Minimum Viable Data Products

From 0 to Data Science Pipeline

Daniel Imberman

Gonzalo Diaz

Rappi

ASTRONOMER

What is a MVDP?

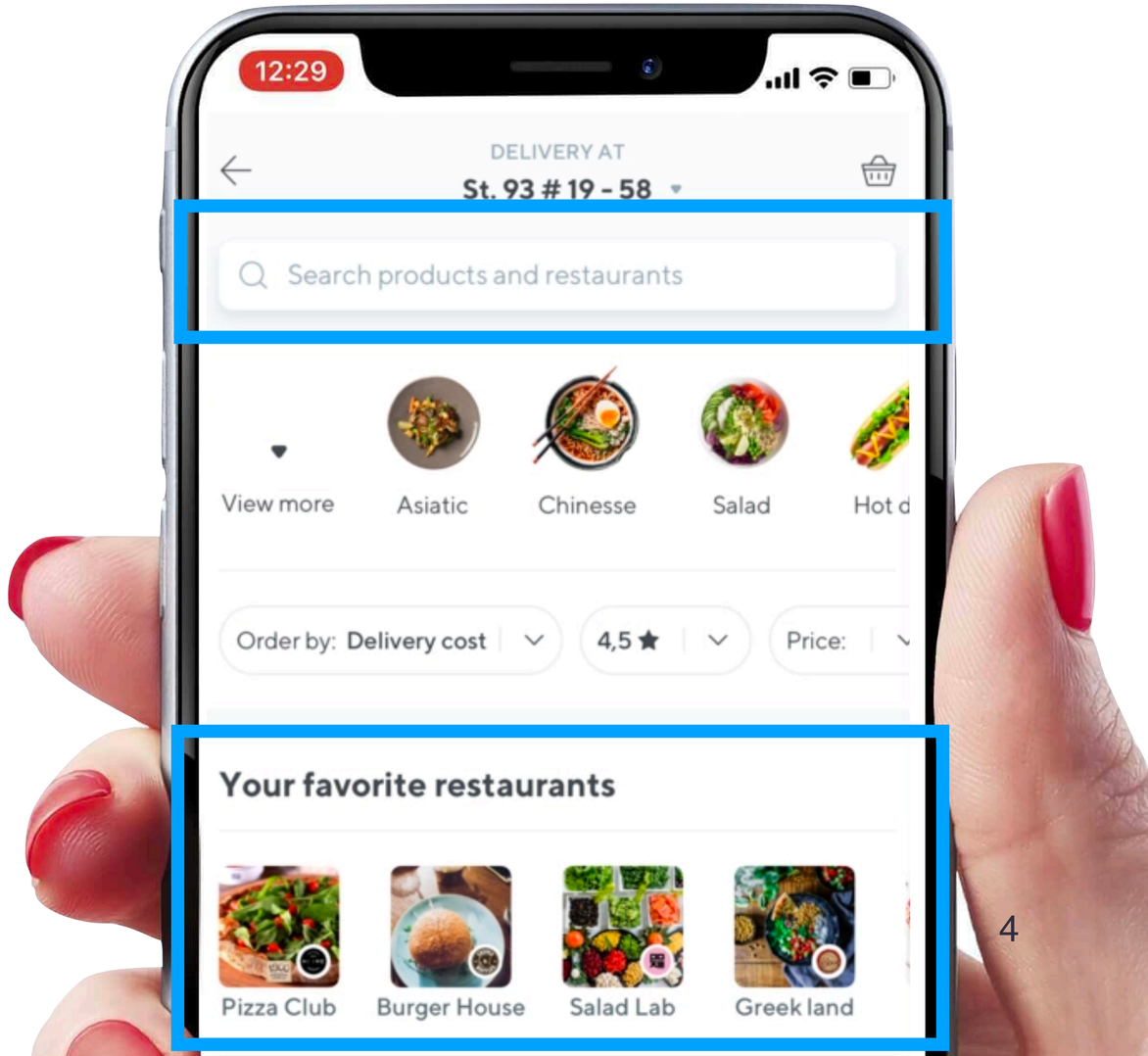
What is a Product?

Rappi
ASTRONOMER

- Brings value to customers
- Those customers pay companies for those products
- Those companies send engineers to Pydata Cordoba to build better products

What is a Data Product?

Rappi
ASTRONOMER

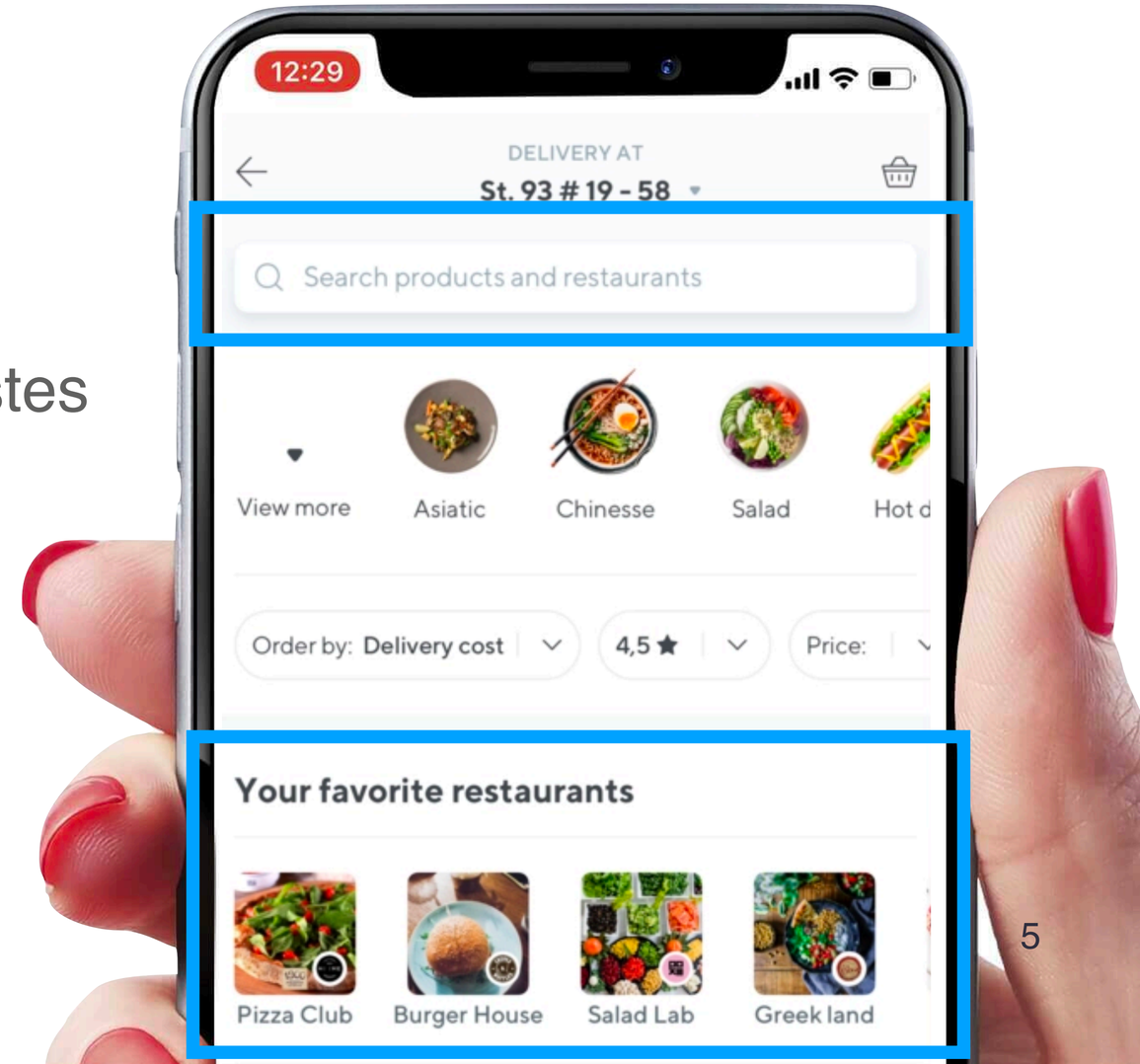


Data Products

Rappi

ASTRONOMER

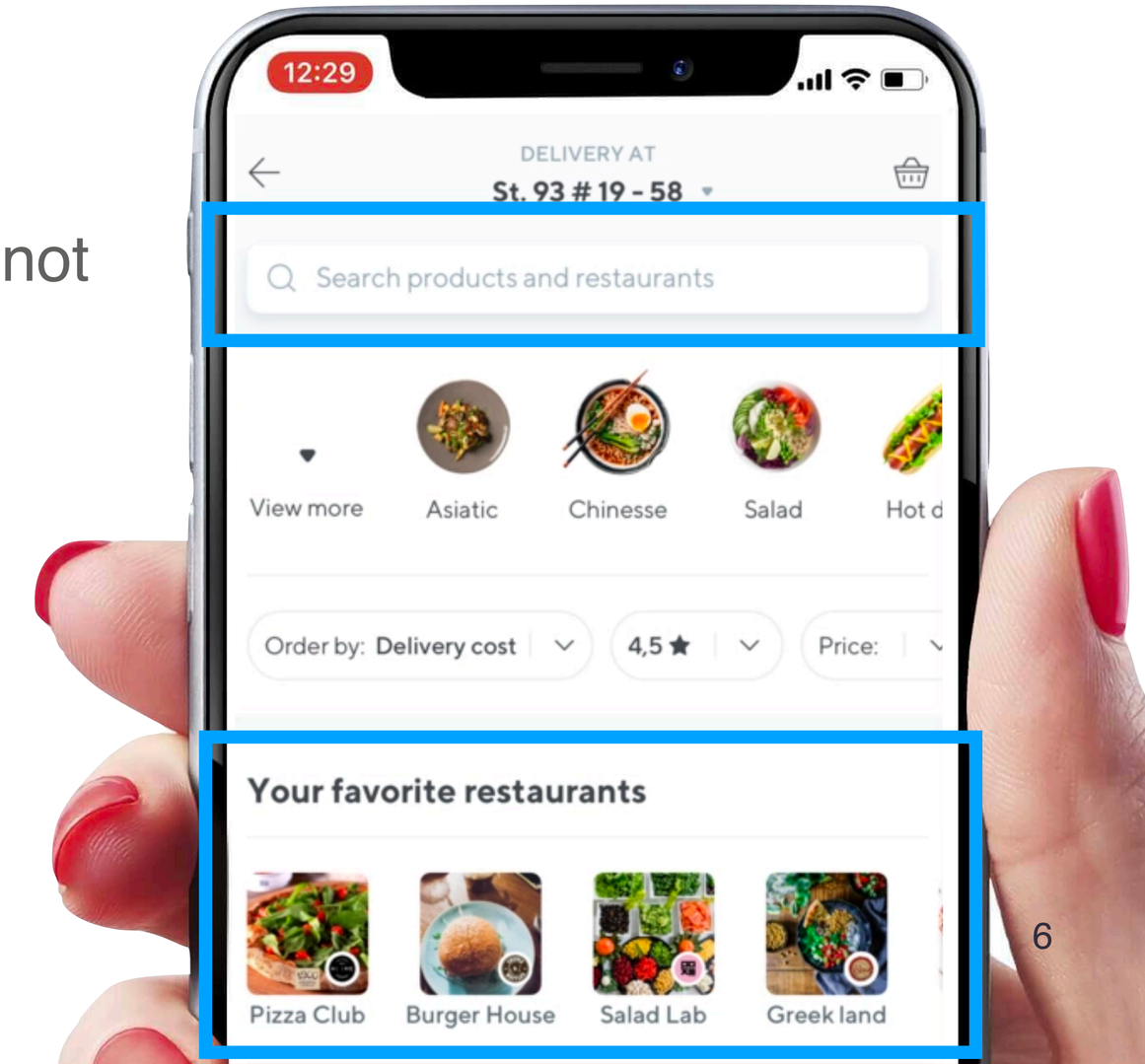
- A product feature that personalizes your experience
- A search bar that remembers your order
- A suggestion tab that can compare your tastes with thousands of similar customers
- An auto-checkout that know what you want after a crappy day



Data Products

Rappi
ASTRONOMER

- Good products AND good data products do not appear out of thin air
- Many start as Minimum Viable Products



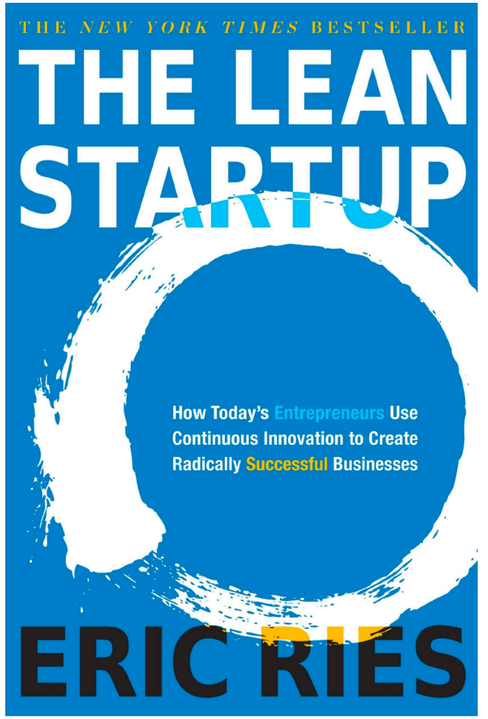
Minimum Viable Product

Rappi

ASTRONOMER

Eric Ries - The Lean Startup

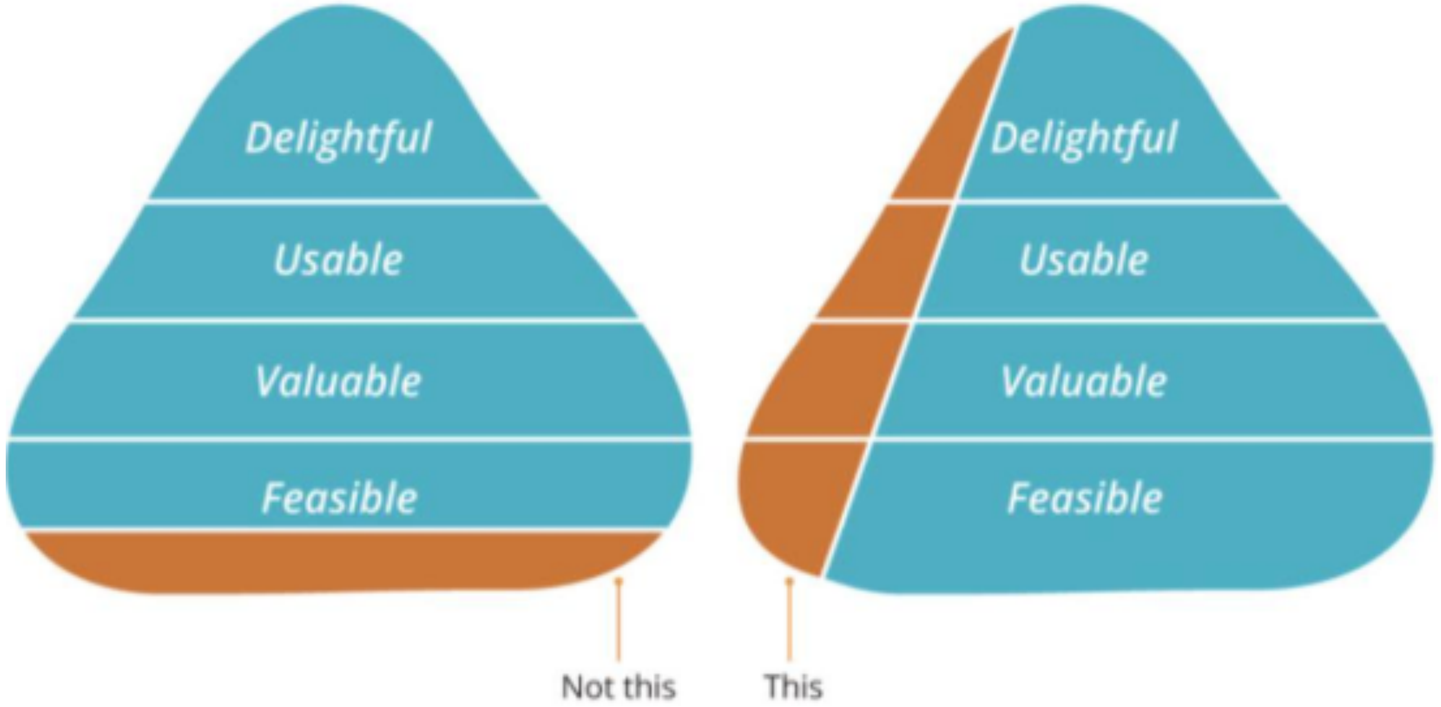
- The minimum feature set to express what the product CAN be
- An easy iteration to incorporate client feedback
- Constant validation of product/market fit at every step



Minimum Viable Data Product

Rappi
ASTRONOMER

- Does this product introduce information the customer cares about?
- Is it doing so with enough accuracy to be valuable?
- Valuable for getting internal buy-in from management

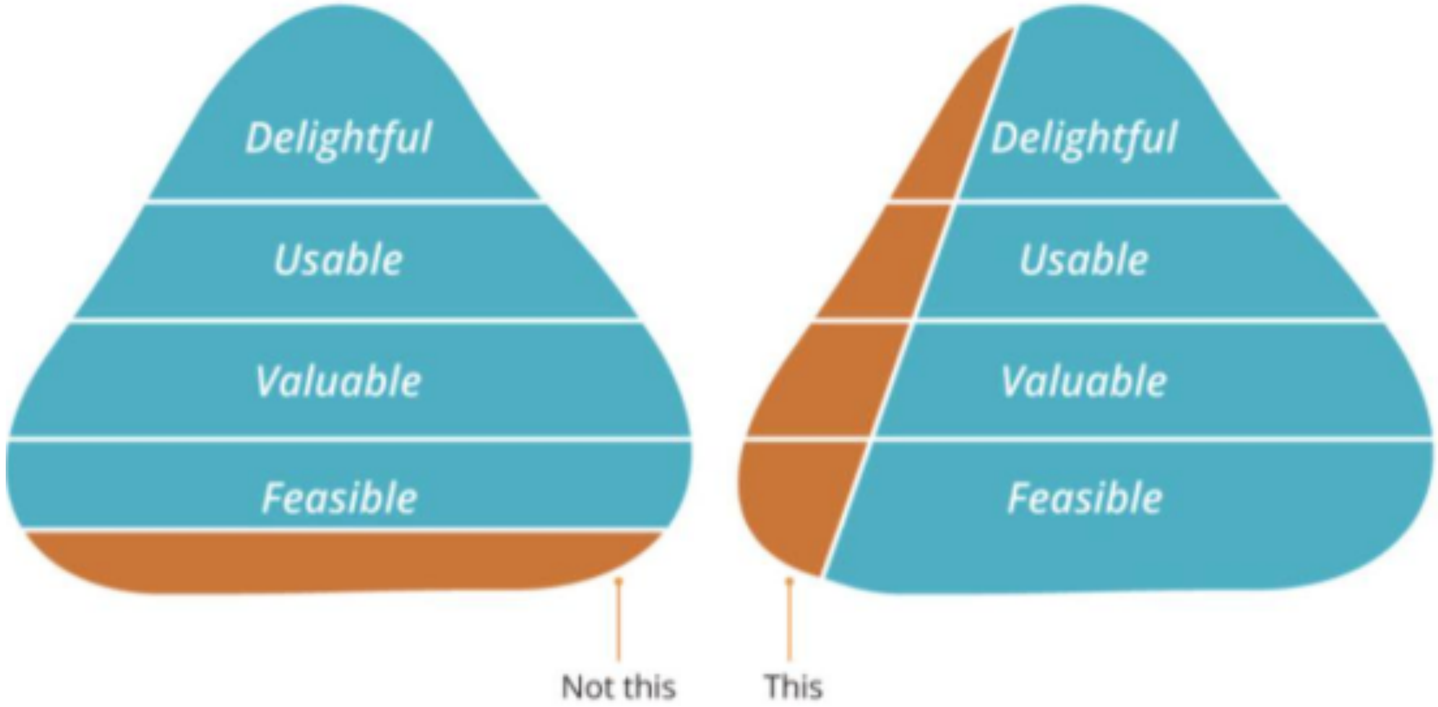


credits @boagworld

Minimum Viable Data Product

Rappi
ASTRONOMER

- With both MVPs and MVDPs there is a single factor that is consistent for success



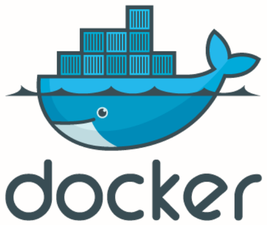
credits @boagworld

Rappi

ASTRONOMER

The Most Important Part of an MVP is The Ability to Iterate Quickly

From 0 to Data Science Pipeline



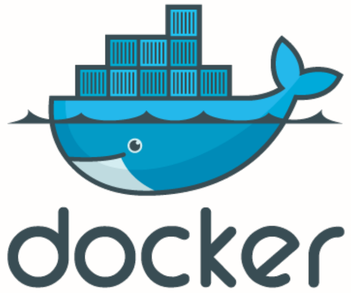
Rappi

ASTRONOMER

https://github.com/gonzalodiaz/pydata2019cba-mvdp-airflow-jupyter

Rappi

ASTRONOMER



Rappi

ASTRONOMER

? ? ?

Rappi

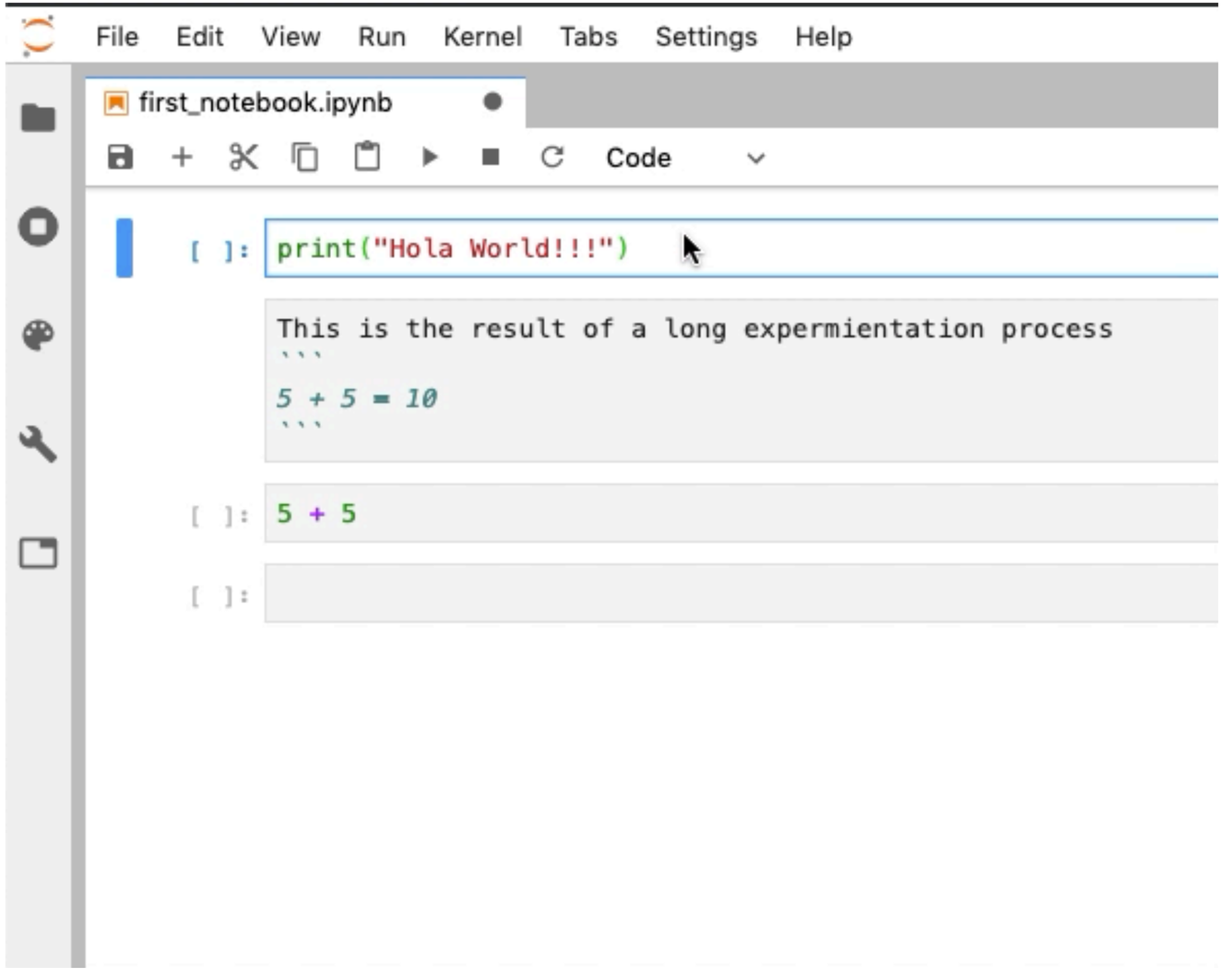
ASTRONOMER

Jupyter Notebooks

Watch 308 | Star 6,282 | Fork 2,631

Rappi
ASTRONOMER

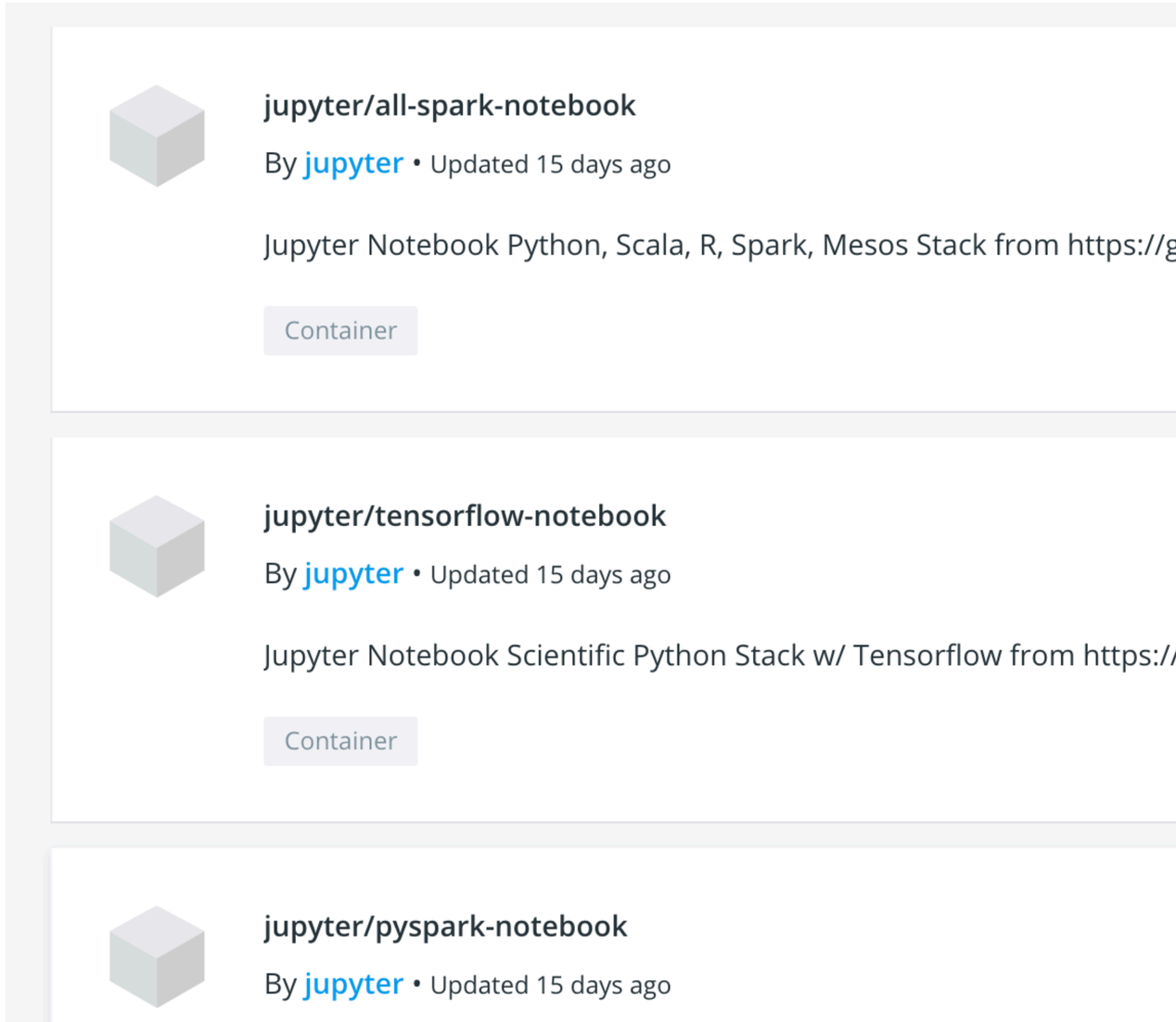
- Notebooks are a great way to tell stories
- Jupyter Notebooks are interactive Python/R/Julia/JS/* interpreters
- Enables iterative experimentation with data
- Pandas, Scikit-learn, Numpy
- Spark, Tensorflow



Jupyter Docker Stacks

Rappi
ASTRONOMER

- Easy Notebook setup
- Many base Jupyter images with popular data science libraries pre-built
- E.g.: tensor flow, pyspark, scikitlearn
- https://hub.docker.com/u/jupyter

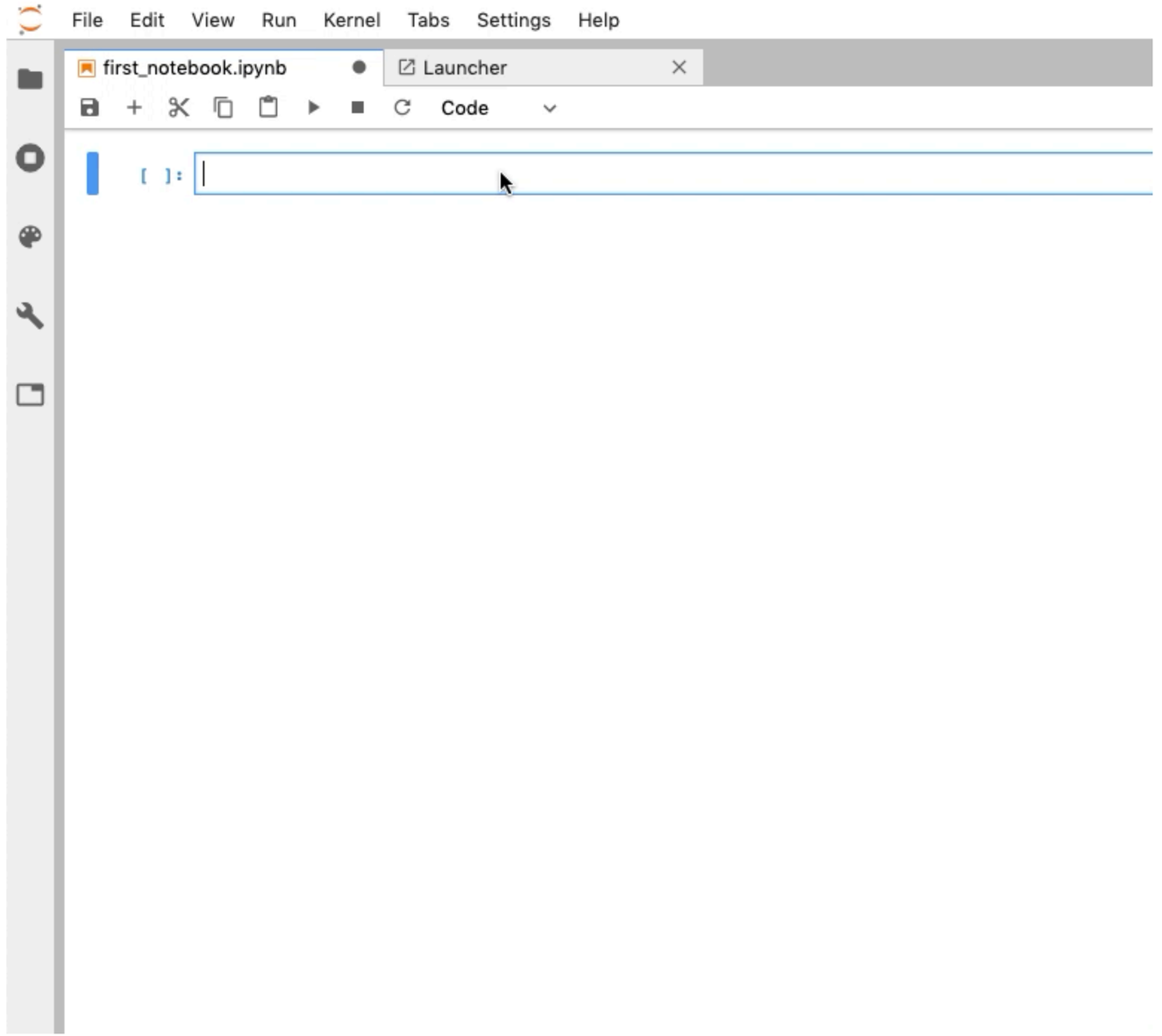


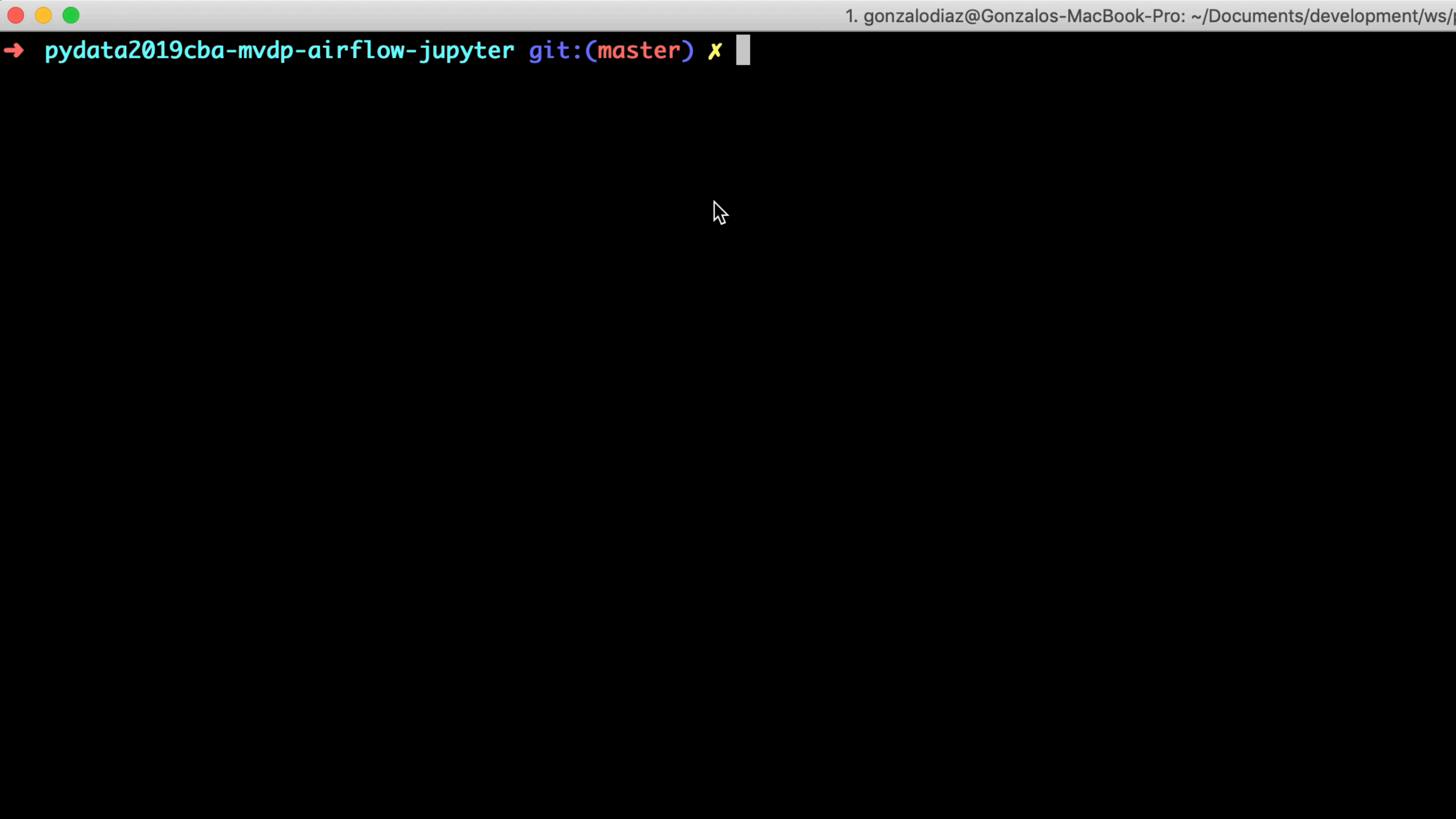
JupyterLab

Watch 347 | Star 8,777 | Fork 1,338

Rappi
ASTRONOMER

- Great integrated development environment for Jupyter Notebooks.
- More than just Notebooks
- Will replace Jupyter Notebooks soon




1. gonzalodiaz@Gonzalos-MacBook-Pro: ~/Documents/development/ws/p
➜ pydata2019cba-mvdp-airflow-jupyter git:(master) ✗

JupyterLab

localhost:8888/lab?

File Edit View Run Kernel Tabs Settings Help

/ templates / nb_originals /

Name	Last Modified
KaggleTitanic.ipynb	a minute ago

Untitled.ipynb | KaggleTitanic.ipynb

Code | Python 3

Parameters

```
[ ]: n_estimators = 50
     datasets_root_path = '/home/jovyan/work/jupyter/templates/datasets'
```

Import Libraries

```
[ ]: # Ignore warnings
     import warnings
     warnings.filterwarnings('ignore')

     # Handle table-like data and matrices
     import numpy as np
     import pandas as pd

     # Modelling Algorithms
     from sklearn.ensemble import RandomForestClassifier

     # Modelling Helpers
     from sklearn.model_selection import train_test_split
```

Data Loading

```
[ ]: train = pd.read_csv(f'{datasets_root_path}/titanic/train.csv')
     test = pd.read_csv(f'{datasets_root_path}/titanic/test.csv')

     full = train.append( test , ignore_index = True, sort = False )
     titanic = full[ :891 ]

     del train , test

     print ('Datasets:' , 'full:' , full.shape , 'titanic:' , titanic.shape)
```

Data Preparations

```
[ ]: ## Data Preparation
     sex = pd.Series( np.where( full.Sex == 'male' , 1 , 0 ) , name = 'Sex' )
     embarked = pd.get_dummies( full.Embarked , prefix='Embarked' )
```

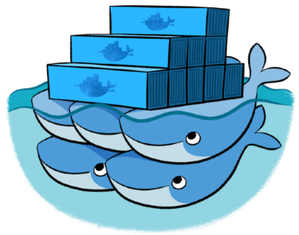

0 2 Python 3 | Idle

Mode: Edit Ln 2, Col 11 KaggleTitanic.ipynb

Distributed Jupyter Notebooks Computing

Rappi
ASTRONOMER

Enterprise Gateway



JupyterHub

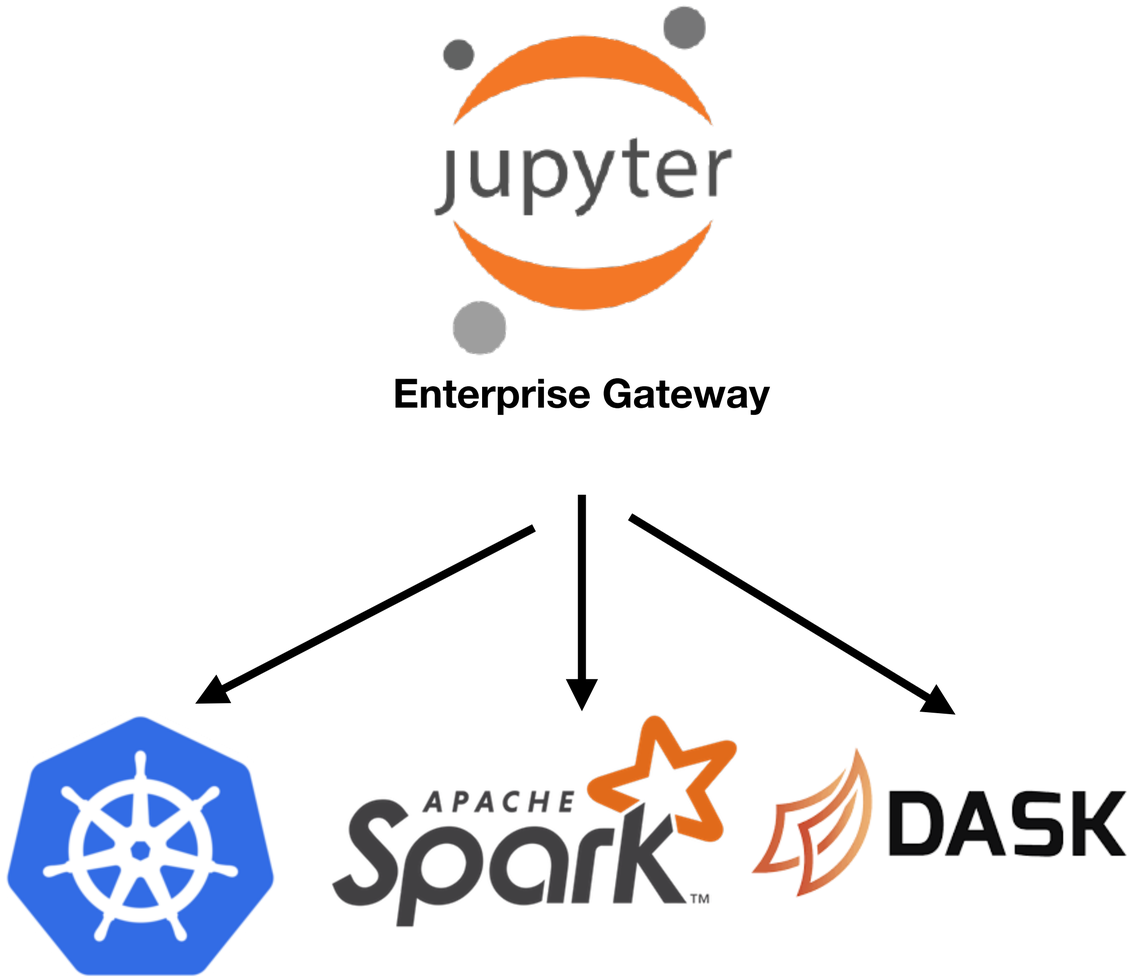
Rappi
ASTRONOMER

- Remote running of Jupyter Notebooks
- Can run Multi-tenant on a single instance
- Offers solution for Kubernetes

Jupyter Enterprise Gateway

Rappi
ASTRONOMER

- Runs on YARN, Kubernetes, etc.
- Easy Management of multi-tenant environments
- Integrates natively with Spark, Dask, and other parallel libraries



BinderHub

Rappi
ASTRONOMER

- Git integration for easy launching and sharing of Jupyter Notebooks
- Not for enterprise: Notebooks must be public

Papermill

Watch 84 | Star 2,457 | Fork 198

Rappi
ASTRONOMER

- Parameterize Notebooks through cell tagging
- Execute using Python API or CLI
- Store to S3, Azure

File Edit View Run Kernel Tabs Settings Help

/ templates / papermill_tutorial_basic /

Name	Last Modified
input.ipynb	seconds ago
papermill_base.ipynb	seconds ago

input.ipynb | papermill_base.ipynb

Code

Set cell metadata!

```
{
    "tags": [
        "parameters"
    ]
}
```

[ ]:
```
#parameters
a = 'a'
b = 'b'
```

[ ]:
```
print(f'a = {a}')
```

[ ]:
```
print(f'b = {b}')
```

[ ]:

/lab?

el Tabs Settings Help

Last Modified

a minute ago

Untitled.ipynb | KaggleTitanic.ipynb

Code | Python 3

Parameters

```
[ ]: n_estimators = 50
     datasets_root_path = '/home/jovyan/work/jupyter/templates/datasets'
```

Import Libraries

```
[ ]: # Ignore warnings
     import warnings
     warnings.filterwarnings('ignore')

     # Handle table-like data and matrices
     import numpy as np
     import pandas as pd

     # Modelling Algorithms
     from sklearn.ensemble import RandomForestClassifier

     # Modelling Helpers
     from sklearn.model_selection import train_test_split
```

Data Loading

```
[ ]: train = pd.read_csv(f'{datasets_root_path}/titanic/train.csv')
     test = pd.read_csv(f'{datasets_root_path}/titanic/test.csv')

     full = train.append( test , ignore_index = True, sort = False )
     titanic = full[ :891 ]
```

Scaling Jupyter Notebooks

Rappi

ASTRONOMER

- Have the algorithm
- Can parameterize notebook model builds using paper mill
- But how do we productionize?

Apache Airflow

Watch 663 | Star 13,982 | Fork 5,202

Rappi

ASTRONOMER

Apache Airflow

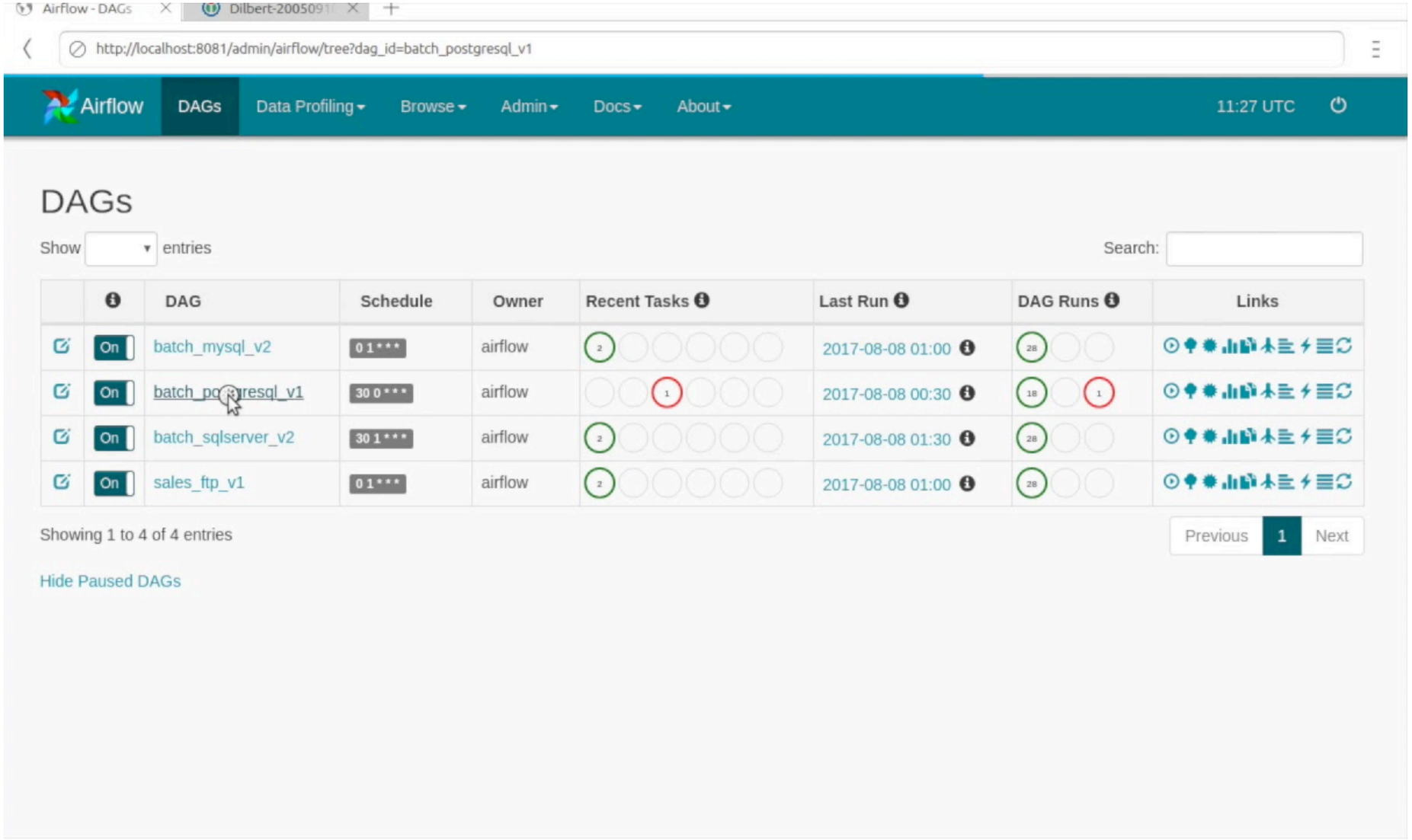
Rappi

ASTRONOMER

- Workflow scheduler with python DAGs
- Easy to schedule, scale, and monitor pipelines

Apache Airflow

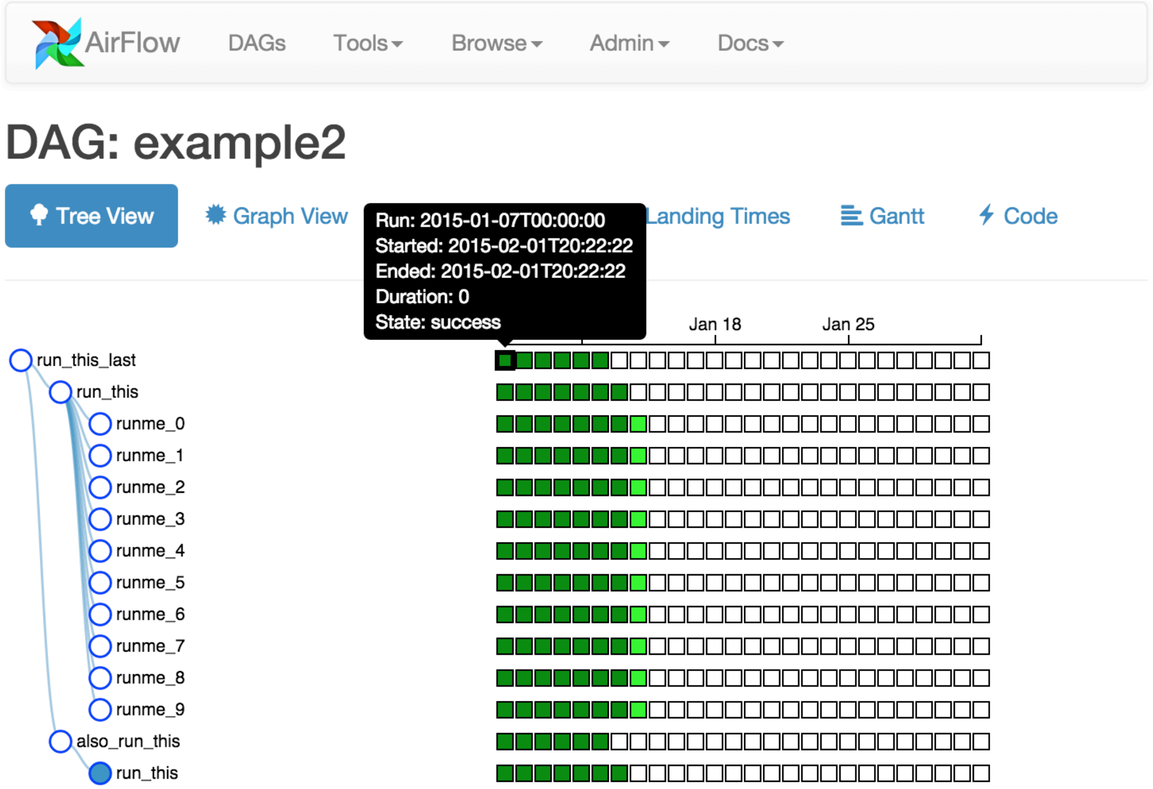
Rappi
ASTRONOMER

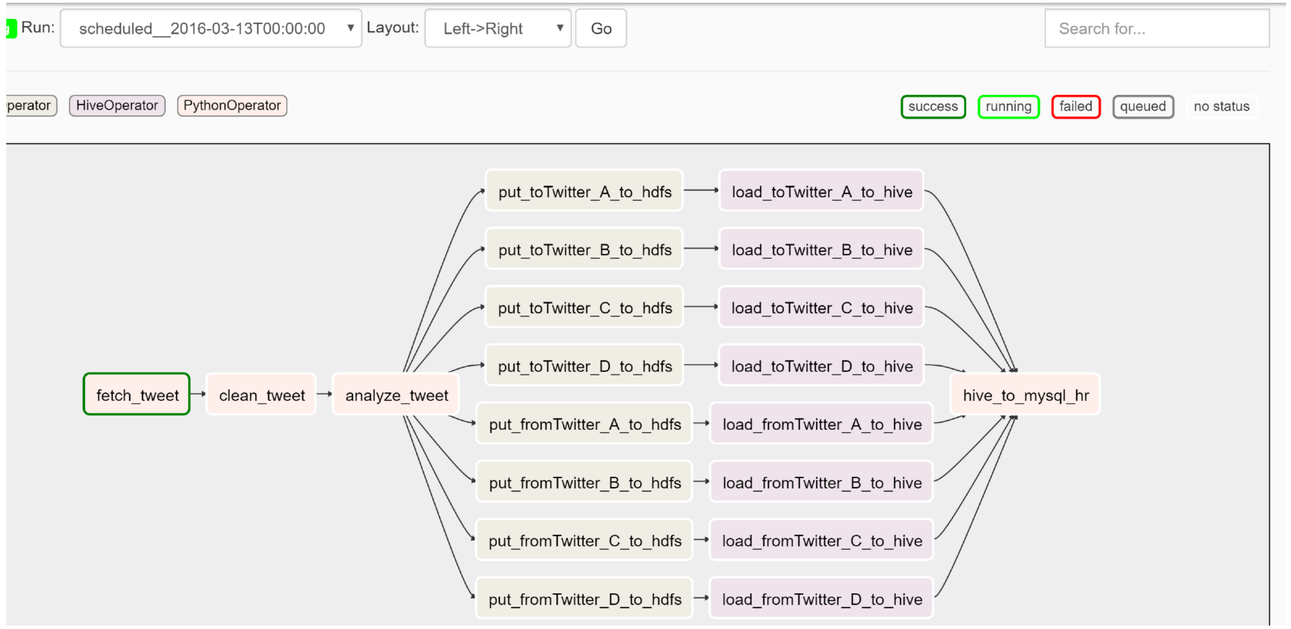


Apache Airflow

Rappi

ASTRONOMER





Apache Airflow

Rappi
ASTRONOMER

```
dag = DAG('tutorial', default_args=default_args)

# t1, t2 and t3 are examples of tasks created by instantiating operators
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)

templated_command = """
    {% for i in range(5) %}
        echo "{{ ds }}"
        echo "{{ macros.ds_add(ds, 7)}}"
        echo "{{ params.my_param }}"
    {% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)

t2.set_upstream(t1)
t3.set_upstream(t1)
```

Define Operators

Set Dependencies

Data Science in Airflow

Rappi

ASTRONOMER

- Hyper-parameter Tuning
- Canary Testing
- A/B Testing
- Model Deployment

Hyper-parameter tuning

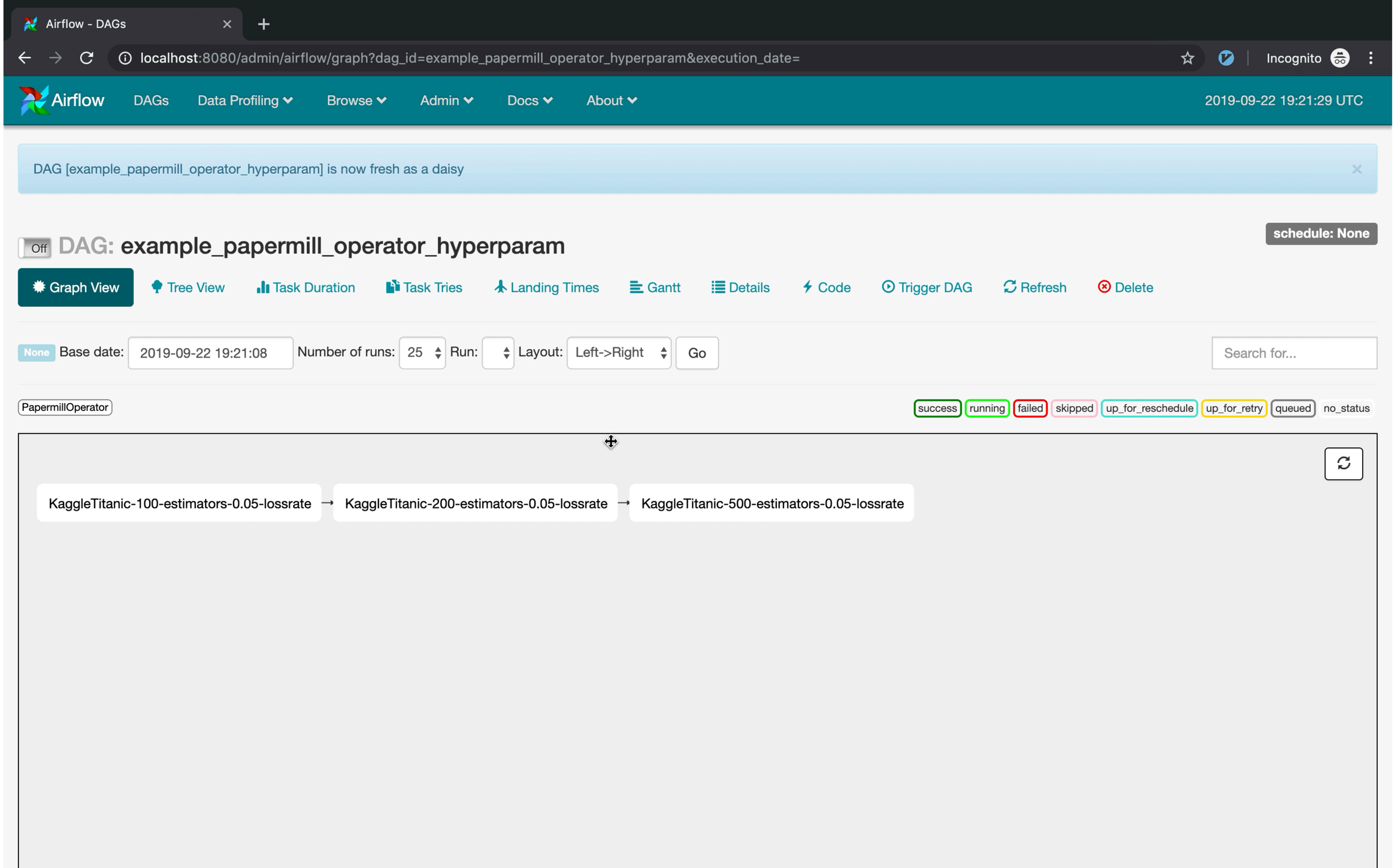
Rappi
ASTRONOMER

- Choosing a set of optimal hyperparameters for a learning algorithm.
- E.g.: Loss-rate, tolerance, etc.

Grid-search

Rappi
ASTRONOMER

- Essentially a for-loop
- Pythonic DAG creation shines here
- Can nest for-loops for multiple degrees


Airflow - DAGs
localhost:8080/admin/airflow/graph?dag_id=example_papermill_operator_hyperparam&execution_date=
Incognito
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
2019-09-22 19:21:29 UTC
DAG [example_papermill_operator_hyperparam] is now fresh as a daisy
Off
DAG: example_papermill_operator_hyperparam
schedule: None
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Trigger DAG
Refresh
Delete
None
Base date: 2019-09-22 19:21:08
Number of runs: 25
Run:
Layout: Left->Right
Go
Search for...
PapermillOperator
success
running
failed
skipped
up_for_reschedule
up_for_retry
queued
no_status
KaggleTitanic-100-estimators-0.05-lossrate
KaggleTitanic-200-estimators-0.05-lossrate
KaggleTitanic-500-estimators-0.05-lossrate

Airflow - DAGs

localhost:8080/admin/airflow/graph?dag_id=example_papermill_operator_hyperparam&execution_date=

Incognito

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | About | 2019-09-22 19:22:36 UTC

DAG [example_papermill_operator_hyperparam] is now fresh as a daisy

Off **DAG: example_papermill_operator_hyperparam**

schedule: None

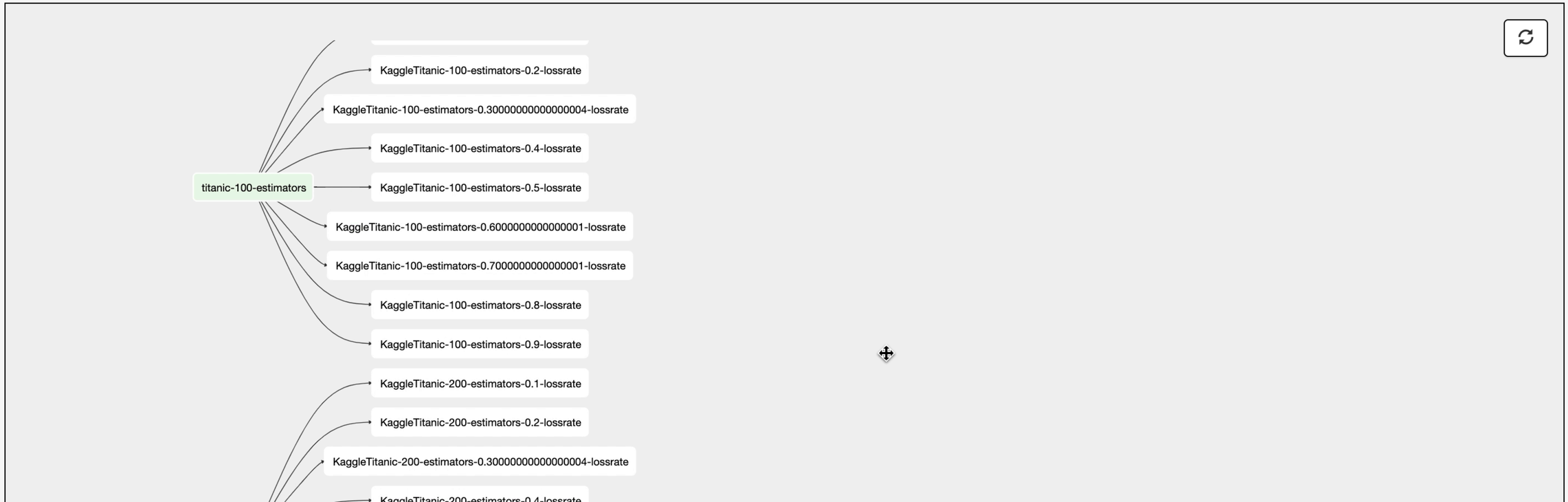
Graph View | Tree View | Task Duration | Task Tries | Landing Times | Gantt | Details | Code | Trigger DAG | Refresh | Delete

None Base date: 2019-09-22 19:22:29 Number of runs: 25 Run: Layout: Left->Right Go

Search for...

DummyOperator PapermillOperator

success running failed skipped up_for_reschedule up_for_retry queued no_status



Grid-search

Rappi
ASTRONOMER

- Essentially a for-loop
- Pythonic DAG creation shines here
- Can nest for-loops for multiple degrees
- However not super efficient…

```
num_estimators = [100,200,500]
loss_rates = [.01, .04, .07]
with DAG(
    dag_id='example_papermill_operator', default_args=args,
    schedule_interval=None
) as dag:
    start = DummyOperator()
    dag >> start
    last = start
    for num_estimator in num_estimators:
        merge = DummyOperator()
        merge << last
        for loss_rate in loss_rates:
            task_name = 'KaggleTitanic-{}-{}-estimators'.format(num_estimator, loss_rate)
            output_name = "/nb_outputs/{}.ipynb".format(task_name)
            titanic_estimator = PapermillOperator(
                task_id=task_name,
                input_nb="/nb_originals/KaggleTitanic.ipynb",
                output_nb=,
                parameters={"n_estimators": num_estimator, "loss_rate": loss_rate}
            )
            merge << titanic_estimator
```

Optimized Grid-search

Rappi
ASTRONOMER

- Set a “maximum” fit
- Maintain a “source of truth”

```
[15]: if get_current_max() > max_fit
          model.fit( train_X , train_y )
```

Optimized Grid-search

Rappi
ASTRONOMER

Stochastic Grid Search

Rappi
ASTRONOMER

- Use ShortCircuitOperators to complete loop early if optimal answer found
- Use XCom to communicate the results of the previous round to the next round

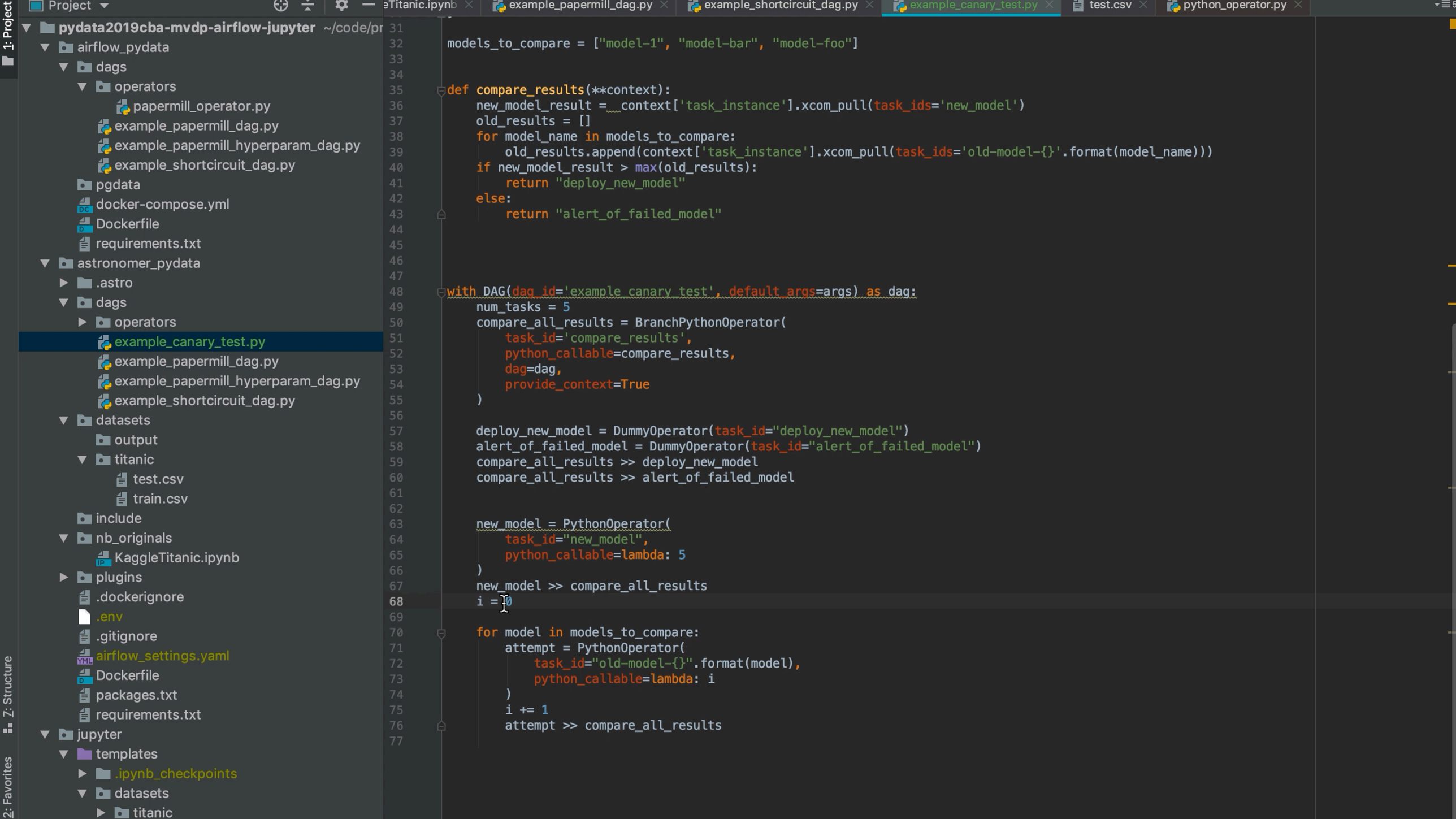
```
num_estimators = [100,200,500]
loss_rates = [.01, .04, .07]
with DAG(
    dag_id='example_papermill_operator', default_args=args,
    schedule_interval=None
) as dag:
    start = DummyOperator()
    dag >> start
    last = start
    for num_estimator in num_estimators:
        merge = DummyOperator()
        merge << last
        for loss_rate in loss_rates:
            task_name = 'KaggleTitanic-{}-{}-estimators'.format(num_estimator, loss_rate)
            output_name = "/nb_outputs/{}.ipynb".format(task_name)
            titanic_estimator = PapermillOperator(
                task_id=task_name,
                input_nb="/nb_originals/KaggleTitanic.ipynb",
                output_nb=,
                parameters={"n_estimators": num_estimator, "loss_rate": loss_rate}
            )
            merge << titanic_estimator
```

Canary Testing

Rappi
ASTRONOMER

- Method for ensuring that your new models are an improvement compared to old ones
- Can also be use to detect model errors (e.g. model isn't a huge deviation from previous models

```python
def compare_results(**context):
    new_model_result = context['task_instance'].xcom_pull(task_ids='new_model')
    old_results = []
    for model_name in models_to_compare:
        old_results.append(context['task_instance'].xcom_pull(task_ids='old-model-{}'.format(model_name)))
    if new_model_result > max(old_results):
        return "deploy_new_model"
    else:
        return "alert_of_failed_model"


with DAG(dag_id='example_canary_test', default_args=args) as dag:
    num_tasks = 5
    compare_all_results = BranchPythonOperator(
        task_id='compare_results',
        python_callable=compare_results,
        dag=dag,
        provide_context=True
    )

    deploy_new_model = DummyOperator(task_id="deploy_new_model")
    alert_of_failed_model = DummyOperator(task_id="alert_of_failed_model")
    compare_all_results >> deploy_new_model
    compare_all_results >> alert_of_failed_model

    new_model = PythonOperator(
        task_id="new_model",
        python_callable=lambda: 5
    )
    new_model >> compare_all_results
    i = 5

    for model in models_to_compare:
        attempt = PythonOperator(
            task_id="old-model-{}".format(model),
            python_callable=lambda: i
        )
        i += 1
        attempt >> compare_all_results
```


Project
pydata2019cba-mvdp-airflow-jupyter ~/code/pr
airflow_pydata
dags
operators
papermill_operator.py
example_papermill_dag.py
example_papermill_hyperparam_dag.py
example_shortcircuit_dag.py
pgdata
docker-compose.yml
Dockerfile
requirements.txt
astronomer_pydata
.astro
dags
operators
example_canary_test.py
example_papermill_dag.py
example_papermill_hyperparam_dag.py
example_shortcircuit_dag.py
datasets
output
titanic
test.csv
train.csv
include
nb_originals
KaggleTitanic.ipynb
plugins
.dockerignore
.env
.gitignore
airflow_settings.yaml
Dockerfile
packages.txt
requirements.txt
jupyter
templates
.ipynb_checkpoints
datasets
titanic
eTitanic.ipynb
example_papermill_dag.py
example_shortcircuit_dag.py
example_canary_test.py
test.csv
python_operator.py
models_to_compare = ["model-1", "model-bar", "model-foo"]
def compare_results(**context):
new_model_result = context['task_instance'].xcom_pull(task_ids='new_model')
old_results = []
for model_name in models_to_compare:
old_results.append(context['task_instance'].xcom_pull(task_ids='old-model-{}'.format(model_name)))
if new_model_result > max(old_results):
return "deploy_new_model"
else:
return "alert_of_failed_model"
with DAG(dag_id='example_canary_test', default_args=args) as dag:
num_tasks = 5
compare_all_results = BranchPythonOperator(
task_id='compare_results',
python_callable=compare_results,
dag=dag,
provide_context=True
)
deploy_new_model = DummyOperator(task_id="deploy_new_model")
alert_of_failed_model = DummyOperator(task_id="alert_of_failed_model")
compare_all_results >> deploy_new_model
compare_all_results >> alert_of_failed_model
new_model = PythonOperator(
task_id="new_model",
python_callable=lambda: 5
)
new_model >> compare_all_results
i = 0
for model in models_to_compare:
attempt = PythonOperator(
task_id="old-model-{}".format(model),
python_callable=lambda: i
)
i += 1
attempt >> compare_all_results
1: Project
7: Structure
2: Favorites

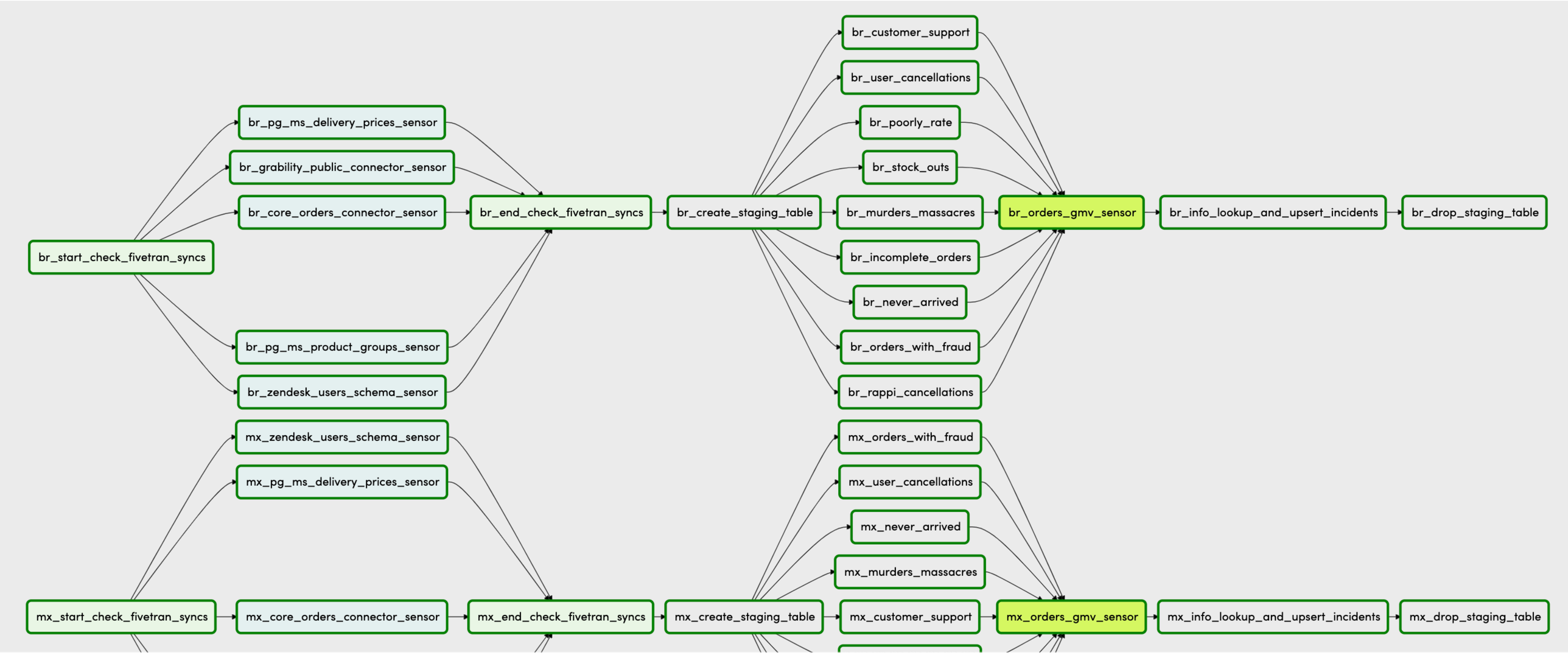
Airflow at Rappi

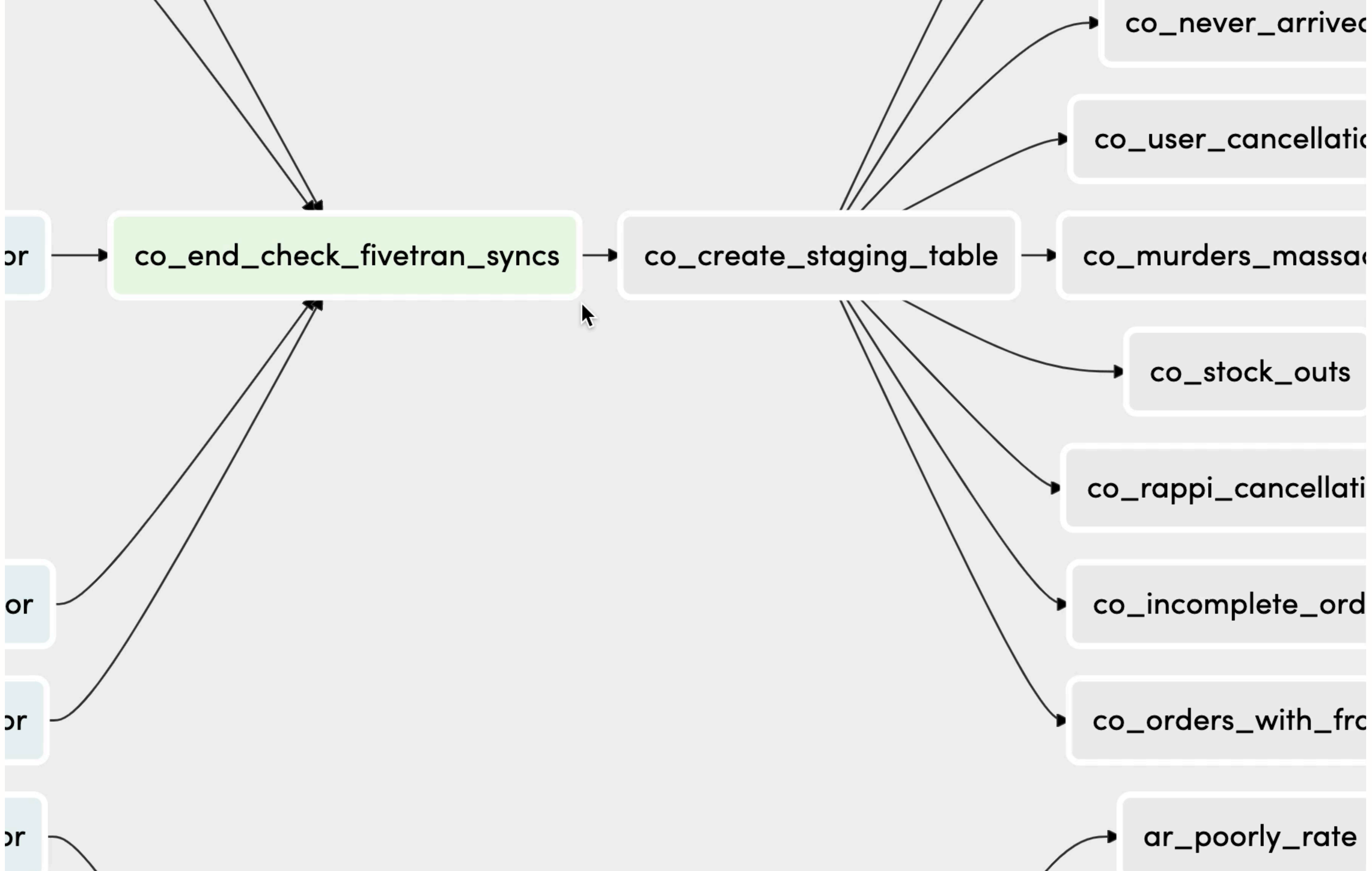
Rappi

ASTRONOMER

- Take advantage of Airflow extensibility
- RappiFlow: Custom plugins + Astronomer
- SQL + Pandas + Programatic Pipelines to generalize the DAGs
- Docker Images for Training and Offline Prediction

Airflow at Rappi




co_never_arrived
co_user_cancellatio
or
co_end_check_fivetran_syncs
co_create_staging_table
co_murders_massa
co_stock_outs
co_rappi_cancellati
or
co_incomplete_ord
or
co_orders_with_fro
or
ar_poorly_rate

```python
class TrainAndPredictCatalogOperator(KubernetesPodOperator):
    """Train and Predict Recommendation using the Docker Image."""

    @apply_defaults
    def __init__(self, training_params: dict, image_name: str, image_tag_name: str, image_pull_secret: str,
                 k8s_namespace: str, *args, **kwargs):
        """Class Initializer."""
        namespace = k8s_namespace
        image = f"{image_name}:{image_tag_name}"
        name = f"{training_params['country_code']}-ml-train-cpgs-catalog"
        super().__init__(
            namespace=namespace, image=image, image_pull_secrets=image_pull_secret, name=name, *args, **kwa
        )
        self.get_logs = True
        self.in_cluster = True
        self.is_delete_operator_pod = True
        self.env_vars = {
            'COUNTRY': training_params['country_code'],
            'DATE': '{{ ds_nodash }}',
            'AWS_BUCKET': settings.CPGS_DATA_SCIENCE_S3_BUCKET,
        }

        self.resources = Resources(
            request_memory=get_k8s_resources_mapping(training_params['node_size'])['memory'],
            limit_memory=get_k8s_resources_mapping(training_params['node_size'])['memory'],
            request_cpu=get_k8s_resources_mapping(training_params['node_size'])['cpu'],
            limit_cpu=get_k8s_resources_mapping(training_params['node_size'])['cpu']
        )

        self.pool = training_params['pool']

        sf_secrets = Secret(
            deploy_type='env',
            deploy_target=settings.SNOWFLAKE_CPGS_DS_WRITE_CONN_ID,
            secret=settings.SNOWFLAKE_CPGS_DS_WRITE_CONN_ID.replace('_', '-')
        )

        s3_secrets = Secret(
            deploy_type='env',
            deploy_target=settings.S3_CPGS_DS_CONN_ID,
            secret=settings.S3_CPGS_DS_CONN_ID.replace('_', '-')
        )
        self.secrets = [sf_secrets, s3_secrets]
```

Rappi

ASTRONOMER

Scaling Apache Airflow

Rappi

ASTRONOMER

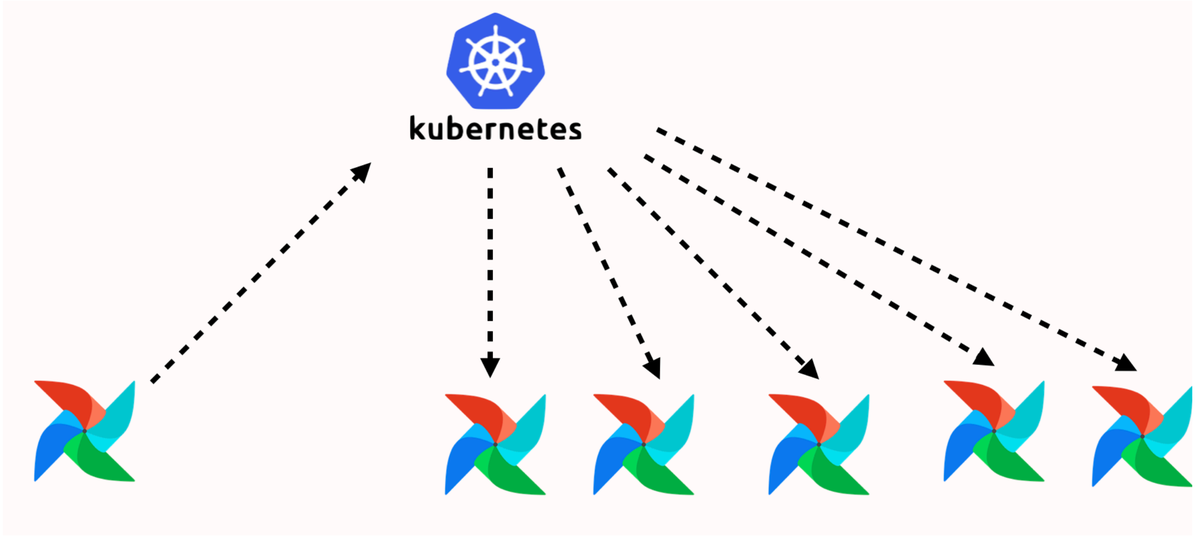
KubernetesExecutor

CeleryExecutor

KubernetesExecutor

Rappi
ASTRONOMER

- Dynamic Allocation
- Fault Tolerance
- Only Requires Kubernetes



CeleryExecutor

Rappi

ASTRONOMER

- Immediate SLAs
- Multiple tasks per-worker
- Queues for differing requirements

ExecutorConfig

Rappi
ASTRONOMER

- Users can define elements of tasks in the task description
- Set RAM/CPU requests per-task
- service-accounts for PII
- Node affinity to a GPU-machine

```
t = BashOperator(
  task_id = ‘account-test’,
  bash_command = ‘gcloud auth application-default login’,
  dag = dag,

  executor_config = {
  ‘request_memory’: ‘128Mi’,
  ‘limit_memory’: ‘128Mi’
  ‘image’: ‘airflow/scipy:1.1.5’
   ‘gcp-service-account’ : ‘service-account@xxx.iam.gserviceaccount.com’
   }

)
```

Deploying Apache Airflow

Rappi
ASTRONOMER

KubernetesExecutor

CeleryExecutor

Deploying Apache Airflow

Rappi

ASTRONOMER

kubernetes

KubernetesExecutor

CeleryExecutor

Astronomer

Rappi

ASTRONOMER

- Disclosure: I work for Astronomer
- Apache Airflow as a Service on Kubernetes
- Easy deployment of K8sExecutor and CeleryExecutor
- Monitoring, support,

Astronomer Local Development

Rappi
ASTRONOMER

- CLI allows for local airflow server without signing up
- curl -sSL https://install.astronomer.io l sudo bash

```
dimberman astronomer (⎈|context|default) $
```

```
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $ mkdir test
dimberman astronomer (⎈|context|default) $ cd test/
dimberman test (⎈|context|default) $
```

```
dimberman astronomer (⎈|context|default) $
```

```
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $
dimberman astronomer (⎈|context|default) $ mkdir test
dimberman astronomer (⎈|context|default) $ cd test/
dimberman test (⎈|context|default) $ astro dev init
Initialized empty astronomer project in /Users/dimberman/code/astronomer/test
dimberman test (⎈|context|default) $ ls
Dockerfile              airflow_settings.yaml   dags                    include                 packages.txt            plugins
dimberman test (⎈|context|default) $ vi dags/example-dag.py
dimberman test (⎈|context|default) $
```

```
mberman test (⎈ |context|default) $ ls
ockerfile             airflow_settings.yaml   dags                  include               packages.txt          plugins               requirements.txt
mberman test (⎈ |context|default) $ vi dags/example-dag.py
mberman test (⎈ |context|default) $ astro dev start
nv file ".env" found. Loading...
ending build context to Docker daemon  10.75kB
tep 1/1 : FROM astronomerinc/ap-airflow:0.10.1-alpha.14-1.10.5-onbuild
 10.1-alpha.14-1.10.5-onbuild: Pulling from astronomerinc/ap-airflow
d48c3bd43c5: Already exists
7a4c1370ba7: Pull complete
06a6e31dc93: Pull complete
16afb80ad72: Pull complete
13fd51898e3: Pull complete
3254078e654: Pull complete
c3c2ca9ebe7: Pull complete
76233955f1d: Pull complete
igest: sha256:5358698d4c0a0644c3fd5ba022f7aaa7793461cfeca49177639fdaf84f184b3e
tatus: Downloaded newer image for astronomerinc/ap-airflow:0.10.1-alpha.14-1.10.5-onbuild
 Executing 5 build triggers
---> Running in c3d8dbad470d
etch http://dl-cdn.alpinelinux.org/alpine/v3.10/main/x86_64/APKINDEX.tar.gz
etch http://dl-cdn.alpinelinux.org/alpine/v3.10/community/x86_64/APKINDEX.tar.gz
etch http://dl-cdn.alpinelinux.org/alpine/edge/community/x86_64/APKINDEX.tar.gz
etch http://dl-cdn.alpinelinux.org/alpine/edge/testing/x86_64/APKINDEX.tar.gz
etch http://dl-cdn.alpinelinux.org/alpine/edge/main/x86_64/APKINDEX.tar.gz
K: 234 MiB in 69 packages
emoving intermediate container c3d8dbad470d
---> Running in 792f3e843298
ARNING: You are using pip version 19.2.1, however version 19.2.3 is available.
ou should consider upgrading via the 'pip install --upgrade pip' command.
emoving intermediate container 792f3e843298
---> 45dc3804cc92
uccessfully built 45dc3804cc92
uccessfully tagged test_bf7430/airflow:latest
reating network "testbf7430_airflow" with driver "bridge"
reating volume "testbf7430_airflow_logs" with driver "local"
reating volume "testbf7430_postgres_data" with driver "local"
NFO[0089] [0/3] [postgres]: Starting
NFO[0090] [1/3] [postgres]: Started
NFO[0090] [1/3] [scheduler]: Starting
NFO[0091] [2/3] [scheduler]: Started
NFO[0091] [2/3] [webserver]: Starting
NFO[0092] [3/3] [webserver]: Started
irflow Webserver: http://localhost:8080
ostgres Database: localhost:5432/postgres
he default credentials are admin:admin
mberman test (⎈ |context|default) $
```

Testing Airflow Development

Rappi

ASTRONOMER

- Once you have a running local instance you can continually change your DAG and compare on WebServer
- Use local docker image to test images for easy iteration
- Treat airflow as an orchestrator and try to keep complex logic in the K8s pod/ SparkJob/Bigquery

Deploying Airflow

Rappi

ASTRONOMER

- Once the python files have been deployed to the airflow machines, Airflow will read the new values and update
- K8sOperator allows users to bake-in files in Docker image, deploy using git, and share files using a persistent-volume
- We recommend baking into docker images

New to Astronomer?

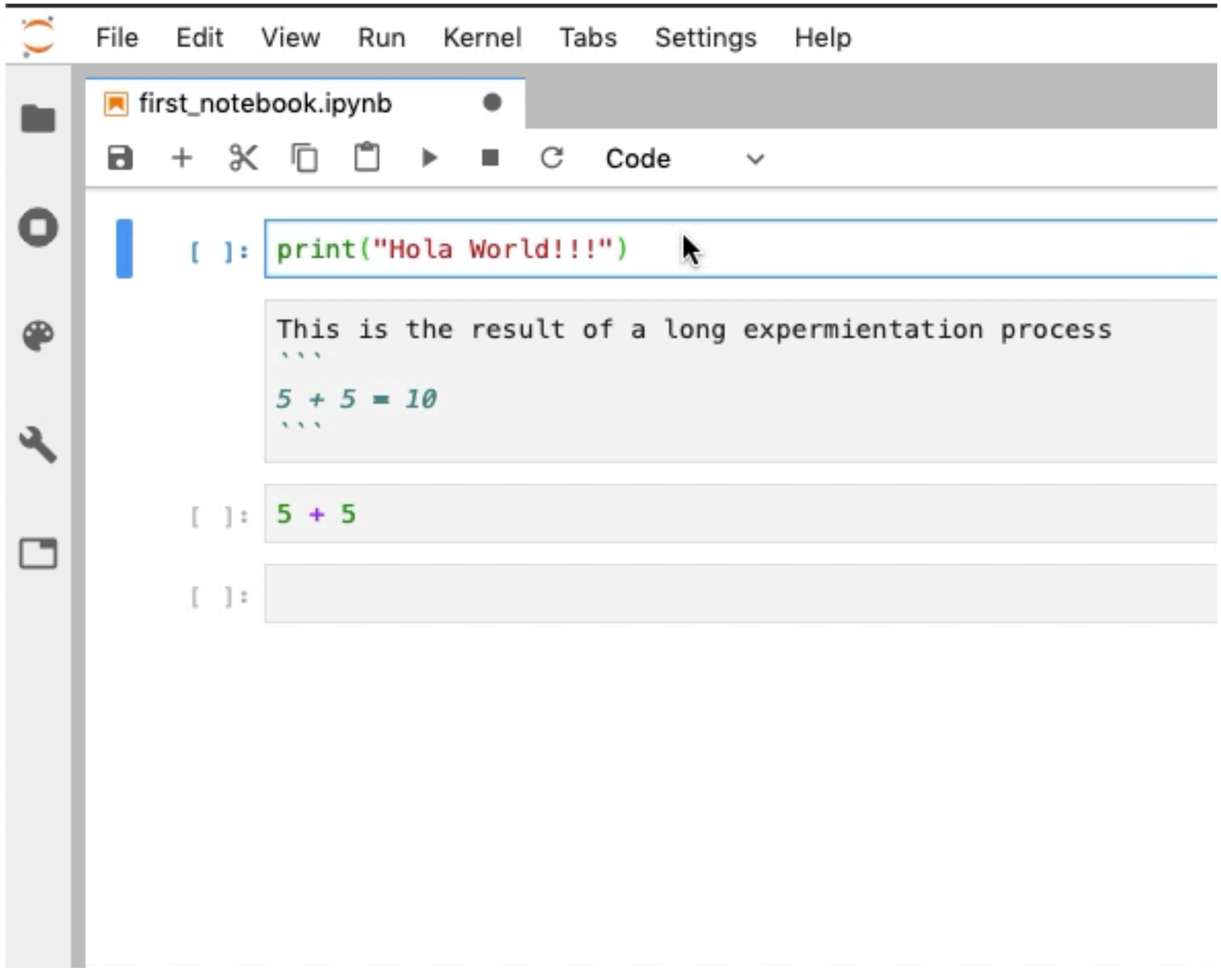
```
Building image...
Sending build context to Docker daemon  147.5kB
Step 1/5 : FROM astronomerinc/ap-airflow:0.10.1-alpha.15-1.10.5-onbuild
# Executing 5 build triggers
 ---> Using cache
 ---> Using cache
 ---> Using cache
 ---> Using cache
 ---> 5a9b7e448924
Step 2/5 : COPY nb_originals /nb_originals
 ---> 9fa83b3d609f
Step 3/5 : COPY datasets /datasets
 ---> 8371b7e26901
Step 4/5 : RUN chmod 777 -R /datasets
 ---> Running in d43360d4d79e
Removing intermediate container d43360d4d79e
 ---> 505a2a7c177c
Step 5/5 : RUN mkdir /nb_outputs && chmod 777 /nb_outputs
 ---> Running in ba4b33852642
Removing intermediate container ba4b33852642
 ---> dc8d45f4eb90
Successfully built dc8d45f4eb90
Successfully tagged exact-spacecraft-1900/airflow:latest
Pushing image to Astronomer registry
The push refers to repository [registry.datarouter.ai/exact-spacecraft-1900/airflow]
c04e606d1092: Pushed
9fe552deb343: Pushed
936003bf8301: Pushed
167af4c9608e: Pushed
3163524aae03: Pushed
403ab3f7e6a8: Layer already exists
96f7b8559712: Layer already exists
ad99845b229c: Layer already exists
19379c7d9566: Layer already exists
2ee04610c8cd: Layer already exists
2465aef6a67e: Layer already exists
d65d405770c8: Layer already exists
58b4b74cd080: Layer already exists
d9729030c916: Layer already exists
63ea16be0911: Layer already exists
d8145dd556fa: Layer already exists
03901b4a2ea8: Layer already exists
cli-7: digest: sha256:61385770922947e44dd2187666d7f84d00b61269be1a1c5665b04bc72810f34a size: 3865
Untagged: registry.datarouter.ai/exact-spacecraft-1900/airflow:cli-7
Untagged: registry.datarouter.ai/exact-spacecraft-1900/airflow@sha256:61385770922947e44dd2187666d7f84d00b61269be1a1c5665b04bc72810f34a
Deploy succeeded!
dimberman astronomer_pydata (airflow-additions) (⎈|context|default) $
```

```
  File "/usr/local/lib/python3.6/site-packages/airflow/models/__init__.py", line 189, in get_fernet
    for fernet_part in fernet_key.split(',')
  File "/usr/local/lib/python3.6/site-packages/airflow/models/__init__.py", line 189, in <listcomp>
    for fernet_part in fernet_key.split(',')
  File "/usr/local/lib/python3.6/site-packages/cryptography/fernet.py", line 35, in __init__
    key = base64.urlsafe_b64decode(key)
  File "/usr/local/lib/python3.6/base64.py", line 133, in urlsafe_b64decode
    return b64decode(s)
  File "/usr/local/lib/python3.6/base64.py", line 87, in b64decode
    return binascii.a2b_base64(s)
binascii.Error: Incorrect padding

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/bin/airflow", line 32, in <module>
    args.func(args)
  File "/usr/local/lib/python3.6/site-packages/airflow/bin/cli.py", line 1105, in resetdb
    db.resetdb(settings.RBAC)
  File "/usr/local/lib/python3.6/site-packages/airflow/utils/db.py", line 383, in resetdb
    initdb(rbac)
  File "/usr/local/lib/python3.6/site-packages/airflow/utils/db.py", line 102, in initdb
    schema='default'))
  File "<string>", line 4, in __init__
  File "/usr/local/lib/python3.6/site-packages/sqlalchemy/orm/state.py", line 428, in _initialize_ins
tance
    manager.dispatch.init_failure(self, args, kwargs)
  File "/usr/local/lib/python3.6/site-packages/sqlalchemy/util/langhelpers.py", line 67, in __exit__
    compat.reraise(exc_type, exc_value, exc_tb)
  File "/usr/local/lib/python3.6/site-packages/sqlalchemy/util/compat.py", line 277, in reraise
    raise value
  File "/usr/local/lib/python3.6/site-packages/sqlalchemy/orm/state.py", line 425, in _initialize_ins
tance
    return manager.original_init(*mixed[1:], **kwargs)
  File "/usr/local/lib/python3.6/site-packages/airflow/models/connection.py", line 125, in __init__
    self.extra = extra
  File "<string>", line 1, in __set__
  File "/usr/local/lib/python3.6/site-packages/airflow/models/connection.py", line 181, in set_extra
    fernet = get_fernet()
  File "/usr/local/lib/python3.6/site-packages/airflow/models/__init__.py", line 193, in get_fernet
    raise AirflowException("Could not create Fernet object: {}".format(ve))
airflow.exceptions.AirflowException: Could not create Fernet object: Incorrect padding
$ (pydata) dimberman ~ (⎈|context|default) $

Session Contents Restored

Last login: Mon Sep 23 08:14:03 on ttys000
dimberman ~ (⎈|context|default) $
```

TL;DR…

Rappi
ASTRONOMER

- Iteratively built a working model using JupyterNotebooks
- Parameterized our notebooks using Papermill
- Tuned, tested, compared, and productionized our model using Apache Airflow
- Scaled our model building using Kubernetes or Celery



Rappi

ASTRONOMER

Questions?

Thank you!

Daniel Imberman

Platform Engineer

@danimberman

@d_imberz

www.astronomer.io

Gonzalo Diaz

Senior Backend Data Engineer

@gdcor

@gondcor

www.rappi.com.ar

Join us! gonzalo.diaz@rappi.com