



RubiOne

# APACHE SPARK AS THE BACKBONE OF A RETAIL ANALYTICS DEVELOPMENT ENVIRONMENT



Chief Engineer – Adrian Petrescu



RUBIKLOUD

# CORPORATE FOUNDATIONS

8

3

2

Rubikloud is a venture-backed software and technology focused startup. Rubikloud gives retailers the power to lead their markets.

Rubikloud's investors include:

- Horizons Ventures
- Access Industries
- MaRS Investment Accelerator Fund
- TOM Group Limited

Suite 1100, 15 Toronto St,  
Toronto, Canada, M5C 2E3

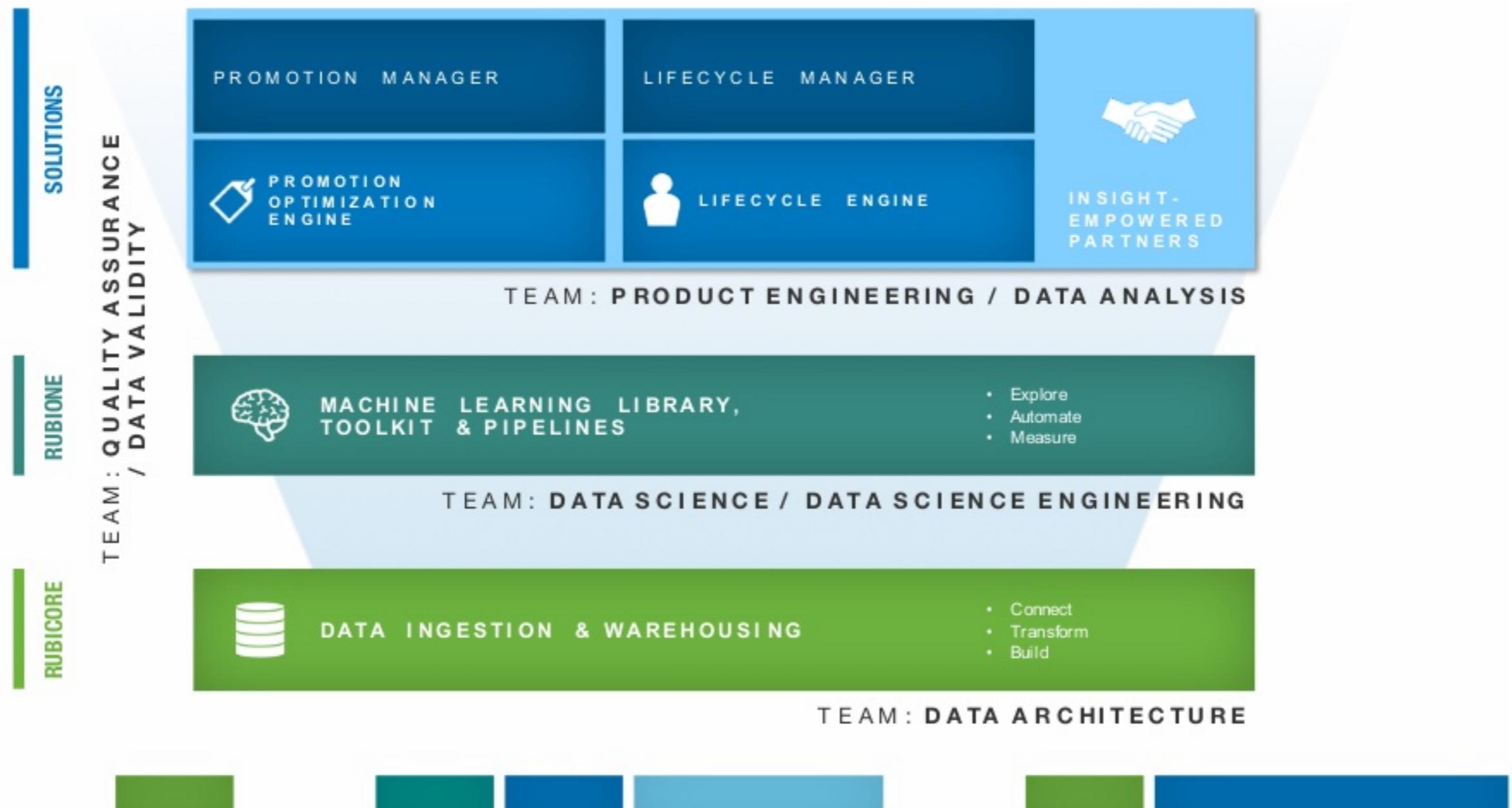
LAST 2 YEARS  
**15-50 Teammates**

GROWING TO  
**100+ (2017)**

FOUNDED EARLY-2013



# The Rubikloud Platform





## Data Science Development Platform RubiOne

### CAPABILITIES

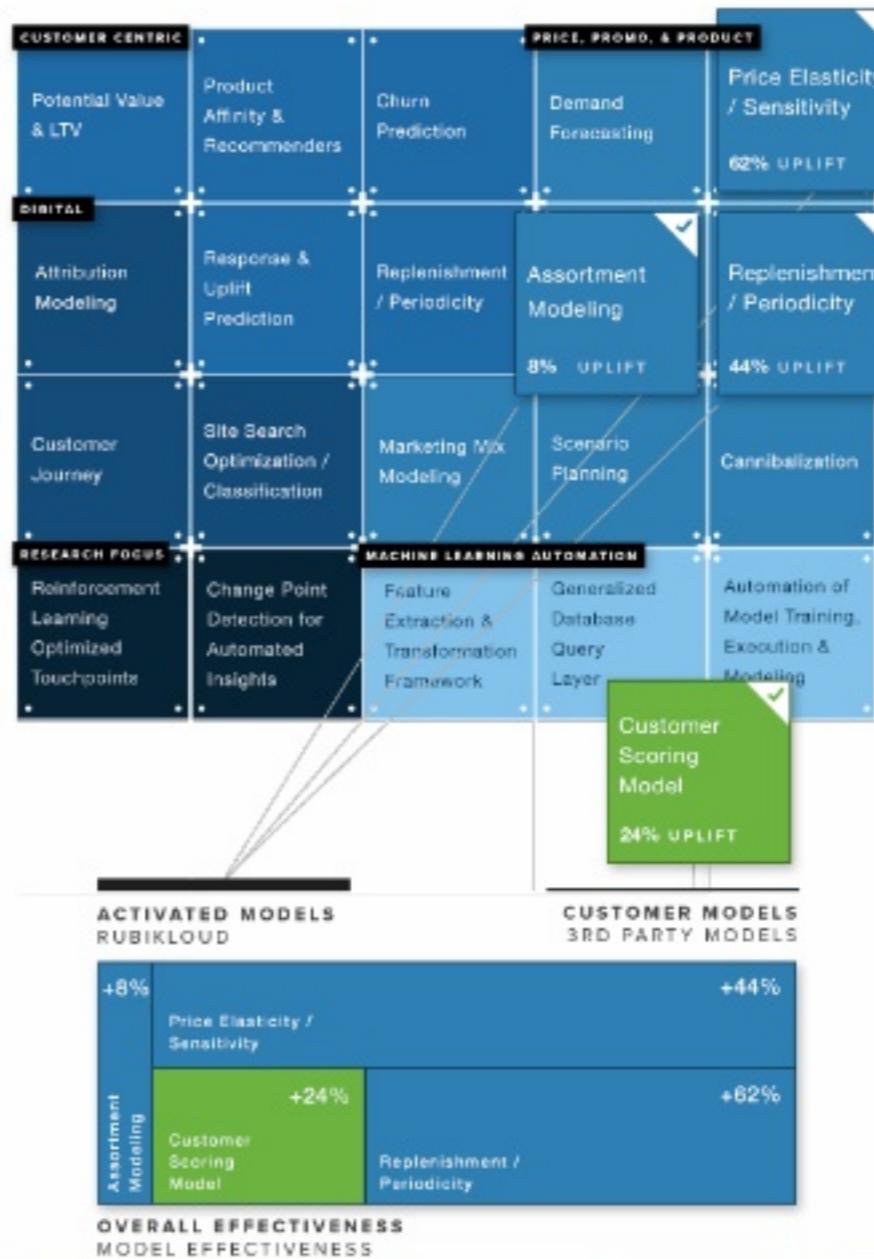
Develop your own models, at scale

Leverage our vertical specific feature-engineering framework

Chain together models in pipelines to deploy multi-stage systems into production

Incorporate or benchmark against Rubikloud's own models, trained on several years of retail data

Leverage various visualization libraries to compare and monitor the performance of models





- Completely self-contained deployment, to work around long IT turnaround times





- Completely self-contained deployment, to work around long IT turnaround times
- Integrations with common enterprise retail authentication systems, data backends, and execution layers





- Completely self-contained deployment, to work around long IT turnaround times
- Integrations with common enterprise retail authentication systems, data backends, and execution layers
- Integrate with familiar tools, while providing transparent scalability





- Completely self-contained deployment, to work around long IT turnaround times
- Integrations with common enterprise retail authentication systems, data backends, and execution layers
- Integrate with familiar tools, while providing transparent scalability
- Multi-cloud/cloud-agnostic





- Completely self-contained deployment, to work around long IT turnaround times
- Integrations with common enterprise retail authentication systems, data backends, and execution layers
- Integrate with familiar tools, while providing transparent scalability
- Multi-cloud/cloud-agnostic
- **Have the ability to apply policy to *everything***





- Completely self-contained deployment, to work around long IT turnaround times
- Integrations with common enterprise retail authentication systems, data backends, and execution layers
- Integrate with familiar tools, while providing transparent scalability
- Multi-cloud/cloud-agnostic
- Have the ability to apply policy to *everything*
- **Batteries included**





- Deep learning





RubiOne

**NON-Goals**





- Deep learning
- Low-latency results





- Deep learning
- Low-latency results
- Streaming input





- Deep learning
- Low-latency results
- Streaming input
- “Big” data









**RubiOne**



RubiOne

## What It Is



- Interactive, ad-hoc development
- Backed by JupyterHub and Jupyter Notebooks
- Integrated with Rubikloud's retail vertical-specific visualizations, ORM, and machine learning library.



RubiOne

## What It Is



- Powerful APIs for production-grade machine learning in retail
- Built-in feature extraction, dependency management, and parameter tuning



RubiOne

## What It Is



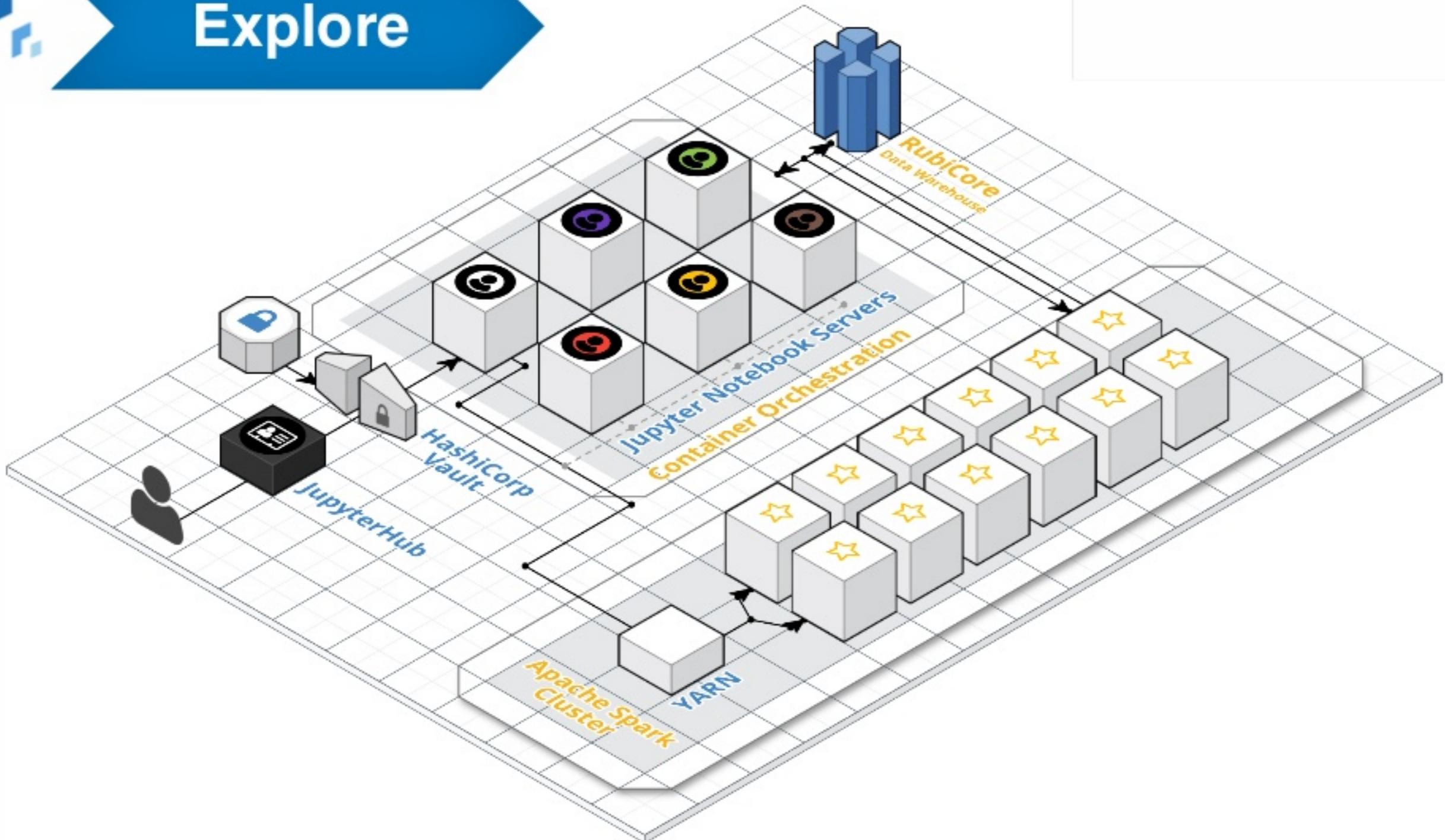
- Plugs in directly to the computational graph for our existing models
- Schedules batch jobs according to data freshness/staleness
- Output directly into execution tools



Apache Spark is the backbone that makes all of this possible.

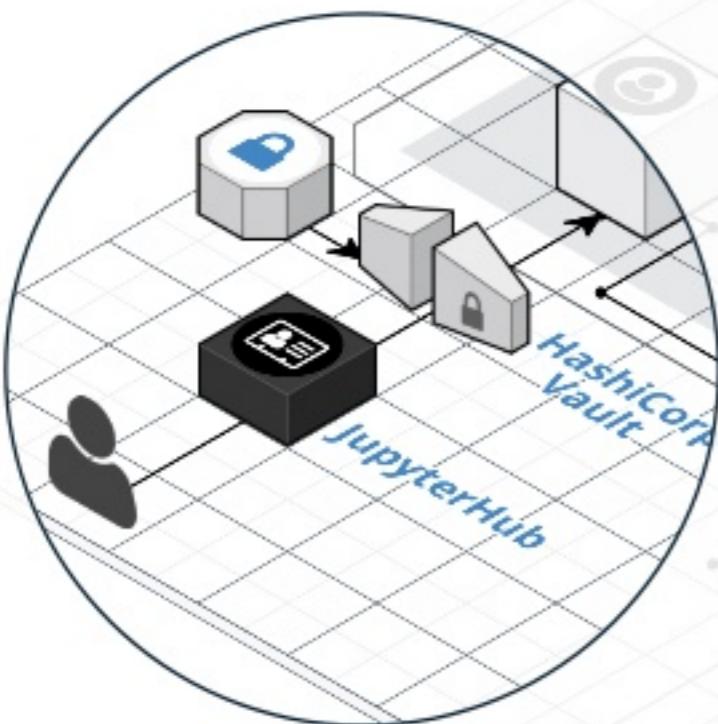


# Explore

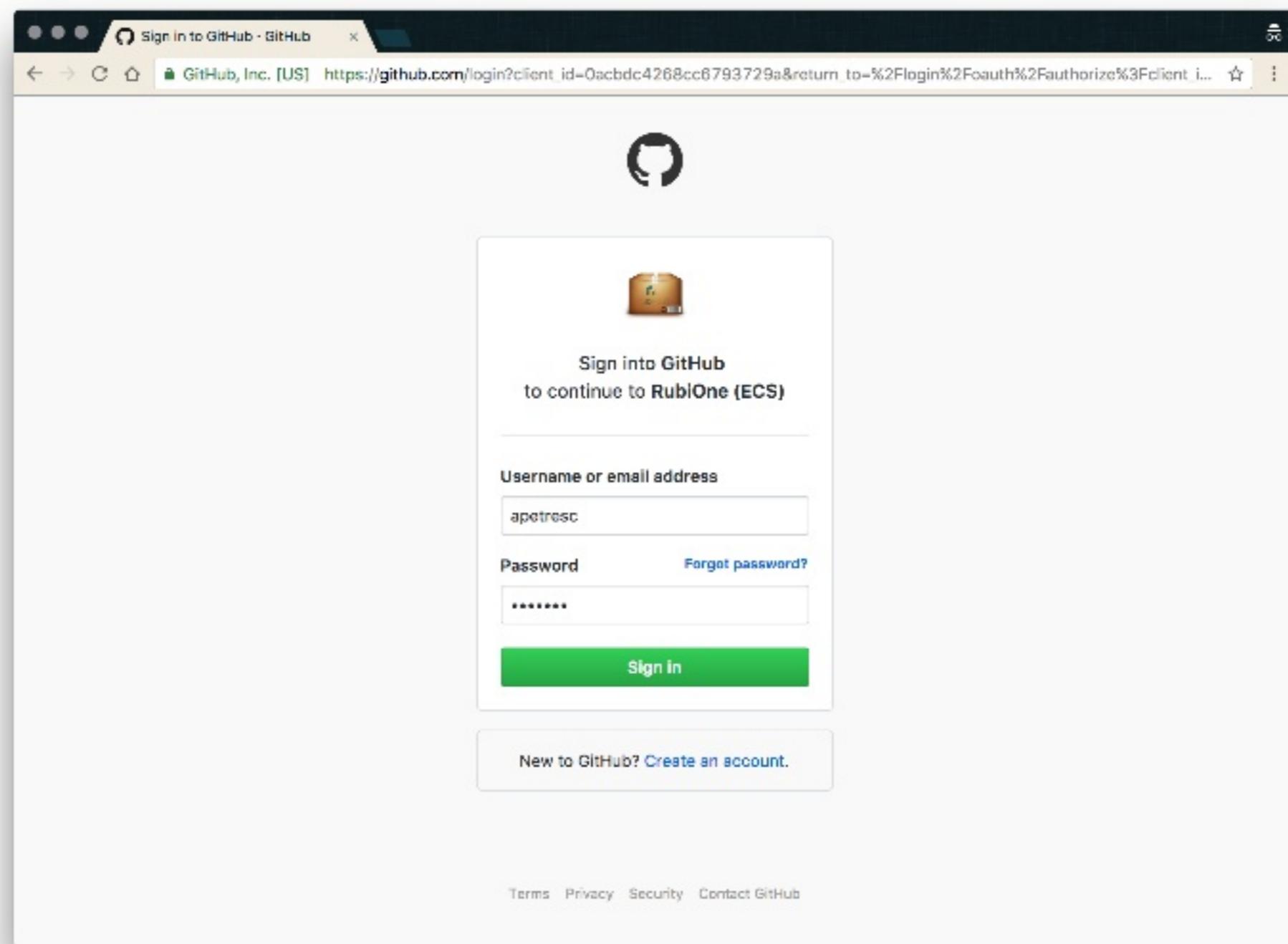


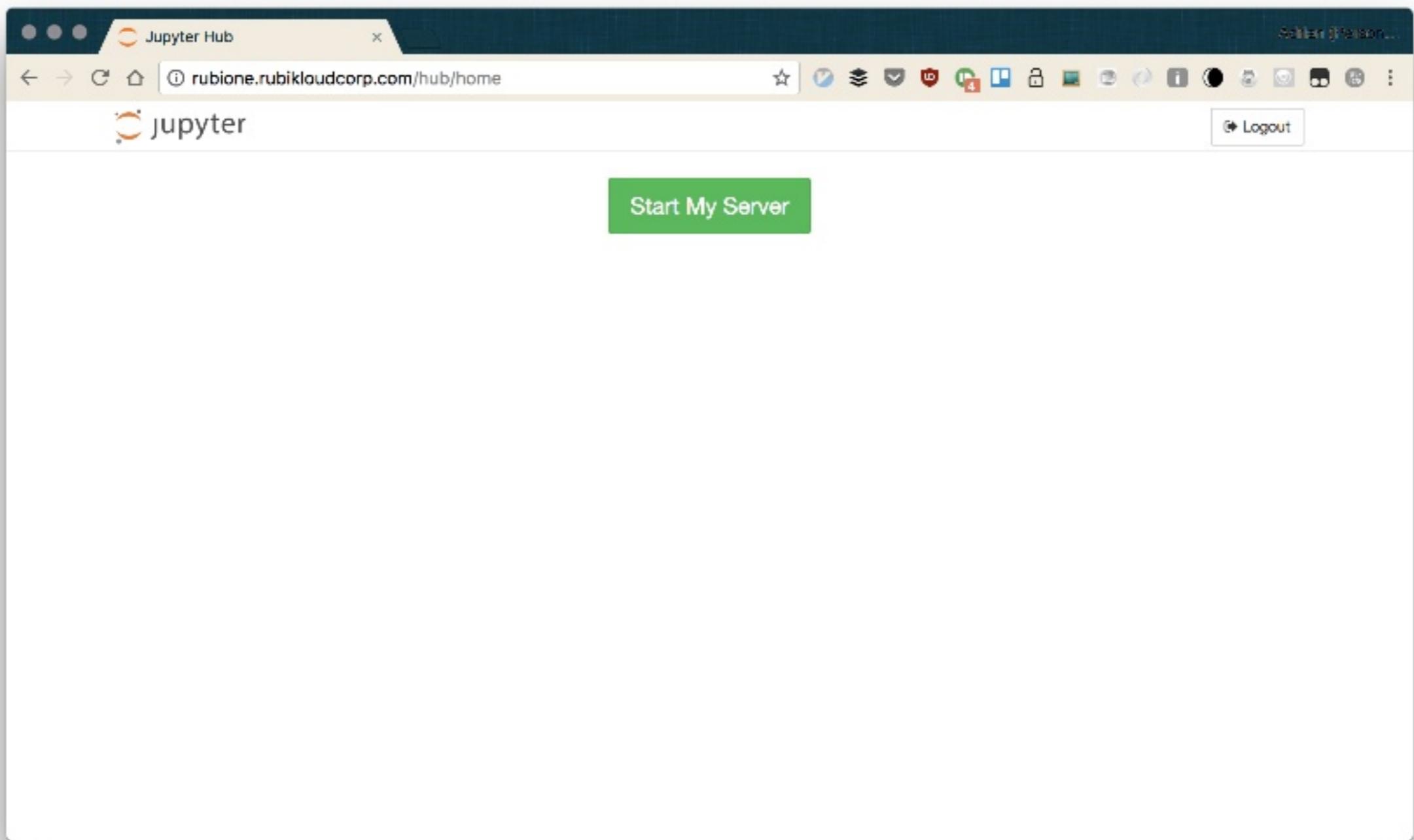


## Explore



- JupyterHub is a multi-user hub which spawns and proxies per-user Jupyter Notebook instances
- Integrates with LDAP, OAuth, SAML, etc through its Authenticator interface
- Integrates with multiple orchestration systems through its Spawner interface
- Responsible for provisioning individual workers with configuration and authentication tokens







## Explore

- Hosts Rubikloud's retail-specific data schema, RDM
- Also contains extension tables, model outputs, and measurement results specific to each client
- Row- and column-level policy enforced at the role level
- Convenient ORM takes advantage of the fact that schema is consistent across BUs and clients



Apache Spark  
Cluster

YARN

## Explore

- Spawns as on-demand ephemeral containers behind a VPC – not accessible on the public internet
- Preloaded with all the tools of the trade:
  - Kernels for Python, R, and Scala
  - Stable version set of common libraries like NumPy, scikit-learn
- Mounted volumes to instantly sync notebooks





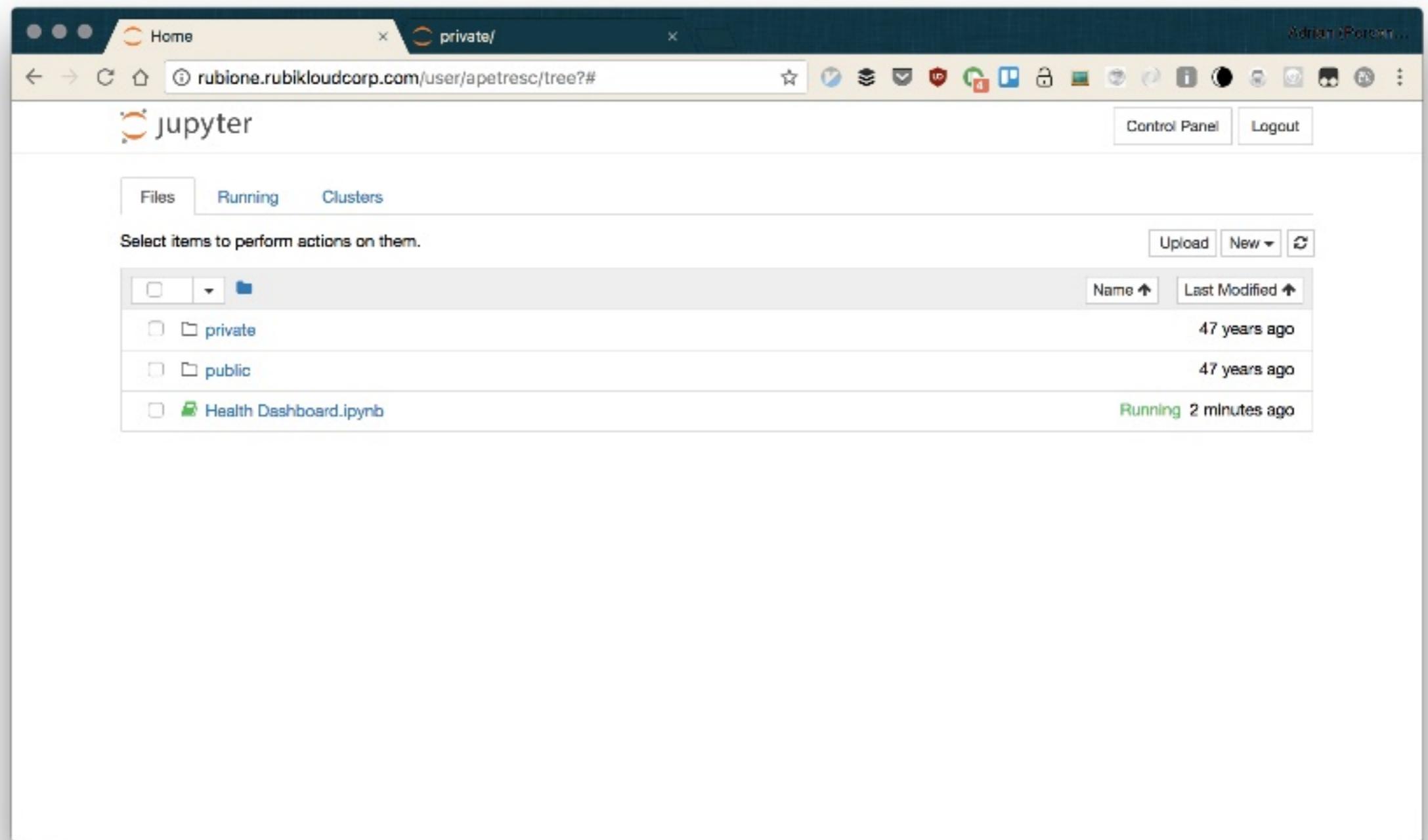
Screenshot of a web browser showing the Rubik Cloud interface. The address bar displays `rubione.rubikloudcorp.com/user/apetresc/tree?#`. The page title is "private/" and the user is "Adrian (Perez)".

The main content area shows a Jupyter notebook environment. A navigation bar at the top includes "Control Panel" and "Logout". Below it, tabs for "Files", "Running", and "Clusters" are visible, with "Running" being the active tab.

A message "Select items to perform actions on them." is displayed above a file list. The file list includes:

	Name	Last Modified
<input type="checkbox"/>	private	47 years ago
<input type="checkbox"/>	public	47 years ago
<input type="checkbox"/>	Health Dashboard.ipynb	Running 2 minutes ago

Buttons for "Upload" and "New" are located at the top right of the file list.





Home    private/    RubiOne Features    Admin (Logout...)

rubione.rubikloudcorp.com/user/apetresc/tree/private

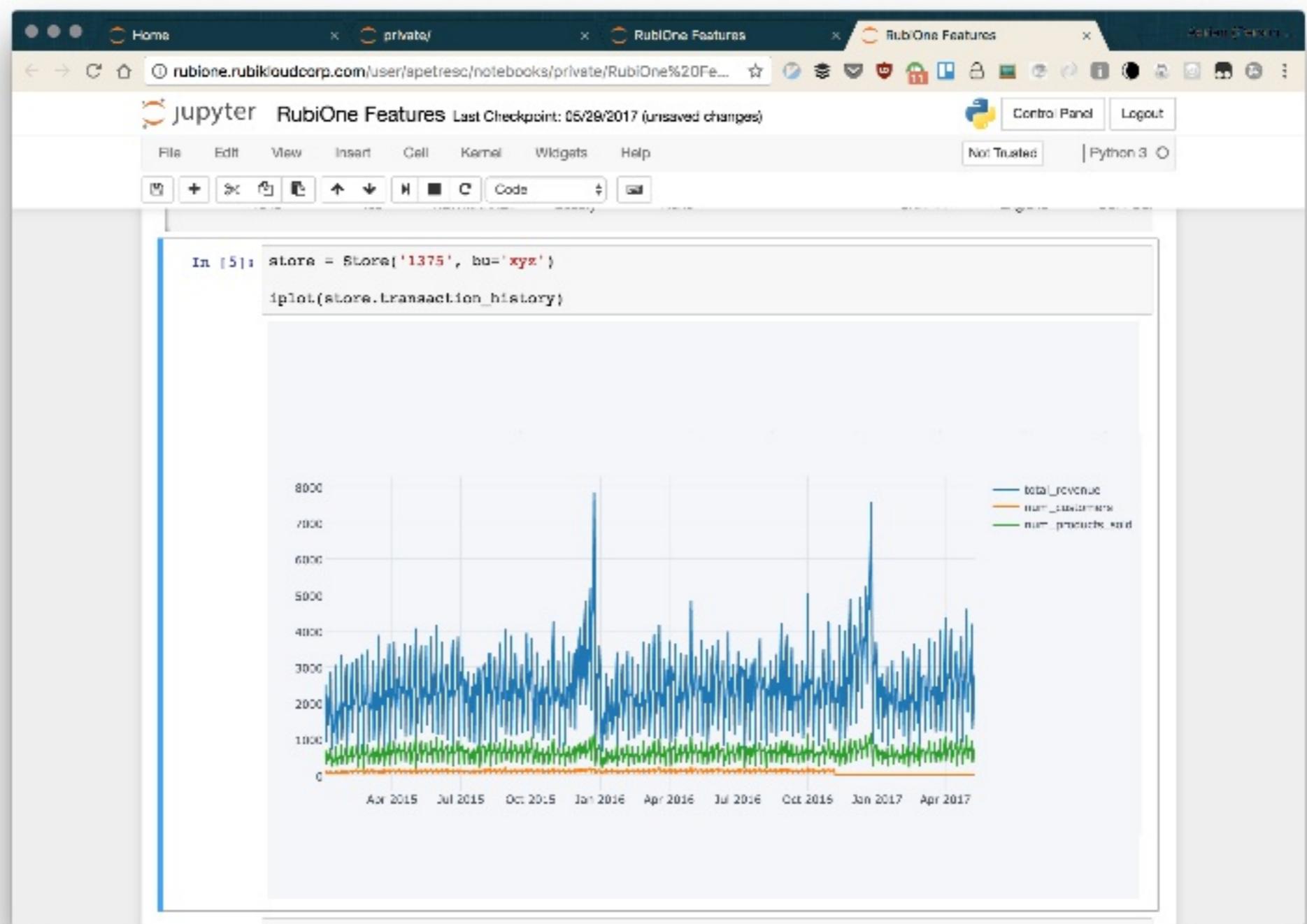
jupyter

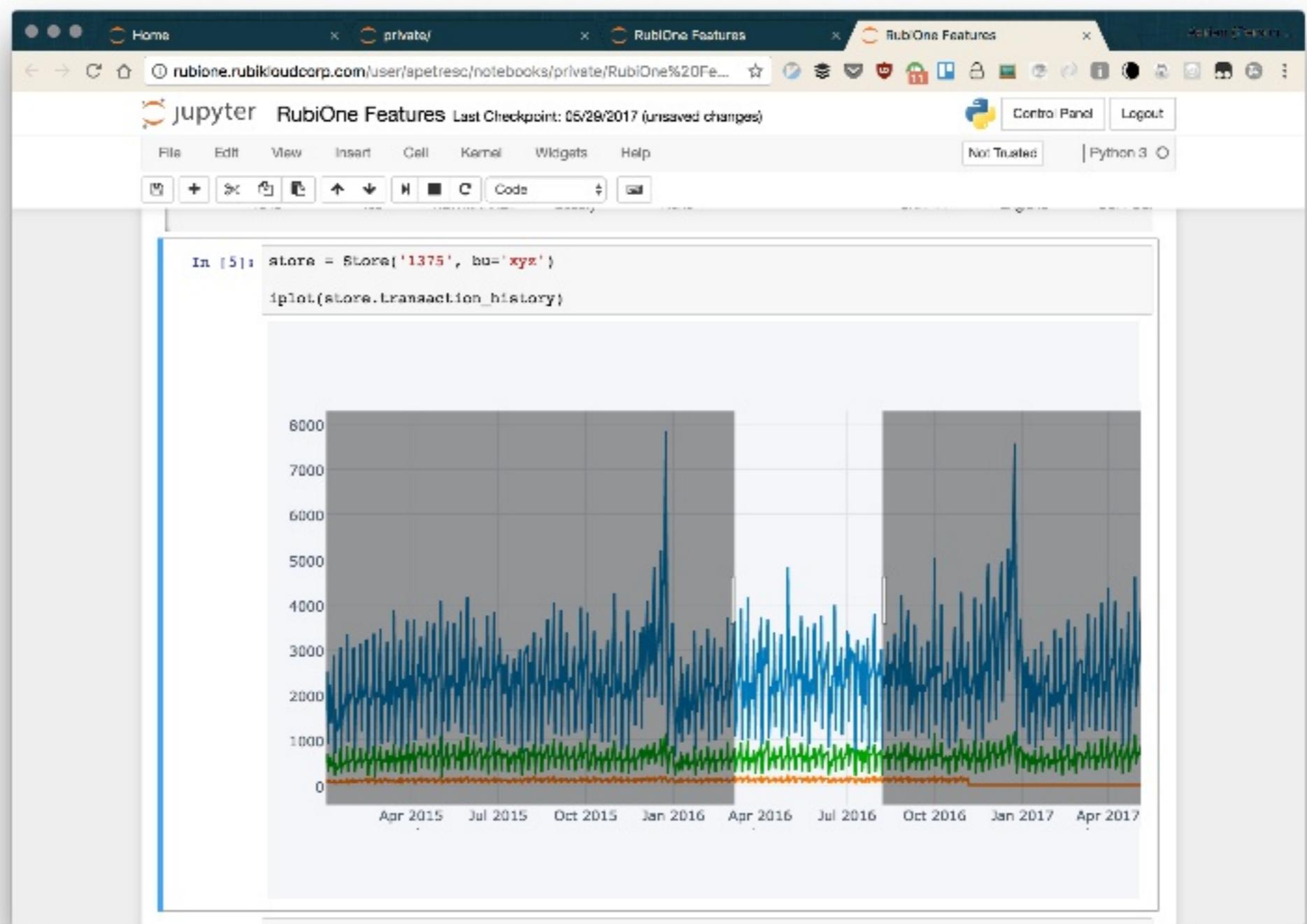
Files    Running    Clusters

Select Items to perform actions on them.

Upload    New   

	Name	Last Modified
<input type="checkbox"/>	..	seconds ago
<input type="checkbox"/>	Primes Stress Test.ipynb	7 days ago
<input type="checkbox"/>	RubiOne Features.ipynb	Running 7 days ago
<input type="checkbox"/>	SparkMagic Test.ipynb	12 days ago







# Explore

File I/O Abstraction

Credential management

nbconvert

## Version control

Spark wrapper kernels

Keyboard shortcuts

LaTeX

Execute timings

R, Scala, Python

Geocoders and GIS imagery

Plot.ly

sparkmagic

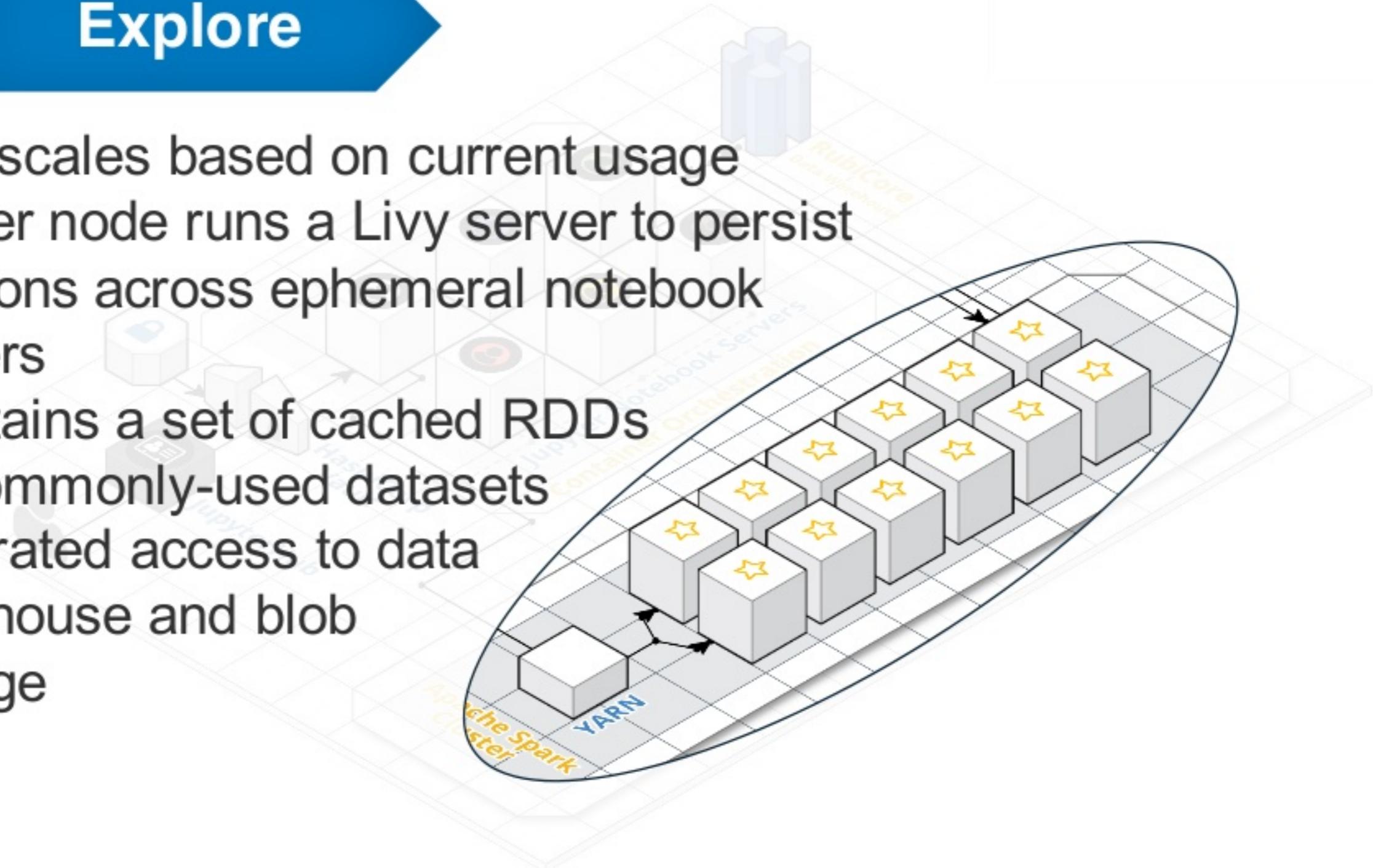
Snippets

Excel/CSV import

A/B Testing

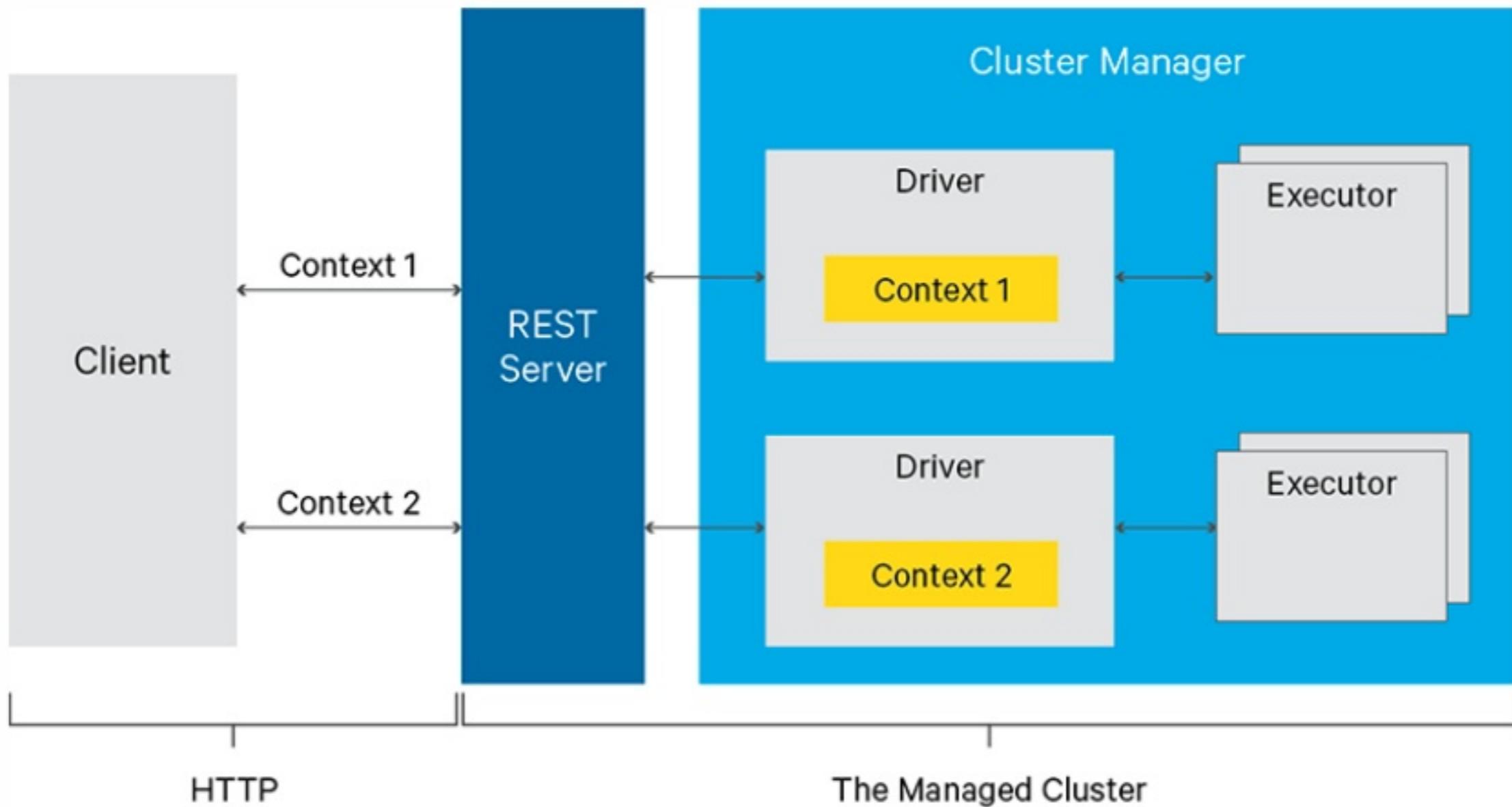
# Explore

- Auto-scales based on current usage
  - Master node runs a Livy server to persist sessions across ephemeral notebook servers
  - Maintains a set of cached RDDs for commonly-used datasets
  - Federated access to data warehouse and blob storage





# Explore





# Explore

The screenshot shows a Jupyter Notebook interface running in a web browser. The title bar indicates the notebook is titled "RubiOne Features". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar has buttons for New, Open, Save, Run, Cell, Kernel, Help, and others. The top cell (In [5]) contains the command `%load_ext sparkmagic.magics`. Below it, the top cell (In [6]) contains the command `%spark`. A modal dialog titled "Manage Sessions" is open, showing a table of sessions:

Name	ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1498178218127_0002	spark	idle	Link	Link	✓

The message "SparkSession available as 'spark'" is displayed below the table. The next cell (In [7]) contains the command `spark?`. The subsequent cell (In [8]) contains the command `sc.textFile("hdfs://dev/hc/sample_input_10k.csv").count()`, which returns the result `10000`. The bottom cell (In [9]) is currently empty.

**Docstring:**

```
spark [-c CONTEXT] [-c SESSION] [-c OUTPUT] [-q [QUIET]]  
[-x SAMPLESIZE] [-n MAXROWS] [-r SAMPLEFRAC] [-c URL]  
[-u USER] [-p PASSWORD] [-l LANGUAGE] [-k [SKIP]] [-i ID]  
[command [command ...]]
```

**Magic to execute spark remotely.**

This magic allows you to create a Hivy Scala or Python session against a hivy endpoint. Every session can



# Explore

The screenshot shows a Jupyter Notebook interface running on RubikOne Features. The top navigation bar includes tabs for Home, private/, RubikOne Features, and catherine.devlin/python-sql:. The RubikOne Features tab is active. The browser address bar shows the URL rubikone.rubikloudcorp.com/user/apetresc/notebooks/private/RubiOne%... . The notebook interface has a toolbar with various icons for file operations and cell types (Code, Text, etc.). A table titled "SparkSession available as 'spark'." is displayed, showing one entry:

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1496176218127_0002	pyspark	Idle	<a href="#">Link</a>	<a href="#">Link</a>	✓

Below the table, a code cell (In [7]) contains a complex SQL query using the PySpark SQL API:

```
%%sql -n campaign_history

select
    cc.BU_CODE,
    x.campaign_name,
    x.campaign_code,
    x.TREATMENT_CHANNEL_NAME,
    final_load,
    count(distinct contact_key) as CONTACTS
from
    (
        select
            co.campaign_name,
            co.campaign_code,
            cc.camp_offer_key,
            cc.TREATMENT_CHANNEL_NAME,
            max(cc.load_num) as final_load
        from
            crm_target.b_camp_offer co
        inner join
            crm_target.b_camp_contact cc
        on co.camp_offer_key = cc.camp_offer_key
        where
            launched_ts >= '01-JAN-17'
```



# Explore

Home private/ RubiOne Features catherine.devlin/python-sql: ✓

jupyter RubiOne Features Last Checkpoint: 05/29/2017 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

ID YARN Application ID Kind State Spark UI Driver log Current session?

0 application\_1496176218127\_0002 pyspark Idle Link Link ✓

SparkSession available as 'spark'.

```
In [7]: %%sql -n campaign_history
    select
        cc.BU_CODE,
        x.campaign_name,
        x.campaign_code,
        x.TREATMENT_CHANNEL_NAME,
        final_load,
        count(distinct contact_key) as CONTACTS
    from
        (
            select
                co.campaign_name,
                co.campaign_code,
                cc.camp_offer_key,
                cc.TREATMENT_CHANNEL_NAME,
                max(cc.load_num) as final_load
            from
                crm_target.b_camp_offer co
            inner join
                crm_target.b_camp_contact cc
            on co.camp_offer_key = cc.camp_offer_key
            where
                launched_ts >= '01-JAN-17'
```



# Explore

The screenshot shows a Jupyter Notebook interface with a blue header bar labeled "Explore". The browser tab is titled "RubiOne Features". The notebook cell contains a SQL query:

```
    CRM_target.p_camp_contact cc
    on
    x.camp_offer_key = cc.camp_offer_key
where
    (cc.load_num = final_load and x.camp_offer_key = cc.camp_offer_key)
group by
    cc.BU_CODE,
    x.campaign_name,
    x.campaign_code,
    x.TREATMENT_CHANNEL_NAME,
    final_load
having
    count(distinct contact_key) > 100
    and final_load < 20
;
```

The output of the cell shows the result of the query:

```
In [11]: %%spark -d campaign_history
campaign_history.count()
6548356
```

The bottom cell is currently empty, indicated by "In [ ]:".



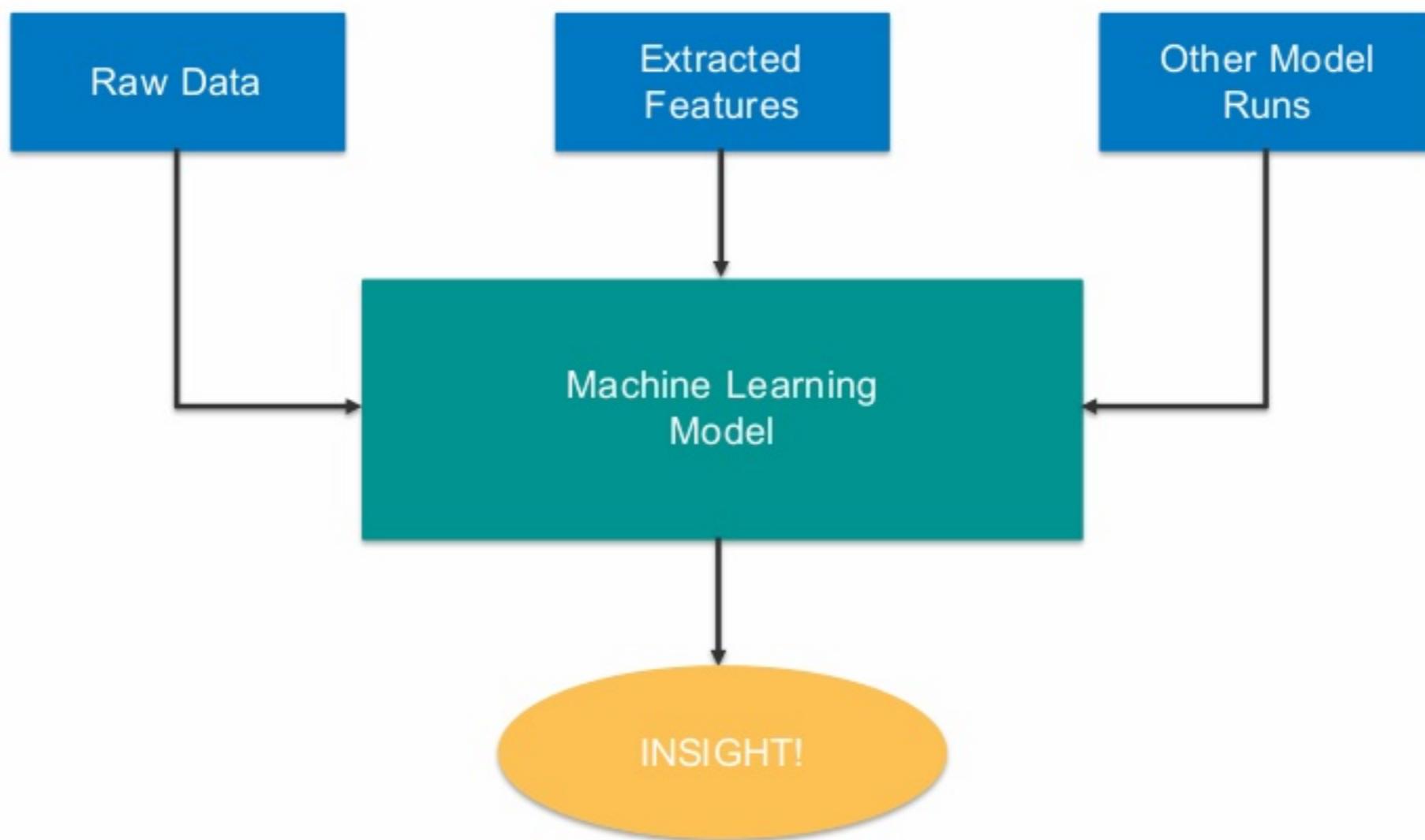
Explore

Develop



# Explore

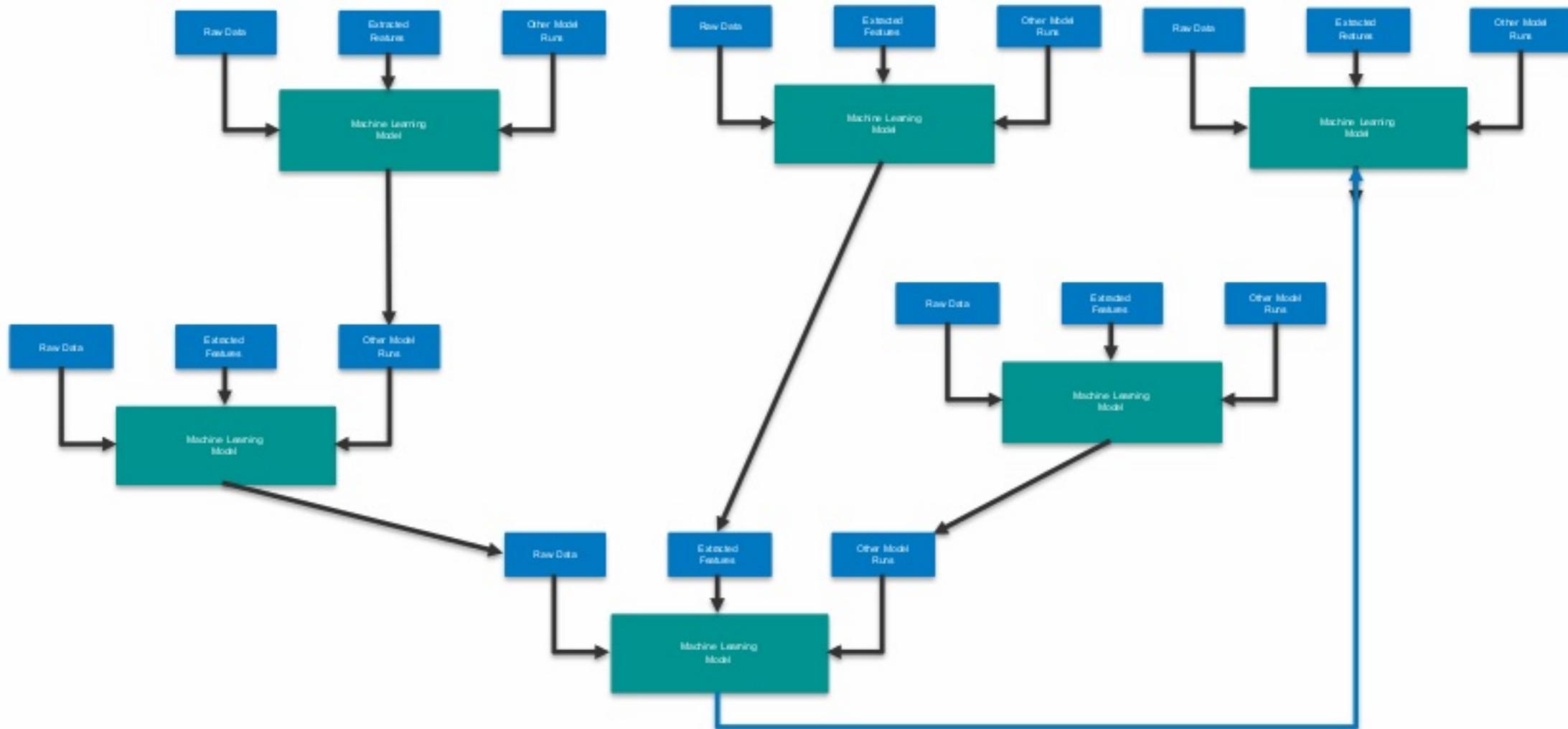
# Develop





# Explore

# Develop





Explore

Develop





Explore

Develop

- Pipeline flow is fixed by the task definition



Explore

Develop

- Pipeline flow is fixed by the task definition
- Parameterization gets to be hugely verbose and redundant



# Explore

# Develop

The screenshot shows a Jupyter Notebook cell with the following Python code:

```
ProductReplenishment.ipynb
```

```
1 class ProductReplenishment(DistributedTask, SparkDistributedTask):
2     bucket = luigi.Parameter()
3     campaign_name = luigi.Parameter()
4
5     def output(self):
6         return S3Target(self.filename)
7
8     def requires(self):
9         return DumpPurchases(campaign_id=self.campaign_name, domain=self.campaign_domain,
10                           dump_bucket=self.dump_bucket,
11                           dump_date=self.dump_date)
12
13     @property
14     def filename(self):
15         return self.compute_filename(suffix=".csv.gz")
16
17
18     @distributed_task
19     def calculate_avg_replenishment(ts):
20         sum(replambda(x, y=x, ts=zip(ts[0], ts[1])) / len(ts) - 1)
21
22     def run(self):
23         if self.complete():
24             return
25
26         dump_data = sc.textFile(self.input().path)
27
28         # The schema on dump_data is:
29         # ('cust_key', 'tsku', 'date', 'orderno', 'brand', 'category', 'quantity', 'price')
30         purchase_averages = dump_data.groupByKey(lambda x: x[0]).mapValues(lambda x: len(x) >= 3).filter(lambda x: len(x) >= 3).mapValues(lambda x: x[2]).mapValues(lambda x: ProductReplenishment.calculate_avg_replenishment(x)).groupByKey(lambda x: x[0][0]).mapValues(lambda x: x[1]).mapValues(lambda x: np.mean(x), np.std(x)).filter(lambda x: x[1][1] < 6)
31
32         with AmazonS3Path(self.filename, S3Client()) as f, gzip.GzipFile(filename=f.name) as zipf:
33             purchase_averages.to_csv(zipf, index=True, index_label='xid', header=True)
```

ProductReplenishment.py 41

L1 Normal L11-B Python 8



# Explore

# Develop

```

1 ProductReplenishment = 
2 class ProductReplenishment(CampaignResource, StockResource):
3     bucket = lambda Parameter(): 
4         campaign_name = Parameter()
5 
6     def output(self):
7         return S3Object(self.filename)
8 
9     def requirements(self):
10         return DataPurchase(xid=lambda x:campaign.xid, domain=lambda campaign:campaign.domain,
11                             dump_bucket=self.model.bucket,
12                             dump_date=self.comp_date)
13 
14     @property
15     def filename(self):
16         return self.compute_filename(suffix='replenish')
17 
18     @staticmethod
19     def calculate_avg_replenishment(x):
20         sum(lambda x,y: y - x, zip(x[1:],x[0])) / (len(x) - 1)
21 
22     def run(self):
23         if self.complete():
24             return
25 
26         dump_data = self.writeFile(self.filename(), path)
27 
28         # The schema on dump_data is:
29         # ('product_key', 'skus', 'category', 'brand', 'category', 'quantity', 'price')
30         purchase_averages = new_data.groupby(lambda x: x[0], x[1]) \
31             .filter(lambda x: len(x) > 3) \
32             .mapValues(lambda x: x[2]) \
33             .mapValues(ProducReplenishment.calculate_avg_replenishment) \
34             .groupByKey() \
35             .mapValues(lambda x: x[0]) \
36             .mapValues(lambda x: np.mean(x), np.var)() \
37             .filter(lambda x: x[1] <= 3)
38 
39         with AtomicS3File(self.filename(), S3Object() as t, grantReadFile=True) as xipto:
40             purchase_averages.to_csv(xipto, index=False, header=False)

```



# Explore

# Develop

```
ProductReplenishment ->
1 class ProductReplenishment(CampaignBaseTask, SparkBaseTask):
2     bucket = Luigi.Parameter()
3     campaign_name = Luigi.Parameter()
4
5     def output(self):
6         return S3Target(self.filename)
7
8     def requires(self):
9         return DumpPurchases(xkid=self.campaign.xkid, domain=self.campaign.domain,
10                           dump_bucket=self.bucket, dump_prefix='',
11                           dump_date=self.campaign.date)
12
13     @property
14     def filename(self):
15         return self.compute_filename(suffix='avg')
16
17     @staticmethod
18     def calculate_avg_replenishment():
19         sum(np(lambda x, y: y - x, ts.zipitstil()[:]) / len(ts) - 1)
20
21     def run(self):
22         if self.complete():
23             return
24
25         dump_data = self.readPickle(self.input().path)
26
27         # The schema on dump_data is:
28         # 'last_order', 'sku', 'category', 'brand', 'category', 'quantity', 'price'
29         purchases_avgmsg = dump_data.groupby(['last_order', 'sku']) \
30             .filter(lambda x: len(x) >= 3) \
31             .aggValues('last_order', 'category', 'brand') \
32             .aggValues(ProductReplenishment.calculate_avg_replenishment) \
33             .groupByKey() \
34             .mapValues(lambda x: np.mean(x), np.std(x)) \
35             .aggValues('last_order', 'category', 'brand') \
36             .filter(lambda x: x['std'] <= 3)
37
38         with AtomicS3File(self.output().path, 'w') as f, gzip.GzipFile(filterobj=f) as zipf:
39             purchases_avgmsg.to_pandas().index=True, index_label='last_order', header=True
40
41
42 ProductReplenishment by 4d
```

```
class ProductReplenishment(  
    CampaignBaseTask,  
    SparkBaseTask):
```



# Explore

# Develop

```
ProductReplenishment:
    class ProductReplenishment(BasePigluTask):
        bucket = luigi.Parameter()
        campaign_name = luigi.Parameter()

    def output(self):
        return S3Output(self.filename)

    def requires(self):
        return DumpPurchases(xkid=self.campaign.xkid, domain=self.campaign.domain,
                             dump_bucket=self.bucket, dump_prefix='dump_data',
                             dump_date=self.campaign.date)

    def filename(self):
        return self.compute_filename(suffix='avg_repl')

    @staticmethod
    def calculate_avg_replenishment():
        summap(lambda x, y: y - x, ts.zipitstil()[:]) / len(ts) + 1

    def run(self):
        if self.complete():
            return

        dump_data = self.readPickle(self.input().path)

        # The schema on dump_data is:
        # ('outstkey', 'sku', 'category', 'brand', 'category', 'quantity', 'price')
        purchases_avgmsg = dump_data.groupby(['brand', 'category', 'category']) \
            .filter(lambda x: len(x) >= 3) \
            .aggvalues('brand', x='xf1') \
            .aggvalues(ProductReplenishment.calculate_avg_replenishment) \
            .groupby(['brand']) \
            .aggvalues('brand', x='xf2') \
            .aggvalues('brand', x='xf3') \
            .aggvalues('brand', x='xf4') \
            .filter(lambda x: len(x) <= 3)

        with StringIO(pickle.dumps(purchases_avgmsg)) as f, gzip.GzipFile(filterobj=f) as zipf:
            purchases_avgmsg.to_csv(zipf, index=True, index_label='brand', header=True)
```

bucket = luigi.Parameter()  
campaign\_name = campaign.Parameter()



# Explore

# Develop

```
1 ProductReplenishment:
2     def __init__(self, campaign_rkid, domain):
3         self.rkid = campaign_rkid
4         self.domain = domain
5
6     def output(self):
7         return S3Target(self.filename)
8
9     def requires(self):
10        return DumpPurchases(rkid=self.campaign.rkid, domain=self.campaign.domain,
11                               dump_bucket=self.model_bucket,
12                               dump_date=self.dump_date)
13
14     @property
15     def filename(self):
16         return self.compute_filename(suffix='.csv.gz')
17
18     @staticmethod
19     def calculate_avg_replenishment():
20         sum(np(lambda x, y: y - x, ts.zipitstir()) / len(ts) - 1)
21
22     def run(self):
23         if self.complete():
24             return
25
26         dump_data = self.read_file(self.input().path)
27
28         # The schema on dump_data is:
29         # 'rout_id', 'sku', 'category', 'brand', 'category', 'quantity', 'price'
30         purchases_avg = dump_data.groupby(['rout_id', 'sku']) \
31                         .filter(lambda x: len(x) >= 3) \
32                         .agg(values('rout_id': xf2)) \
33                         .aggValues(ProductReplenishment.calculate_avg_replenishment) \
34                         .groupby(['rout_id']) \
35                         .aggValues('rout_id': xf1) \
36                         .aggValues('rout_id': np.mean) \
37                         .filter(lambda x: np.mean(x) >= 0)
38
39         with AtomicS3File(self.output(), 'w') as f:
40             gzip.GzipFile(filterobj=f) as zipf:
41                 purchases_avg.to_csv(zipf, index=True, index_label='rout_id', header=True)
```

```
def output(self):
    return S3Target(self.filename)

def requires(self):
    return DumpPurchases(
        rkid=self.campaign.rkid,
        domain=self.campaign.domain,
        dump_bucket=self.model_bucket,
        dump_date=self.dump_date)
```

```
@property
def filename(self):
    return self.compute_filename(
        suffix='*.csv.gz')
```



# Explore

# Develop

```
1 ProductReplenishment:
2     def __init__(self, campaign_name, dump_bucket):
3         self.bucket = Luigi.Parameter()
4         self.campaign_name = Luigi.Parameter()
5
6         self.output_file():
7             return self.bucket.getpath(self.campaign_name, domain=1)
8
9         def requires(self):
10             return DumpPurchases(xid=1, campaign=xid, domain=1, campaign_domain,
11                                   dump_bucket=self.bucket, model_bucket=dump_bucket,
12                                   dump_date=self.campaign_date)
13
14     @property
15     def filenames(self):
16         return self.compute_filenames(suffix='avg')
17
18     @staticmethod
19     def calculate_avg_replenishment(ts):
20         sum(map(lambda x, y: y - x, ts.zip(ts[1:]))) / (len(ts) - 1)
21
22     def run(self):
23         if self.complete():
24             return
25
26         dump_data = sc.textFile(self.input().path)
27
28         # The schema on dump_data is:
29         # ('out_key', 'sku', 'category', 'brand', 'category', 'quantity', 'prior')
30         purchase_averages = dump_data.groupBy(lambda x: x[1], x[0])
31
32         .filter(lambda x: len(x) >= 30)
33         .mapValues(lambda x: x[2])
34         .mapValues(ProductReplenishment.calculate_avg_replenishment)
35         .groupByKey()
36         .mapValues(lambda x: x[1])
37         .mapValues(lambda x: np.mean(x), np.std(x))
38         .filter(lambda x: np.std(x) <= 30)
39
40         with AtomicS3File(self.output_file(), s3client() as t, gzip.GzipFile(filename=t) as zipf:
41             purchase_averages.to_csv(zipf, index=True, index_label='out_key', header=True)
42
```

```
@staticmethod
def calculate_avg_replenishment(ts):
    return sum(map(lambda x, y: y - x, ts.zip(ts[1:]))) / (len(ts) - 1)
```

```
def run(self):
    dump_data = sc.textFile(
        self.input().path)
    purchase_averages = dump_data
        .groupBy(...)
        .mapValues(...)
    ...
    .filter(lambda x: x[1][1] <= D)
```



# Explore

## Develop

```
def workflow(self):
```

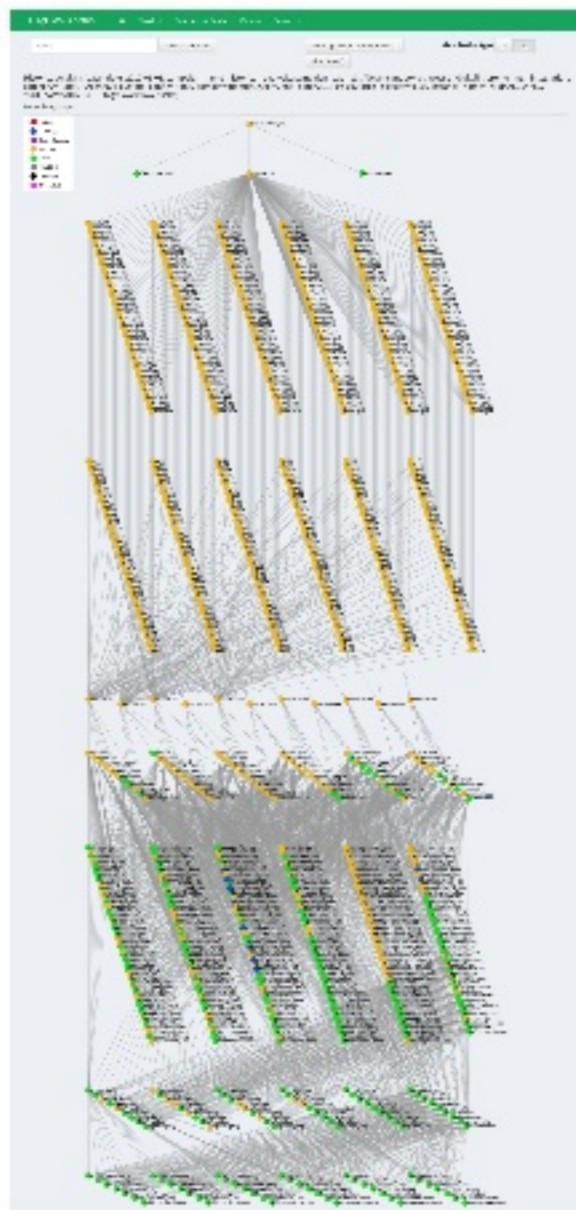
```
self.task_name_1.input_param  
= self.task_name_2.output_param
```



Explore

Develop

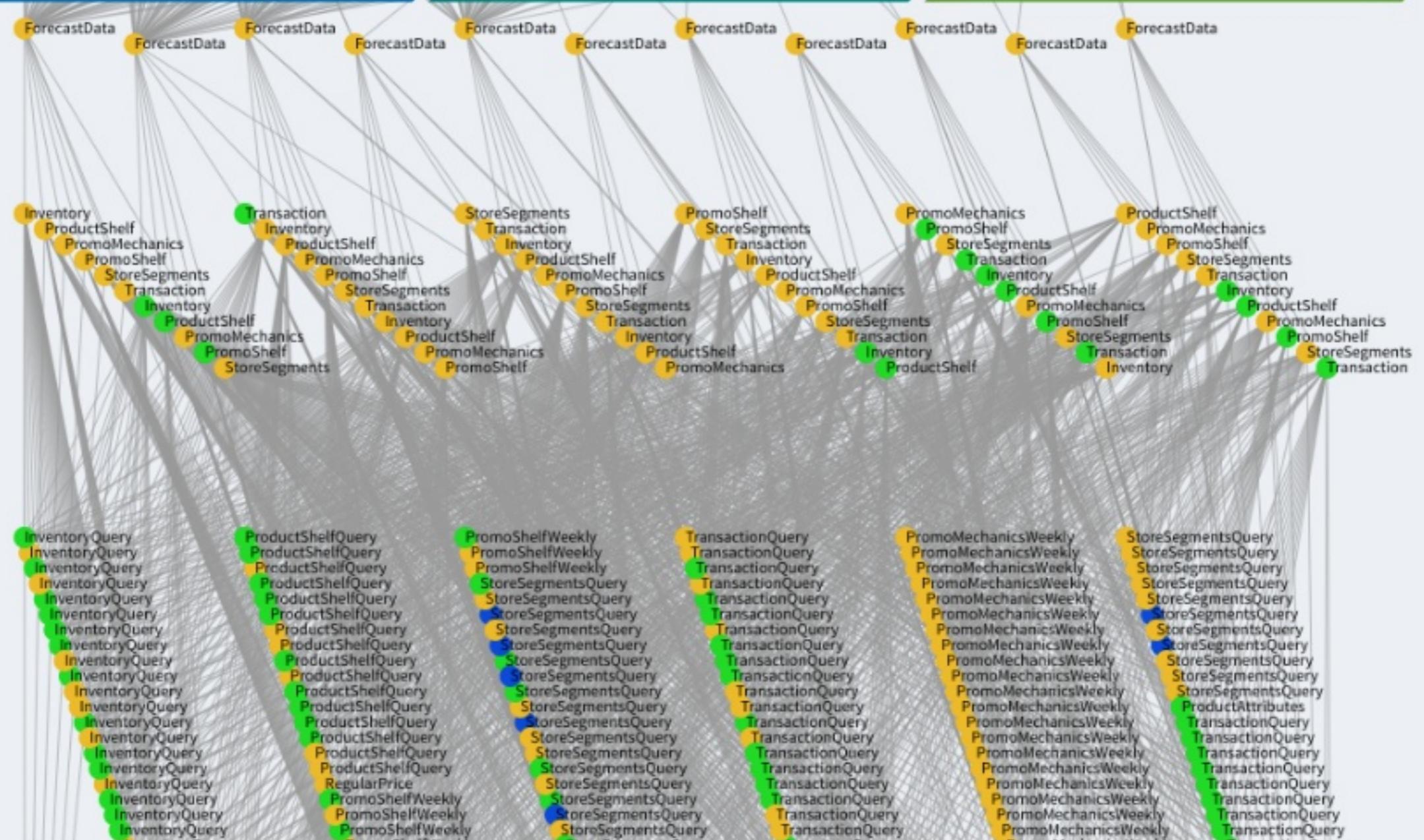
Automate



# Explore

## Develop

# Automate





Case Study: RubiOne

## Personalized Search Results based on Relevancy Scores

$$W_f = aM_w + bR_w$$

$$\mathbf{R} = \begin{bmatrix} r_{1,1} & \dots & r_{1,N} \\ \dots & r_{i,j} & \dots \\ r_{M,1} & \dots & r_{M,N} \end{bmatrix}$$

$$\mathbf{R} = \mathbf{U}\mathbf{P} = \begin{bmatrix} u_{1,1} & \dots & u_{1,F} \\ \dots & u_{i,j} & \dots \\ u_{M,1} & \dots & u_{M,F} \end{bmatrix} \begin{bmatrix} p_{1,1} & \dots & p_{1,N} \\ \dots & p_{i,j} & \dots \\ p_{F,1} & \dots & p_{F,N} \end{bmatrix}$$

$$\min \sum_{r_{i,j} \text{ is known}} (r_{i,j} - x_i^T y_j) + \lambda(\|x_i\|^2 + \|y_j\|^2)$$

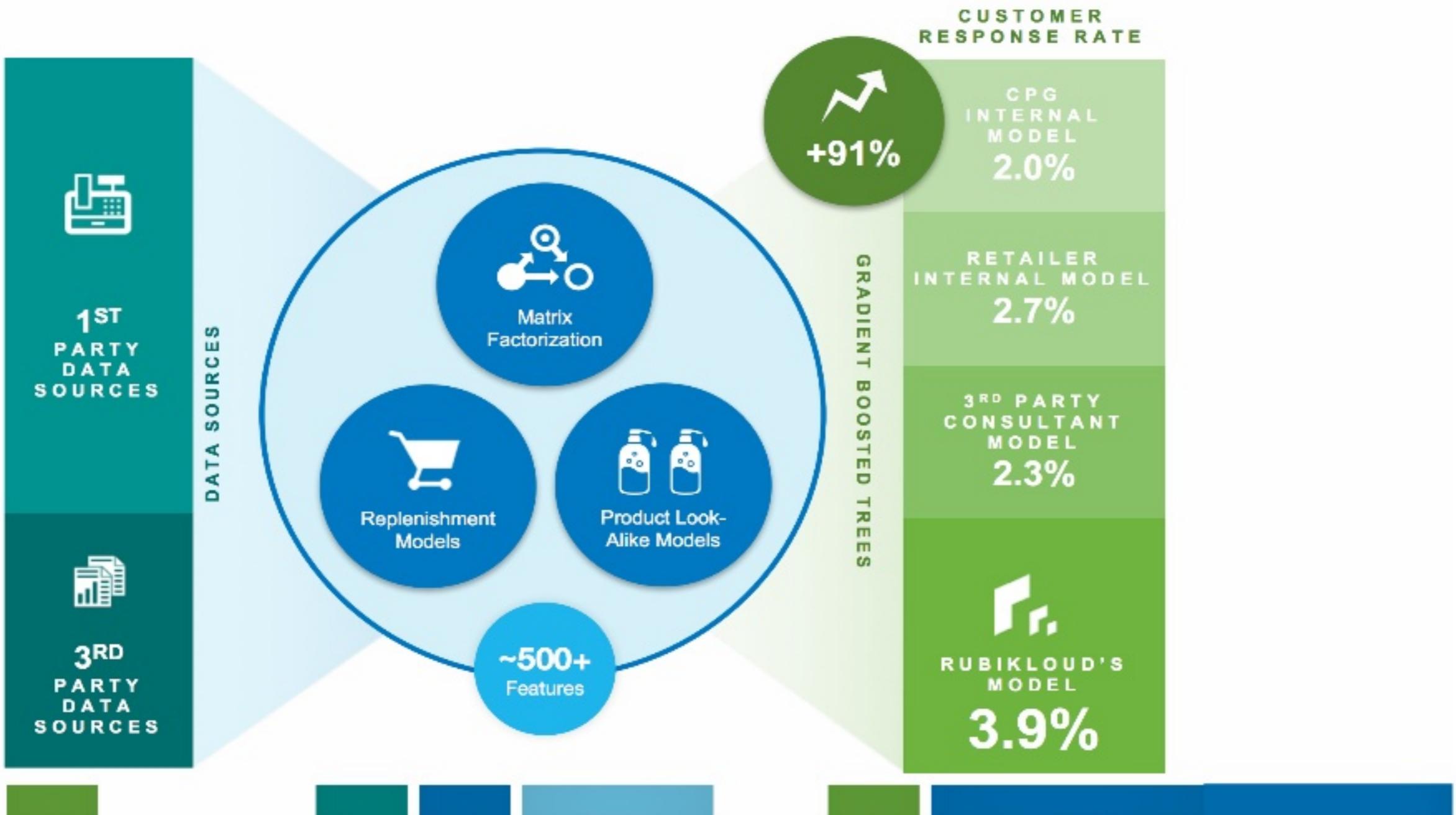
**ORACLE®**  
**ENDECA**





CASE STUDY: RUBIONE

## Use Machine Learning to Solve Business Problems – At Scale





FIN

**That's All, Folks!**

