sugar_per_cup'].quantile(0.75)
```



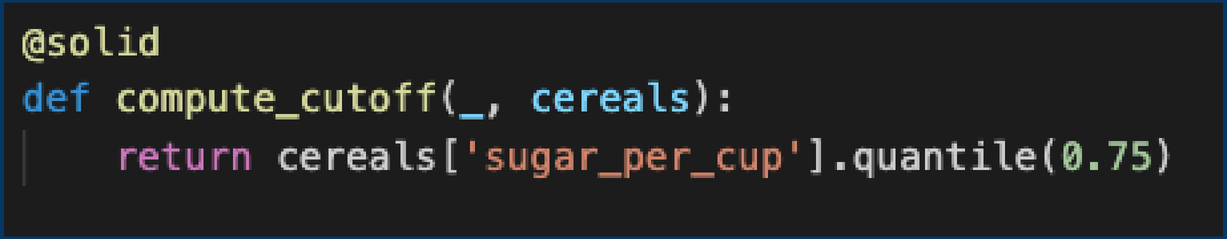
```python
@solid
def compute_cutoff(_, cereals):
    return cereals['sugar_per_cup'].quantile(0.75)
```



```python
def compute_top_quartile():
    with_per_cup = add_sugar_per_cup(load_cereals())
    return write_sugariest(
        filter_below_cutoff(
            cereals=with_per_cup,
            cutoff=compute_cutoff(with_per_cup)
        )
    )
```

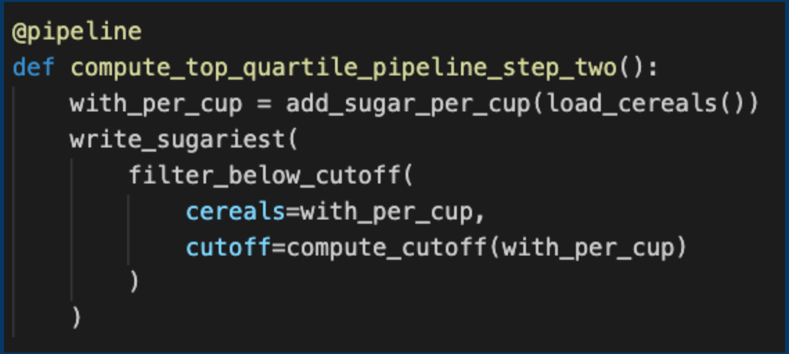


```python
@pipeline
def compute_top_quartile_pipeline_step_two():
    with_per_cup = add_sugar_per_cup(load_cereals())
    write_sugariest(
        filter_below_cutoff(
            cereals=with_per_cup,
            cutoff=compute_cutoff(with_per_cup)
        )
    )
```



```python
@solid
def compute_cutoff(_, cereals):
    return cereals['sugar_per_cup'].quantile(0.75)
```

- Solid: a functional unit of computation in the orchestration graph
- Designed for reuse and testability


```python
@pipeline
def compute_top_quartile_pipeline_step_two():
    with_per_cup = add_sugar_per_cup(load_cereals())
    write_sugariest(
        filter_below_cutoff(
            cereals=with_per_cup,
            cutoff=compute_cutoff(with_per_cup)
        )
    )
```

- Pipeline is a graph of solids
- Connected via data dependencies

```
> pip install dagit
      && dagit
```

DEMO

We wanted flying cars, instead we got

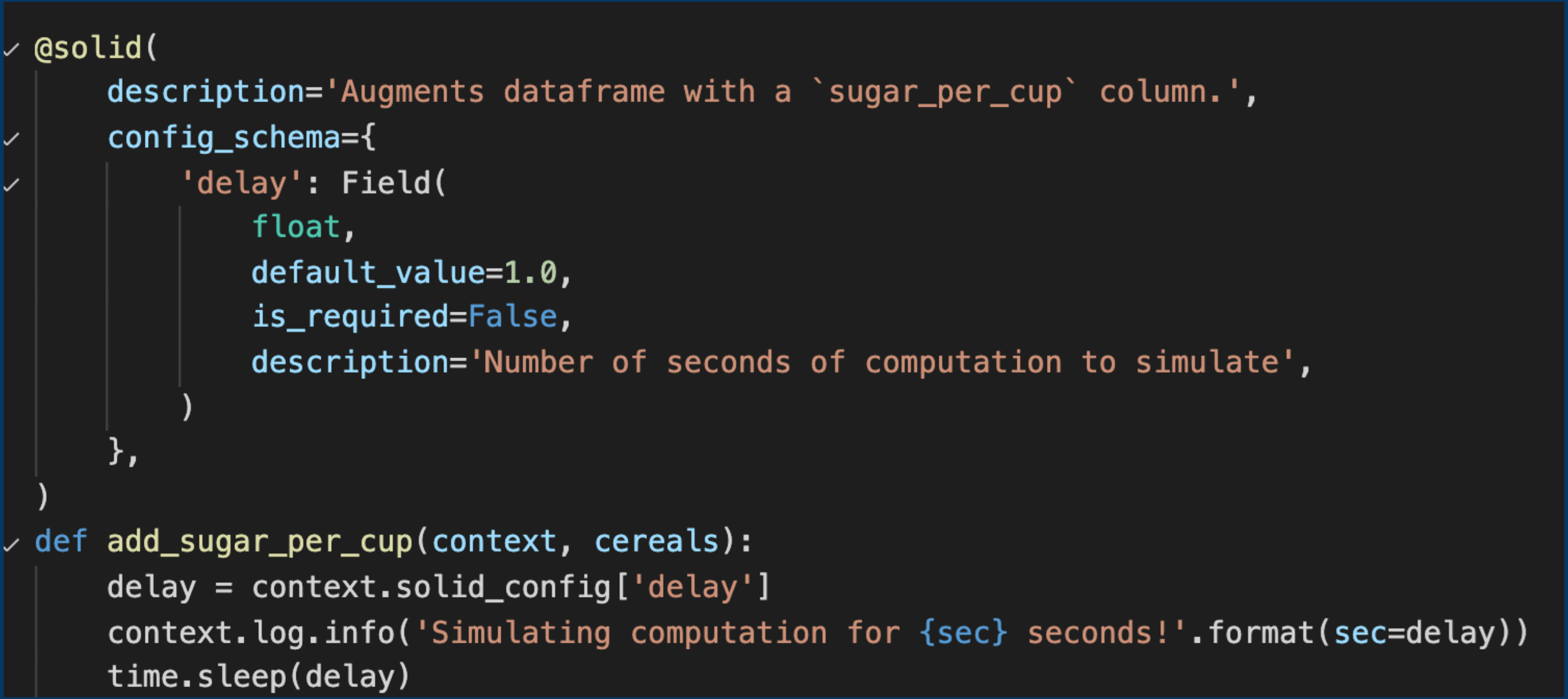
cha s.

Data applications are no exception:
lots of configuration

`dagster.config`

- Schema over python dictionaries
- Self-describing
- High quality error messages
- Catch errors earlier
- Autocompleting YAML editor

Use Config To Make UI Demo Better



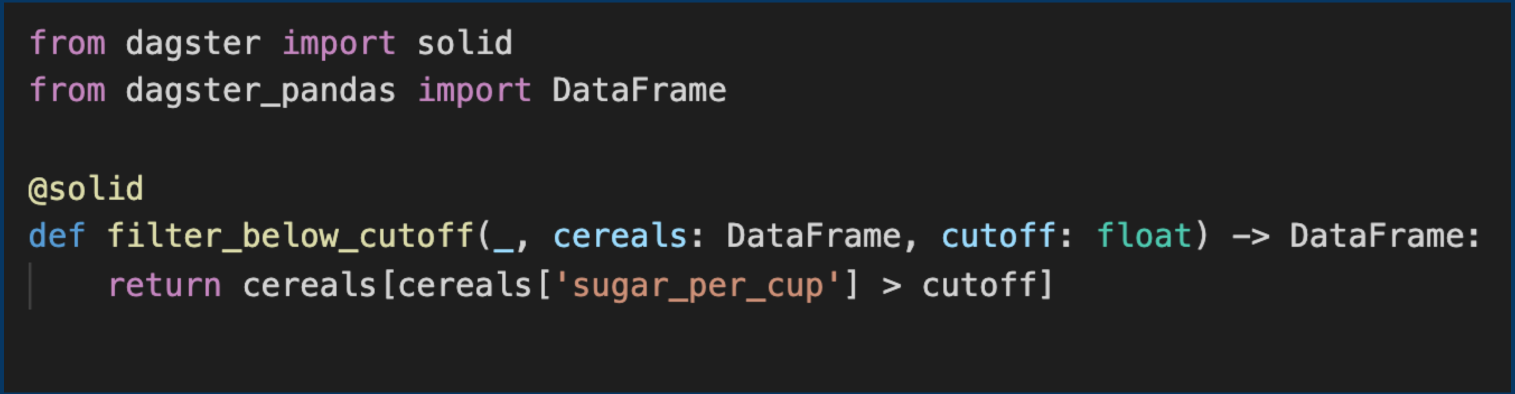
```
@solid(
    description='Augments dataframe with a `sugar_per_cup` column.',
    config_schema={
        'delay': Field(
            float,
            default_value=1.0,
            is_required=False,
            description='Number of seconds of computation to simulate',
        )
    },
)
def add_sugar_per_cup(context, cereals):
    delay = context.solid_config['delay']
    context.log.info('Simulating computation for {sec} seconds!'.format(sec=delay))
    time.sleep(delay)
```

DEMO

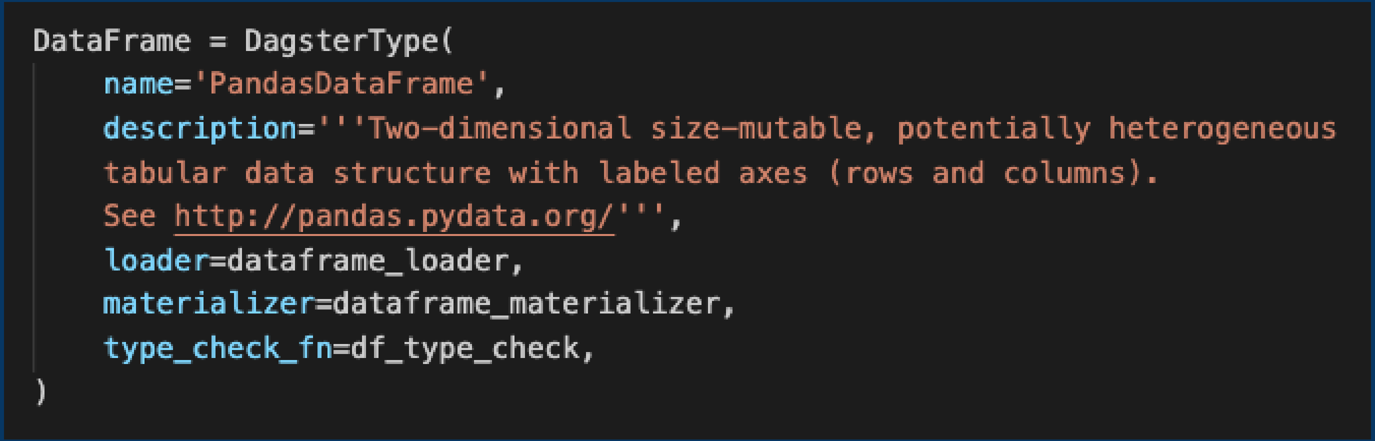
Dagster Types

Gradual, Optional: Defaults to Any

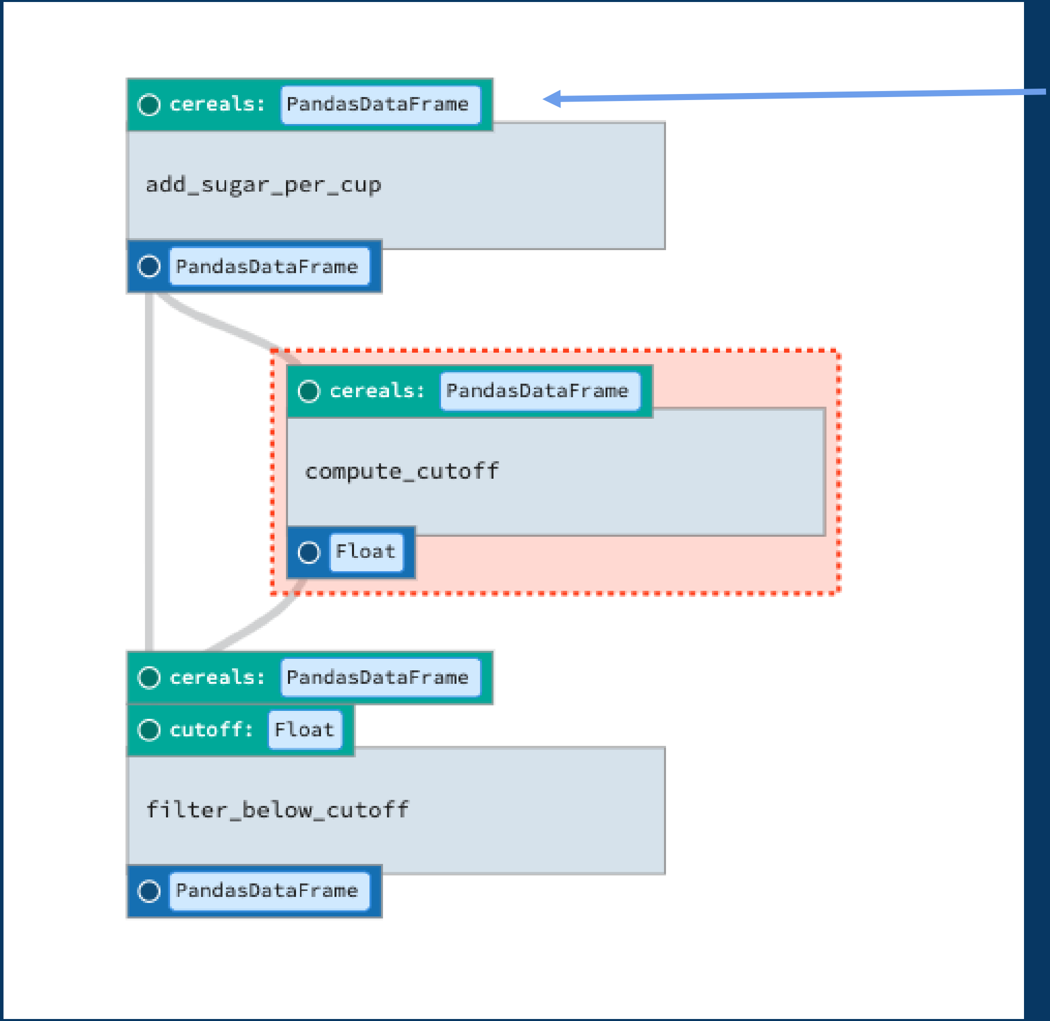
Flexible: Typecheck is a arbitrary code.



```python
from dagster import solid
from dagster_pandas import DataFrame

@solid
def filter_below_cutoff(_, cereals: DataFrame, cutoff: float) -> DataFrame:
    return cereals[cereals['sugar_per_cup'] > cutoff]
```

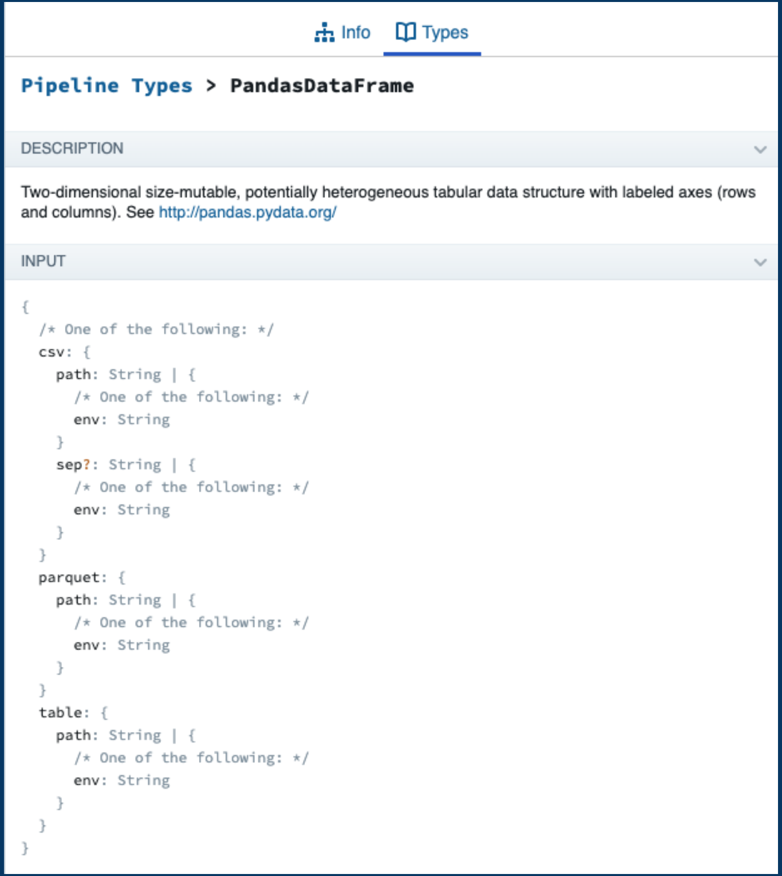


```python
DataFrame = DagsterType(
    name='PandasDataFrame',
    description='''Two-dimensional size-mutable, potentially heterogeneous
    tabular data structure with labeled axes (rows and columns).
    See http://pandas.pydata.org/''',
    loader=dataframe_loader,
    materializer=dataframe_materializer,
    type_check_fn=df_type_check,
)
```



Where does this value come from?

config!



Pipeline Types > PandasDataFrame

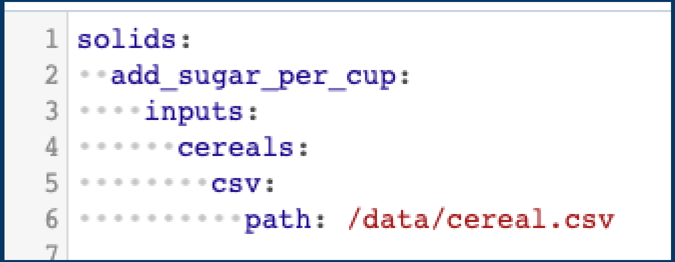
DESCRIPTION

Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). See http://pandas.pydata.org/

INPUT

```
{
  /* One of the following: */
  csv: {
    path: String | {
      /* One of the following: */
      env: String
    }
    sep?: String | {
      /* One of the following: */
      env: String
    }
  }
  parquet: {
    path: String | {
      /* One of the following: */
      env: String
    }
  }
  table: {
    path: String | {
      /* One of the following: */
      env: String
    }
  }
}
```

DEMO

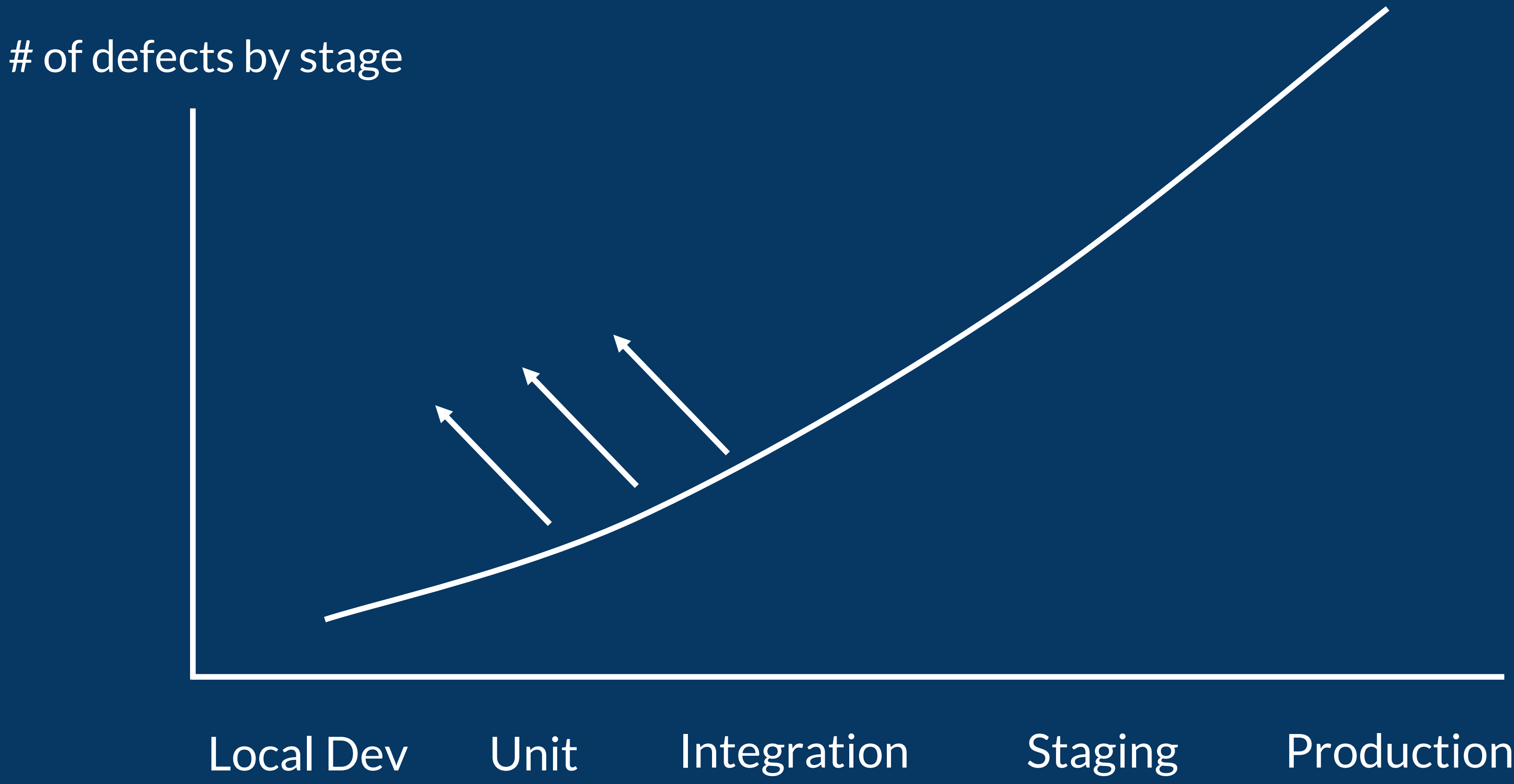


```
solids:
  add_sugar_per_cup:
    inputs:
      cereals:
        csv:
          path: /data/cereal.csv
```

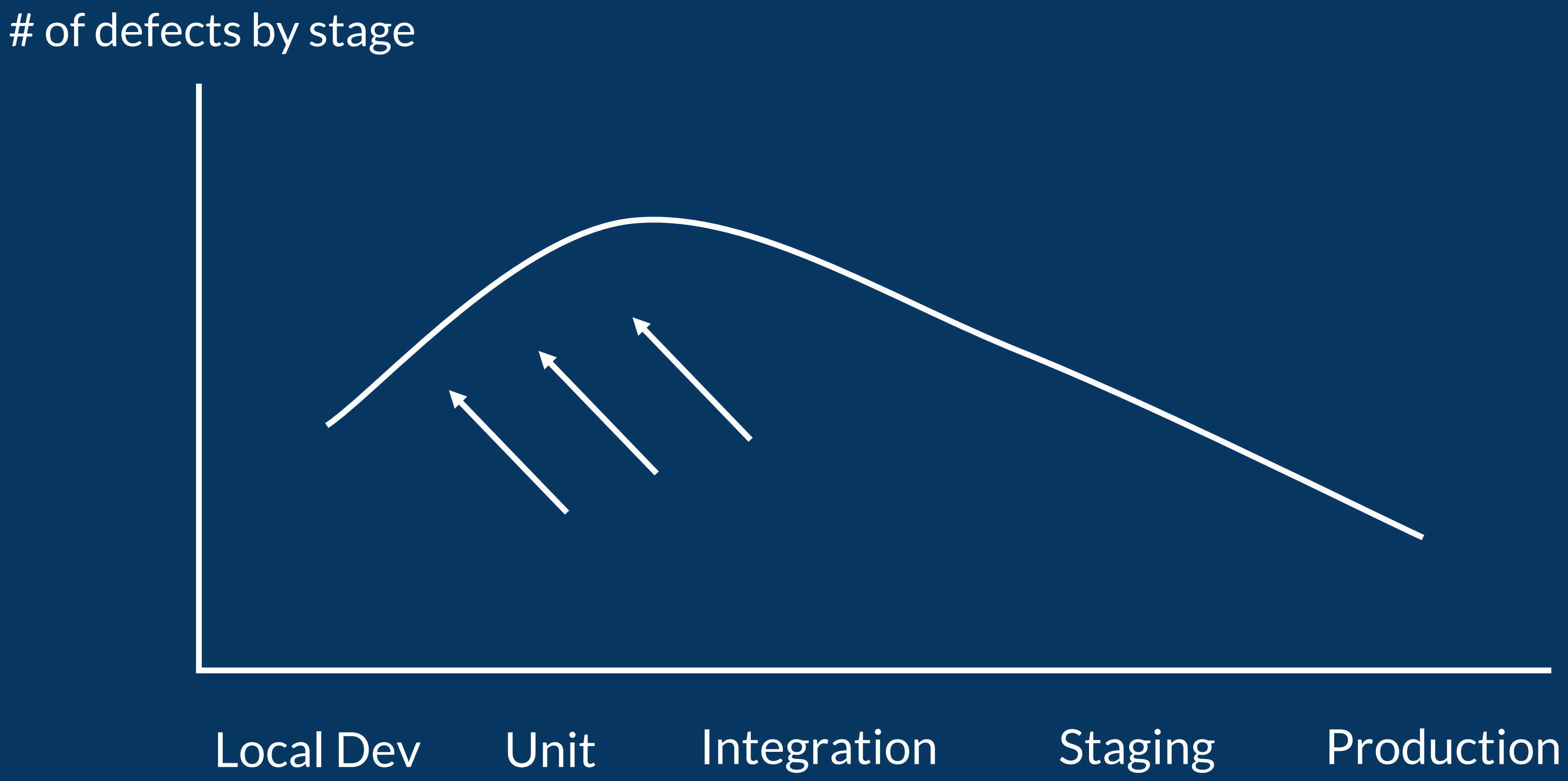
Develop Test Deploy Operate

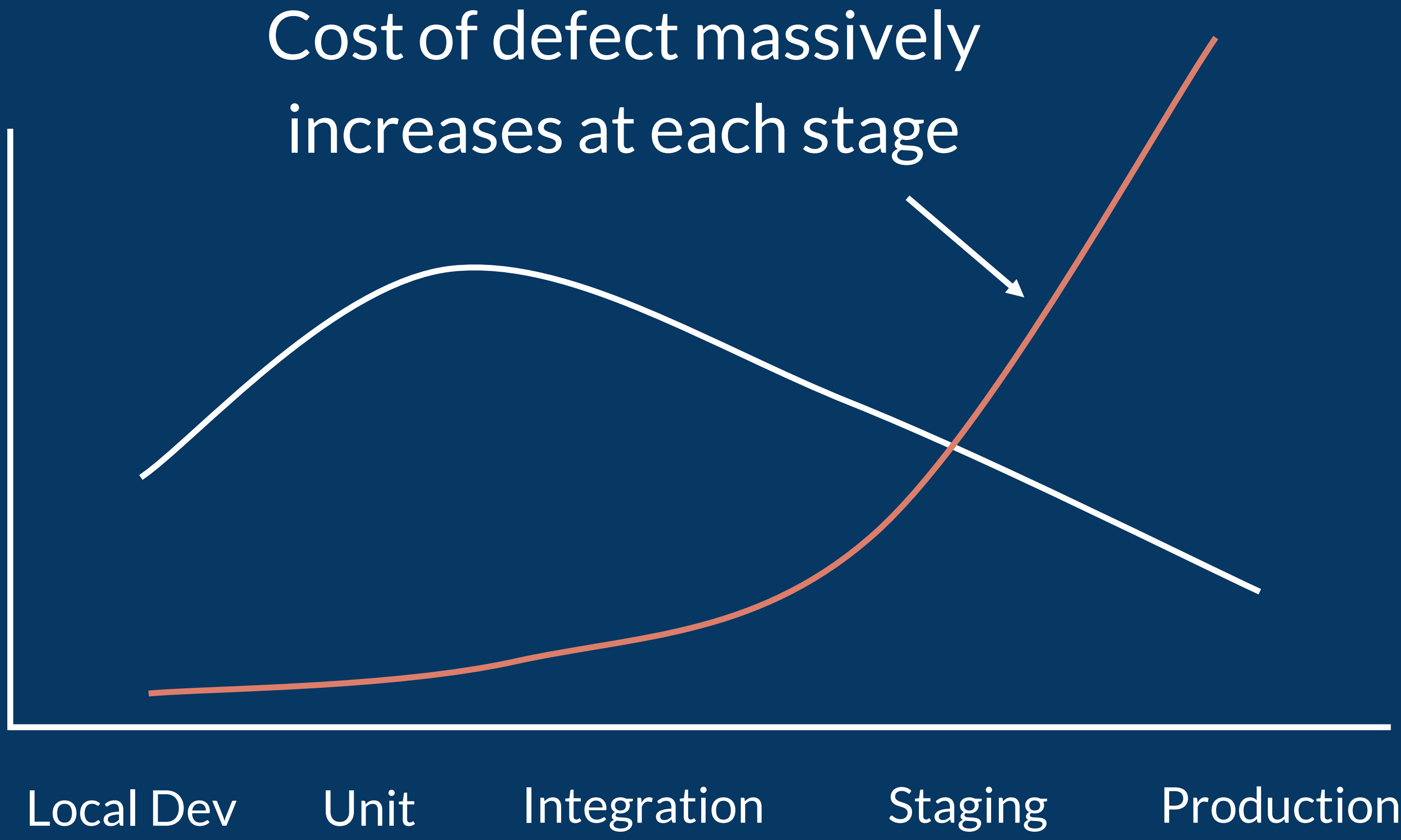
Testing data applications is *uniquely* challenging

Status Quo: Late Defect Detection



Attainable Goal: Bend the Curve




Cost of defect massively increases at each stage
Local Dev
Unit
Integration
Staging
Production

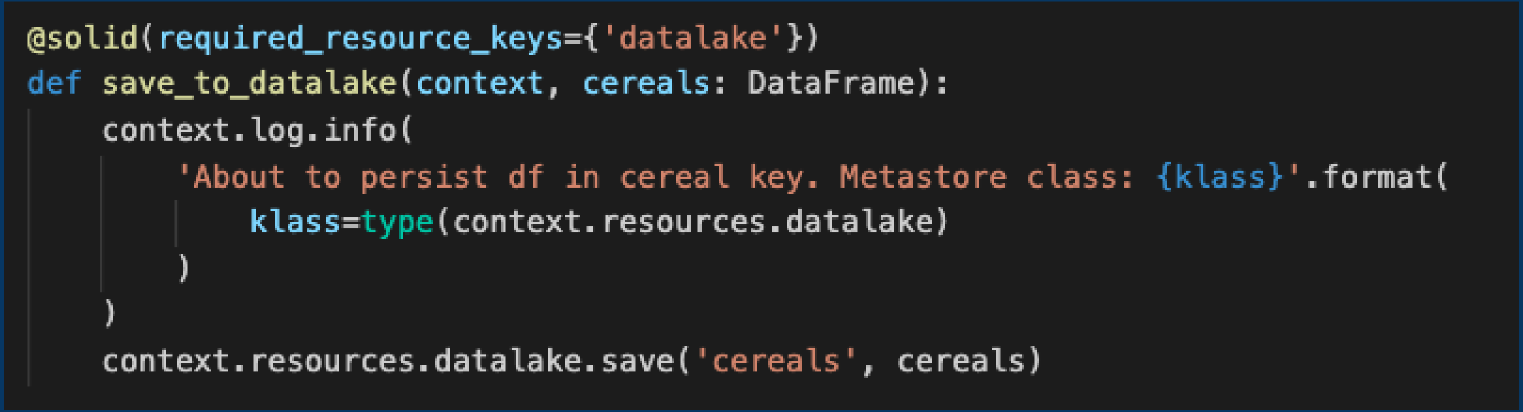
Moving defect detection earlier in the process

Order of magnitude improvement
in productivity and costs.

Foundations of Testability

Parameterized Computations	`@solid`
Pluggable Environment	`@resource`
Pluggable System (i.e. Dagster itself)	`dagster.yaml`

@resource

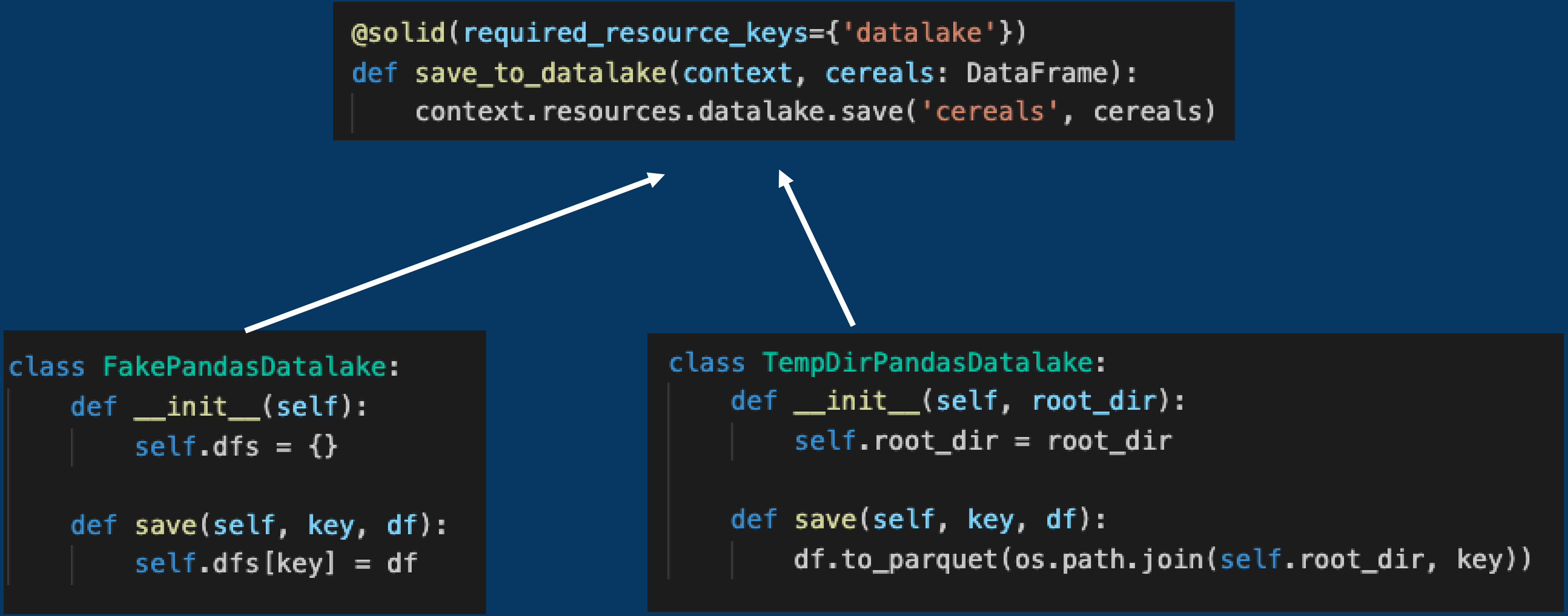


```python
@solid(required_resource_keys={'datalake'})
def save_to_datalake(context, cereals: DataFrame):
    context.log.info(
        'About to persist df in cereal key. Metastore class: {klass}'.format(
            klass=type(context.resources.datalake)
        )
    )
    context.resources.datalake.save('cereals', cereals)
```

Resources and context are the way you separate your business logic from your environment

Solids declare what resources they need

@resource



```
@solid(required_resource_keys={'datalake'})
def save_to_datalake(context, cereals: DataFrame):
    context.resources.datalake.save('cereals', cereals)
```

```
class FakePandasDatalake:
    def __init__(self):
        self.dfs = {}

    def save(self, key, df):
        self.dfs[key] = df
```

```
class TempDirPandasDatalake:
    def __init__(self, root_dir):
        self.root_dir = root_dir

    def save(self, key, df):
        df.to_parquet(os.path.join(self.root_dir, key))
```

Which resource is provided depends on mode

DEMO

Develop Test Deploy Operate

Remember?

Pluggable System
(i.e. Dagster itself)

Key to *both* testing *and* flexible deployment

Example deployment using helm:

Database → Postgres
Intermediate Storage → S3
Execution Substrate → Celery + Kubernetes

Dockerfile

```
FROM "python-3.7.8-slim"

RUN pip install \
    dagster \
    dagit \
    dagster-k8s # … and others

ADD your_project .
```

```
> docker push preso-dc-2020
```

values.yaml

```
dagit:
 image:
   repository: "dagster/preso-dc-2020"
   tag: "latest"

pipeline_run:
 image:
   repository: "dagster/preso-dc-2020"
   tag: "latest"
```

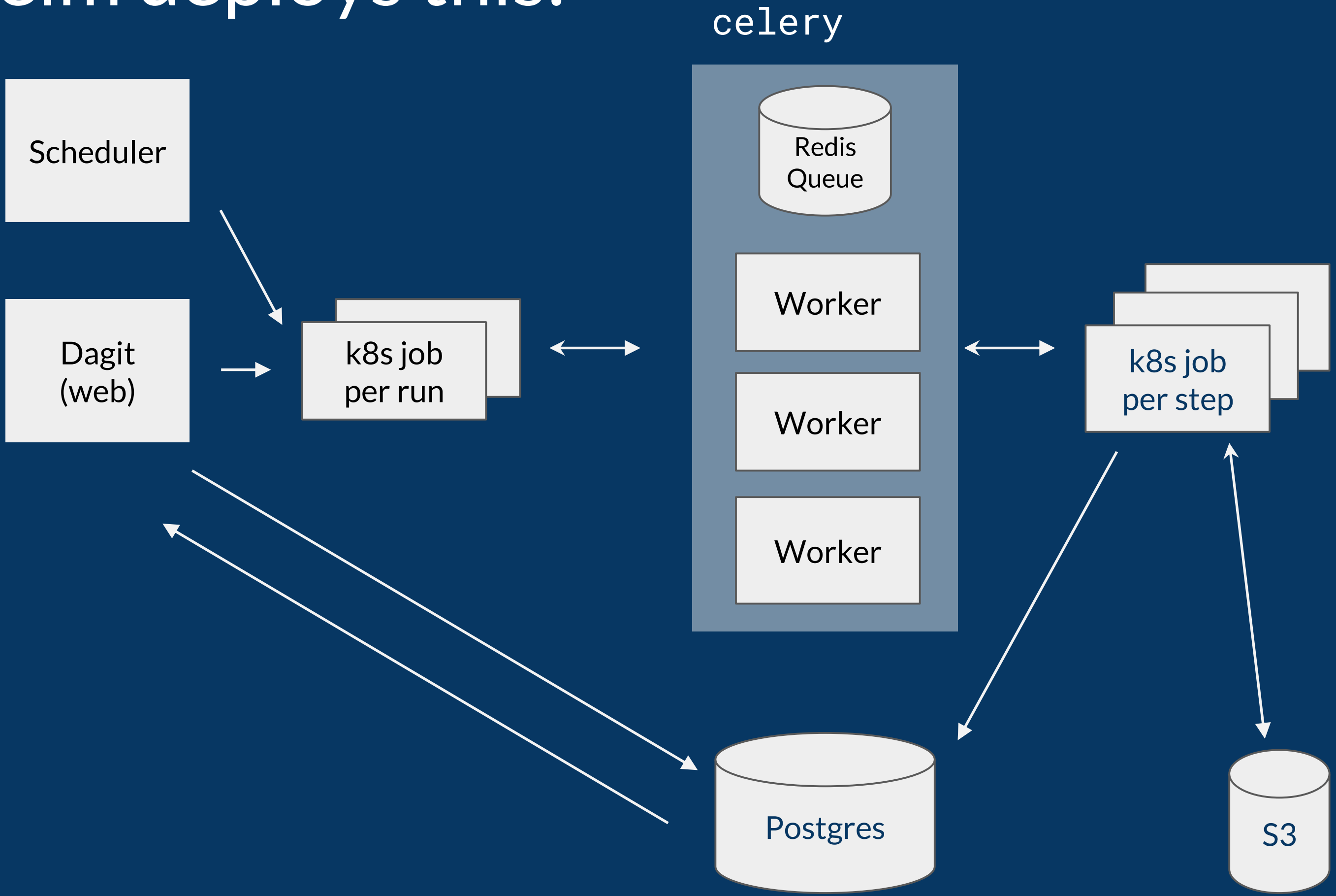
```
> helm install dagster helm/dagster -f /path/to/your/values.yaml
```

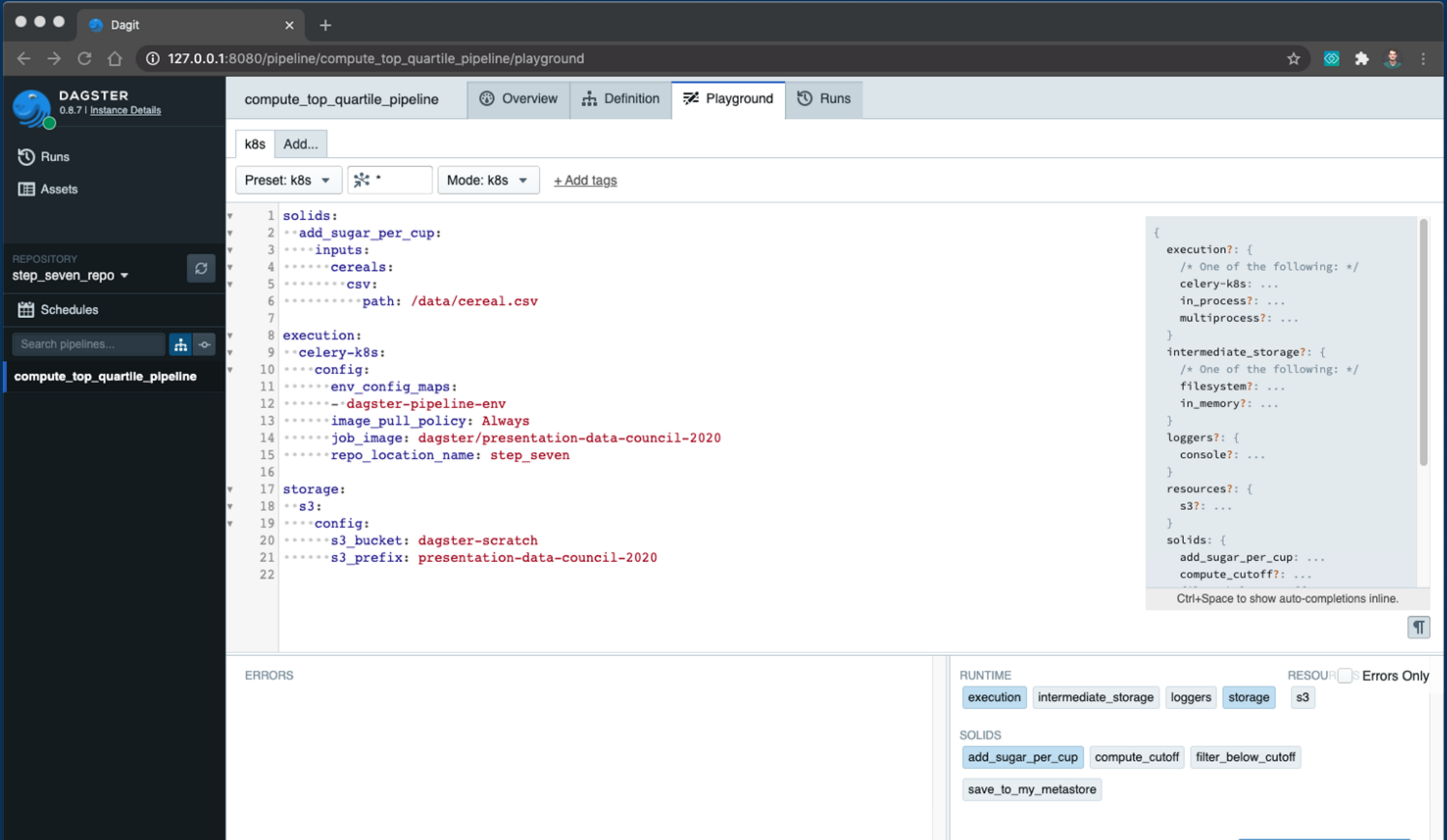
dagster.yaml (generated by helm)

```yaml
run_launcher:
  module: dagster_celery_k8s
  class: CeleryK8sRunLauncher
  config:
    dagster_home:
      env: DAGSTER_HOME
    instance_config_map:
      env: DAGSTER_K8S_INSTANCE_CONFIG_MAP
    postgres_password_secret:
      env: DAGSTER_K8S_PG_PASSWORD_SECRET
    broker:  "pyamqp://test:test@dagster-rabbitmq:5672//"
    backend: "amqp"

# Additional Sections
# run_storage: RDS postgres info
# event_log_storage: ...
# schedule_storage: ...
# etc many pluggable sections
```

Completely user pluggable

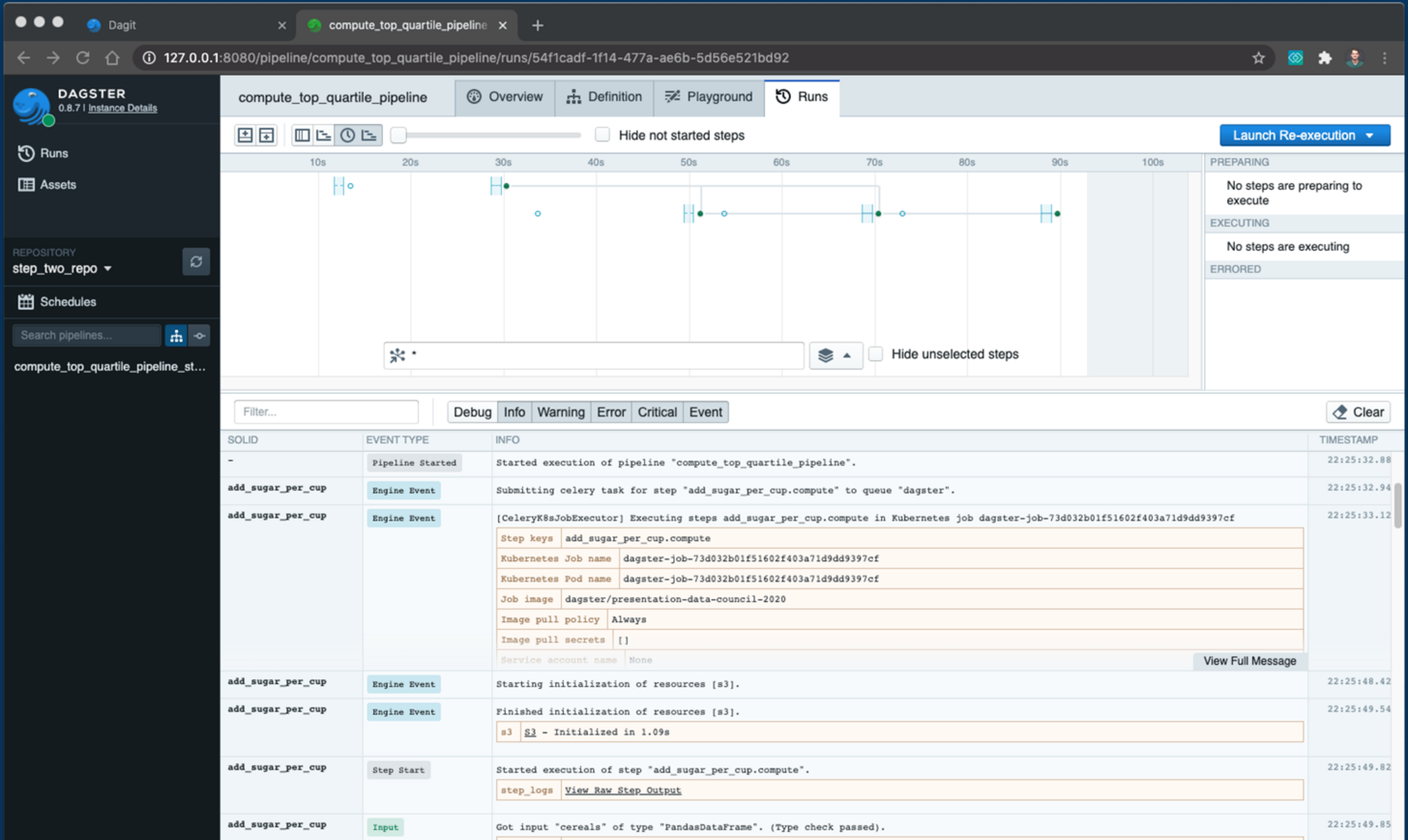
Helm deploys this:





```yaml
solids:
  add_sugar_per_cup:
    inputs:
      cereals:
        csv:
          path: /data/cereal.csv

execution:
  celery-k8s:
    config:
      env_config_maps:
      - dagster-pipeline-env
      image_pull_policy: Always
      job_image: dagster/presentation-data-council-2020
      repo_location_name: step_seven

storage:
  s3:
    config:
      s3_bucket: dagster-scratch
      s3_prefix: presentation-data-council-2020

```

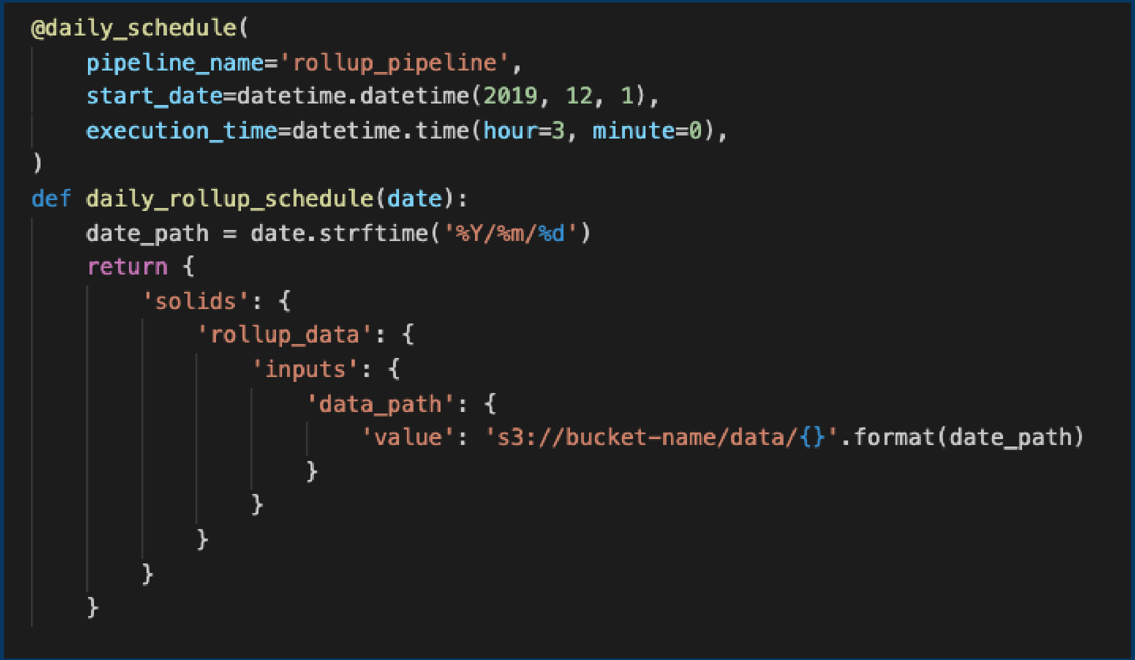
```
{
  execution?: {
    /* One of the following: */
    celery-k8s: ...
    in_process?: ...
    multiprocess?: ...
  }
  intermediate_storage?: {
    /* One of the following: */
    filesystem?: ...
    in_memory?: ...
  }
  loggers?: {
    console?: ...
  }
  resources?: {
    s3?: ...
  }
  solids: {
    add_sugar_per_cup: ...
    compute_cutoff?: ...
```



SOLID	EVENT TYPE	INFO	TIMESTAMP
-	Pipeline Started	Started execution of pipeline "compute_top_quartile_pipeline".	22:25:32.88
add_sugar_per_cup	Engine Event	Submitting celery task for step "add_sugar_per_cup.compute" to queue "dagster".	22:25:32.94
add_sugar_per_cup	Engine Event	[CeleryK8sJobExecutor] Executing steps add_sugar_per_cup.compute in Kubernetes job dagster-job-73d032b01f51602f403a71d9dd9397cf Step keys: add_sugar_per_cup.compute Kubernetes Job name: dagster-job-73d032b01f51602f403a71d9dd9397cf Kubernetes Pod name: dagster-job-73d032b01f51602f403a71d9dd9397cf Job image: dagster/presentation-data-council-2020 Image pull policy: Always Image pull secrets: [] Service account name: None	22:25:33.12
add_sugar_per_cup	Engine Event	Starting initialization of resources [s3].	22:25:48.42
add_sugar_per_cup	Engine Event	Finished initialization of resources [s3]. s3: S3 - Initialized in 1.09s	22:25:49.54
add_sugar_per_cup	Step Start	Started execution of step "add_sugar_per_cup.compute". step_logs: View Raw Step Output	22:25:49.82
add_sugar_per_cup	Input	Got input "cereals" of type "PandasDataFrame". (Type check passed).	22:25:49.85

Develop Test Deploy **Operate**

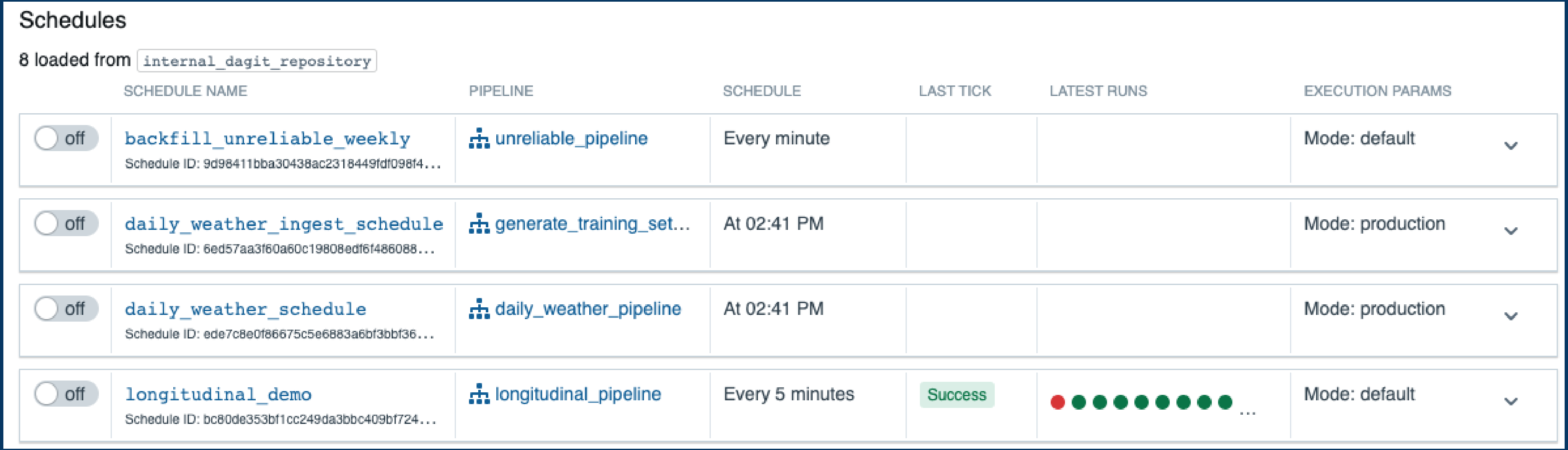
Schedule



```python
@daily_schedule(
    pipeline_name='rollup_pipeline',
    start_date=datetime.datetime(2019, 12, 1),
    execution_time=datetime.time(hour=3, minute=0),
)
def daily_rollup_schedule(date):
    date_path = date.strftime('%Y/%m/%d')
    return {
        'solids': {
            'rollup_data': {
                'inputs': {
                    'data_path': {
                        'value': 's3://bucket-name/data/{}'.format(date_path)
                    }
                }
            }
        }
    }
```

fn(time) → run_config

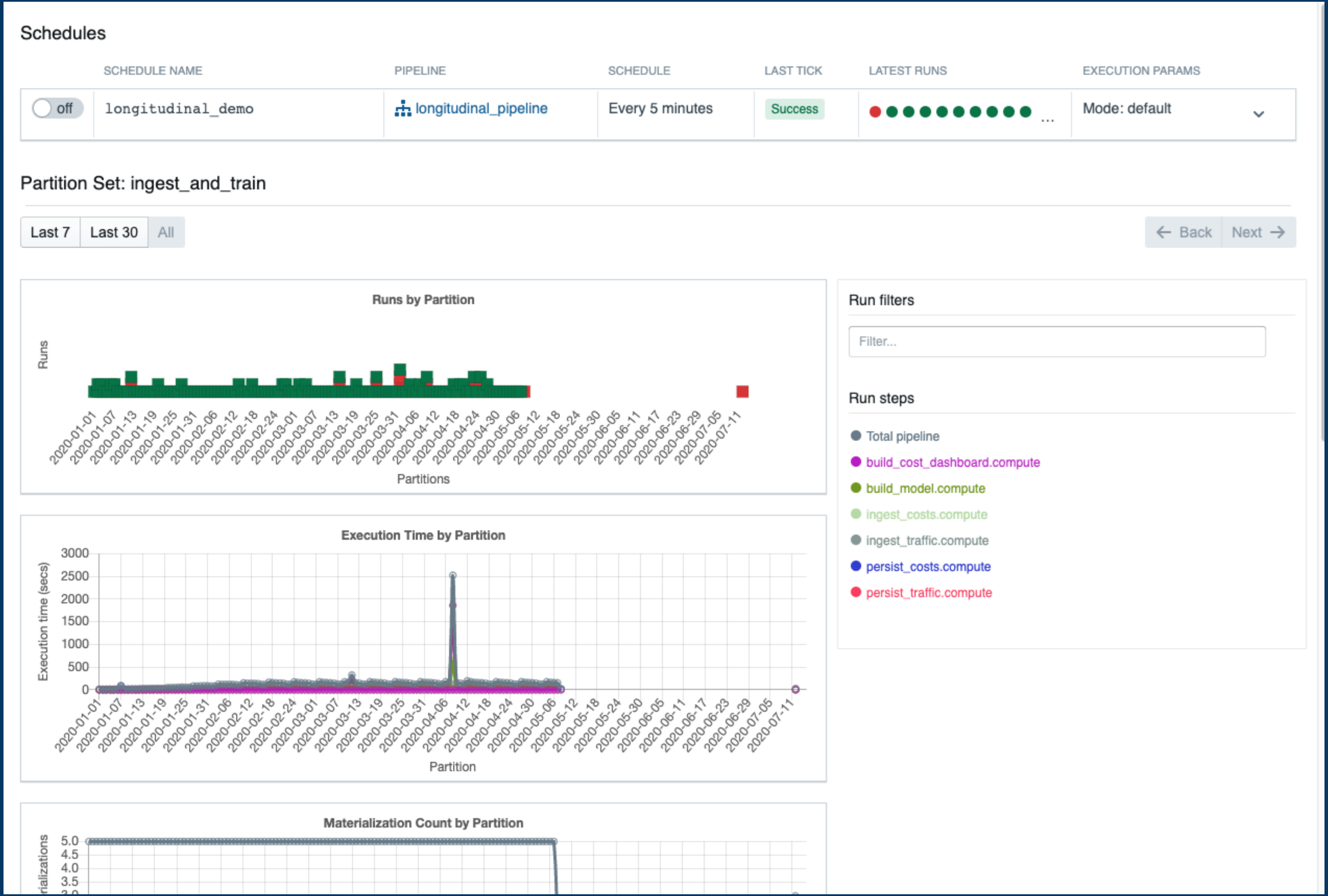
Scheduler



Schedules

8 loaded from `internal_dagit_repository`

	SCHEDULE NAME	PIPELINE	SCHEDULE	LAST TICK	LATEST RUNS	EXECUTION PARAMS
off	`backfill_unreliable_weekly` Schedule ID: 9d98411bba30438ac2318449fdf098f4…	unreliable_pipeline	Every minute			Mode: default
off	`daily_weather_ingest_schedule` Schedule ID: 6ed57aa3f60a60c19808edf6f486088…	generate_training_set…	At 02:41 PM			Mode: production
off	`daily_weather_schedule` Schedule ID: ede7c8e0f86675c5e6883a6bf3bbf36…	daily_weather_pipeline	At 02:41 PM			Mode: production
off	`longitudinal_demo` Schedule ID: bc80de353bf1cc249da3bbc409bf724…	longitudinal_pipeline	Every 5 minutes	Success	🟥 ● ● ● ● ● ● ● ● …	Mode: default



Schedules

	SCHEDULE NAME	PIPELINE	SCHEDULE	LAST TICK	LATEST RUNS	EXECUTION PARAMS
off	`longitudinal_demo`	longitudinal_pipeline	Every 5 minutes	Success	🟥●●●●●●●●●● …	Mode: default

Partition Set: ingest_and_train

Last 7 | Last 30 | All

← Back | Next →

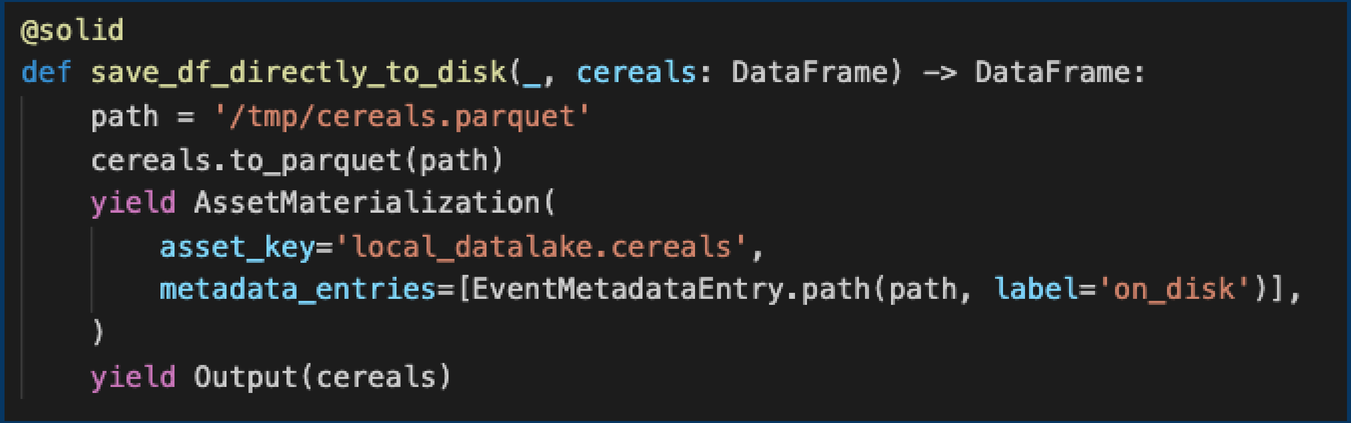
Run filters

Filter...

Run steps

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

Asset Management



```python
@solid
def save_df_directly_to_disk(_, cereals: DataFrame) -> DataFrame:
    path = '/tmp/cereals.parquet'
    cereals.to_parquet(path)
    yield AssetMaterialization(
        asset_key='local_datalake.cereals',
        metadata_entries=[EventMetadataEntry.path(path, label='on_disk')],
    )
    yield Output(cereals)
```

A solid yields a stream of events

`AssetMaterialization`: indicates an asset has been created that will outlive computation

DEMO

The Data Application Lifecycle

Develop

Test

Deploy

Operate

A generalized platform

Not just k8s + pandas!

dbt_

jupyter

APACHE Spark™

DASK
Dask natively scales Python

Google Cloud Platform

AWS ECS

DAGSTER

https://github.com/dagster-io/dagster

Find the slack invite and say hi!