



Open Data Science Conference - East



# Henosis

A generalizable, cloud-native Python recommender framework

Valentino Constantinou, Ian Colwell – May 4<sup>th</sup>, 2018

Copyright 2018 California Institute of Technology. U.S. Government sponsorship acknowledged



**Jet Propulsion Laboratory**  
California Institute of Technology

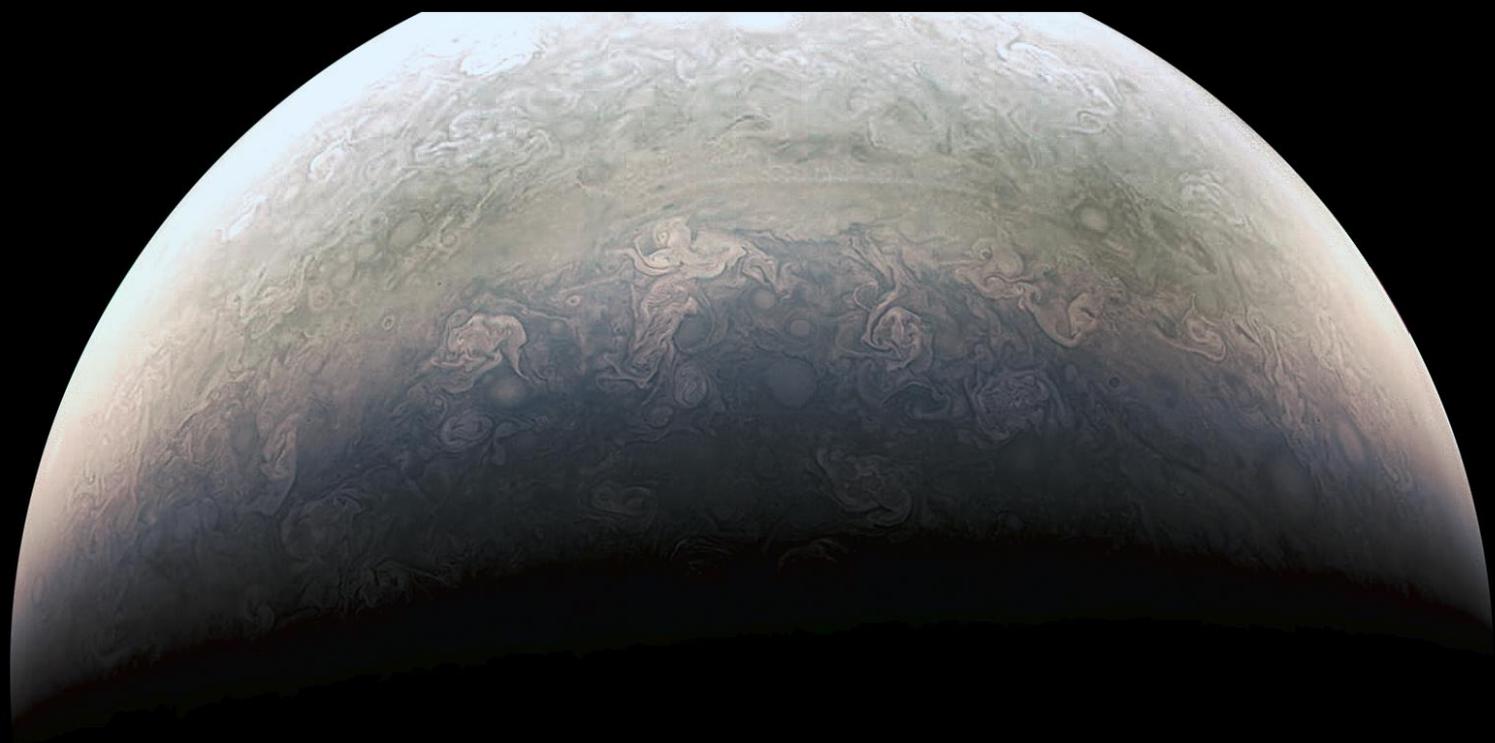
- 1. Introduction**
- 2. Recommendations Made Easier**
- 3. Implementation & Proof of Concept**
- 4. Wrap-Up**

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

# 1. Introduction

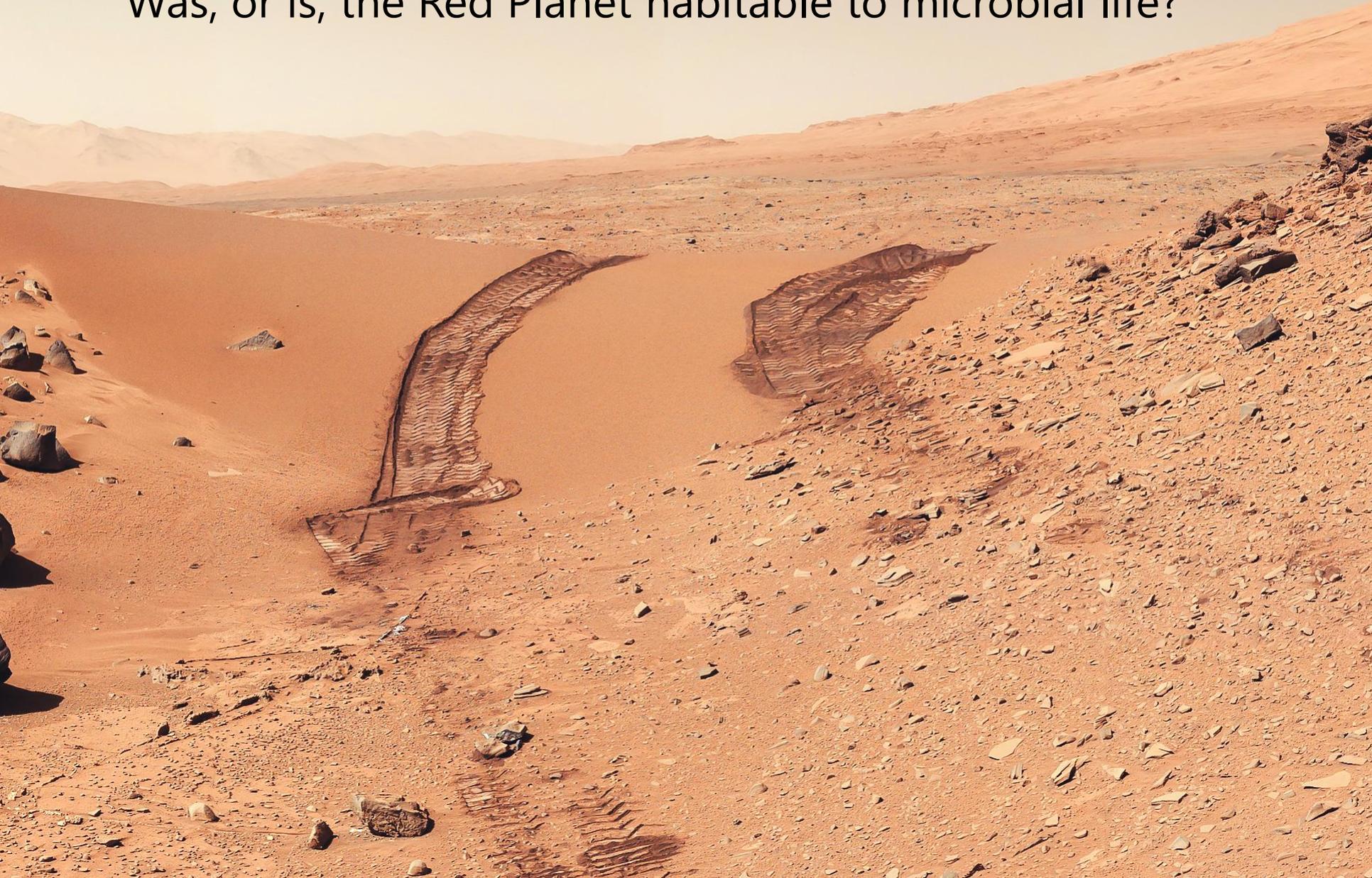
**Leaders** in the robotic exploration of the solar system.

We look to answer questions like...

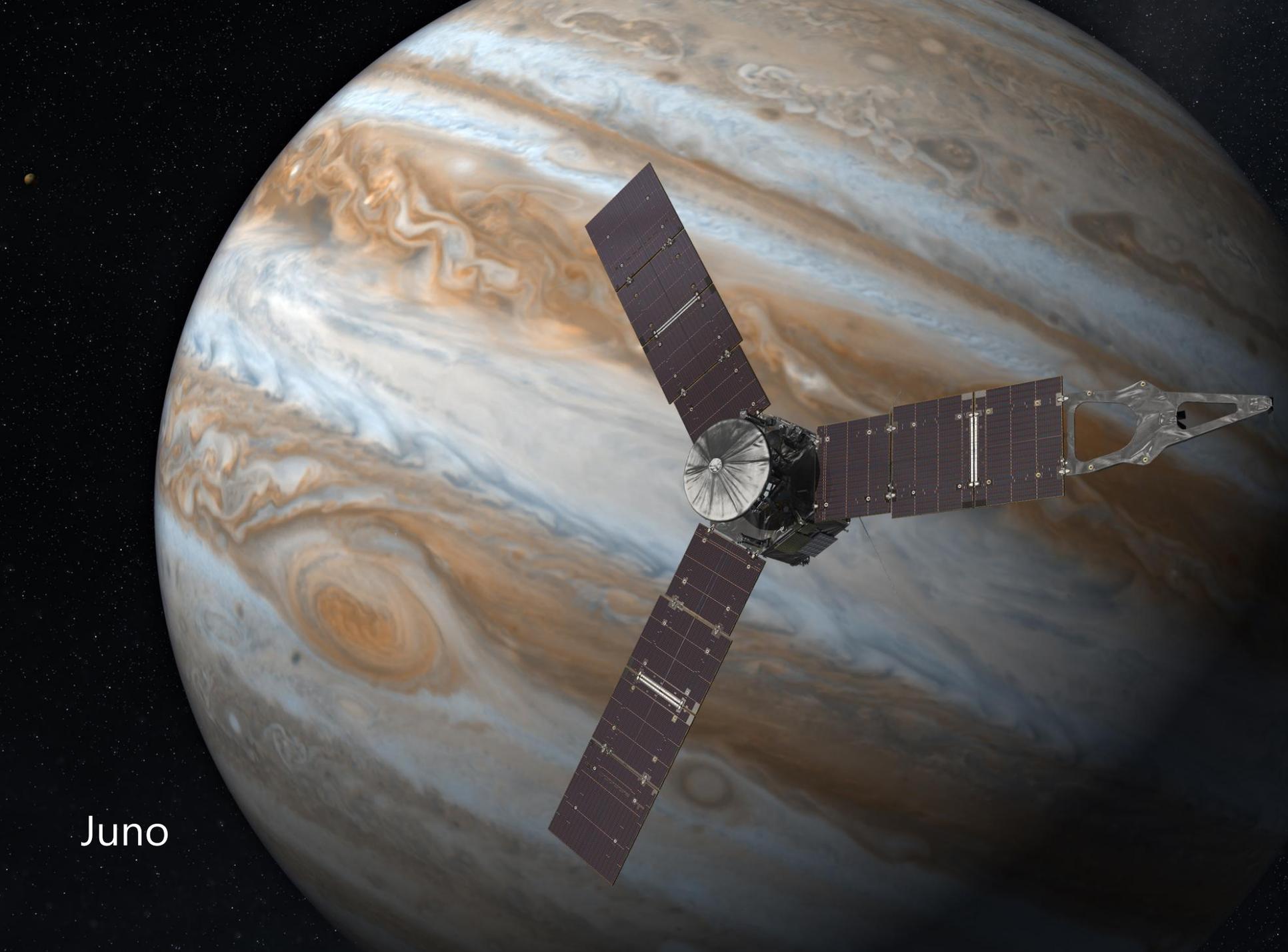


What can Jupiter's formation and evolution tell us about our solar system?

Was, or is, the Red Planet habitable to microbial life?

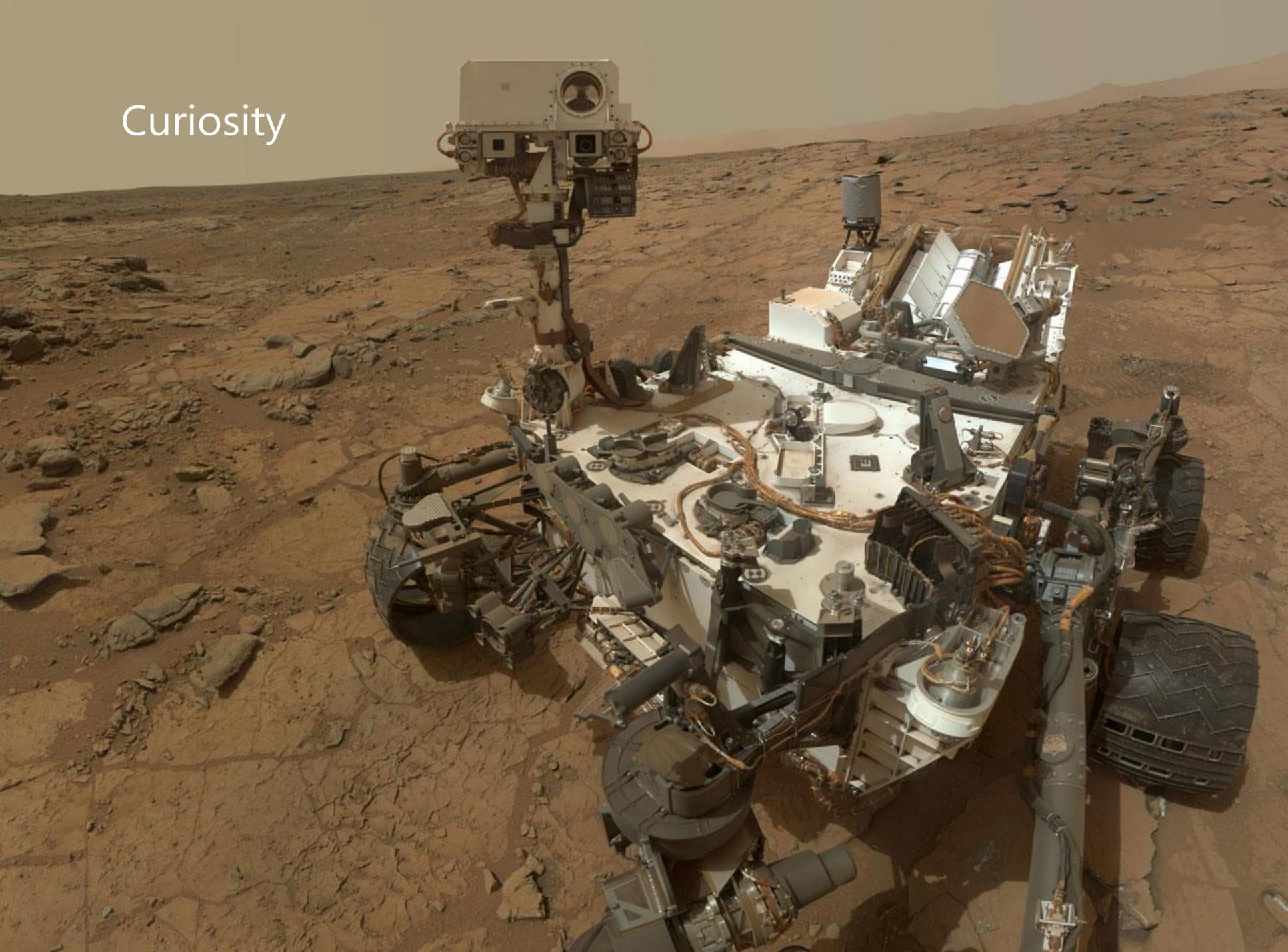


To answer these questions, we require robots like...

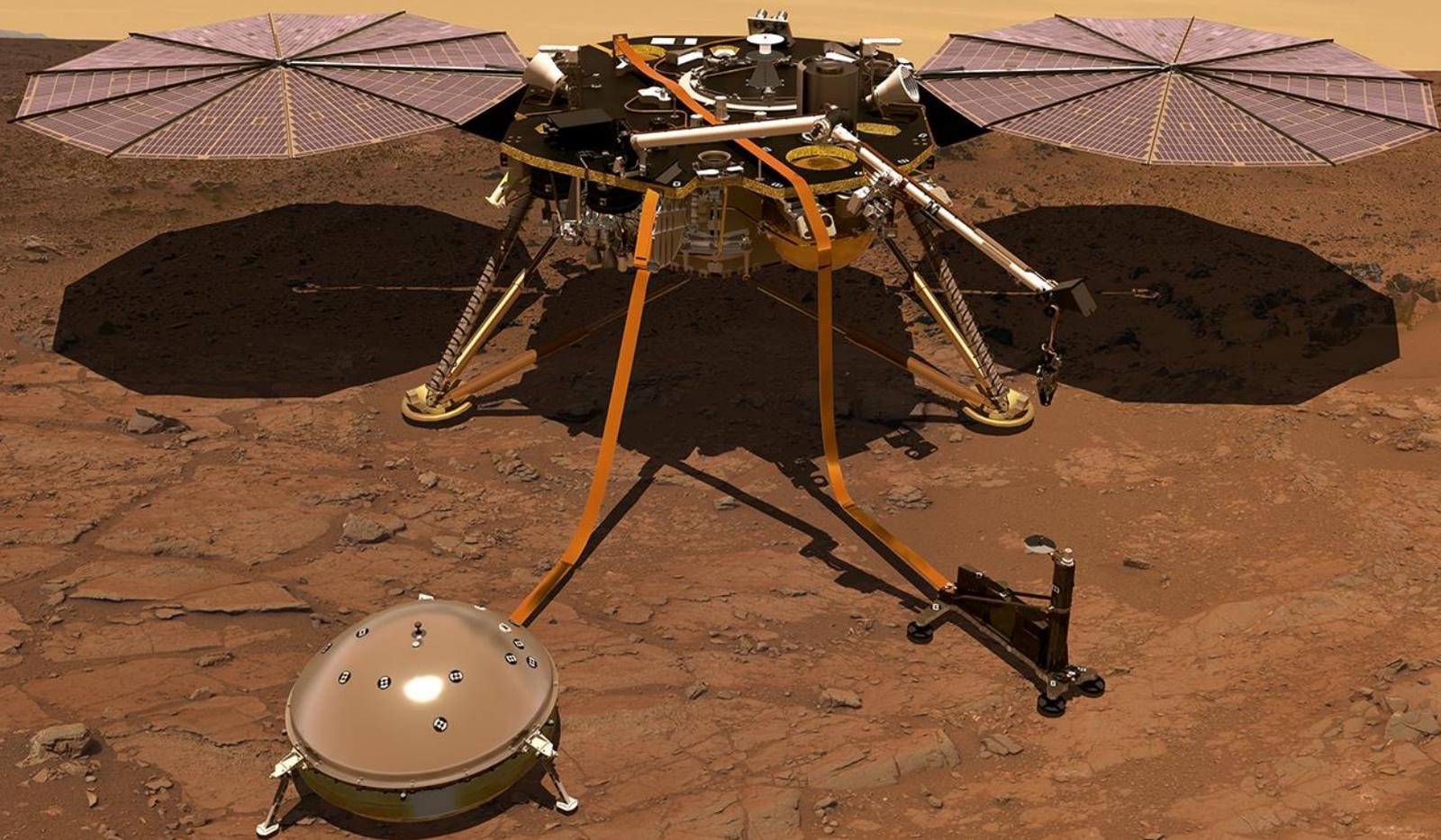


Juno

Curiosity



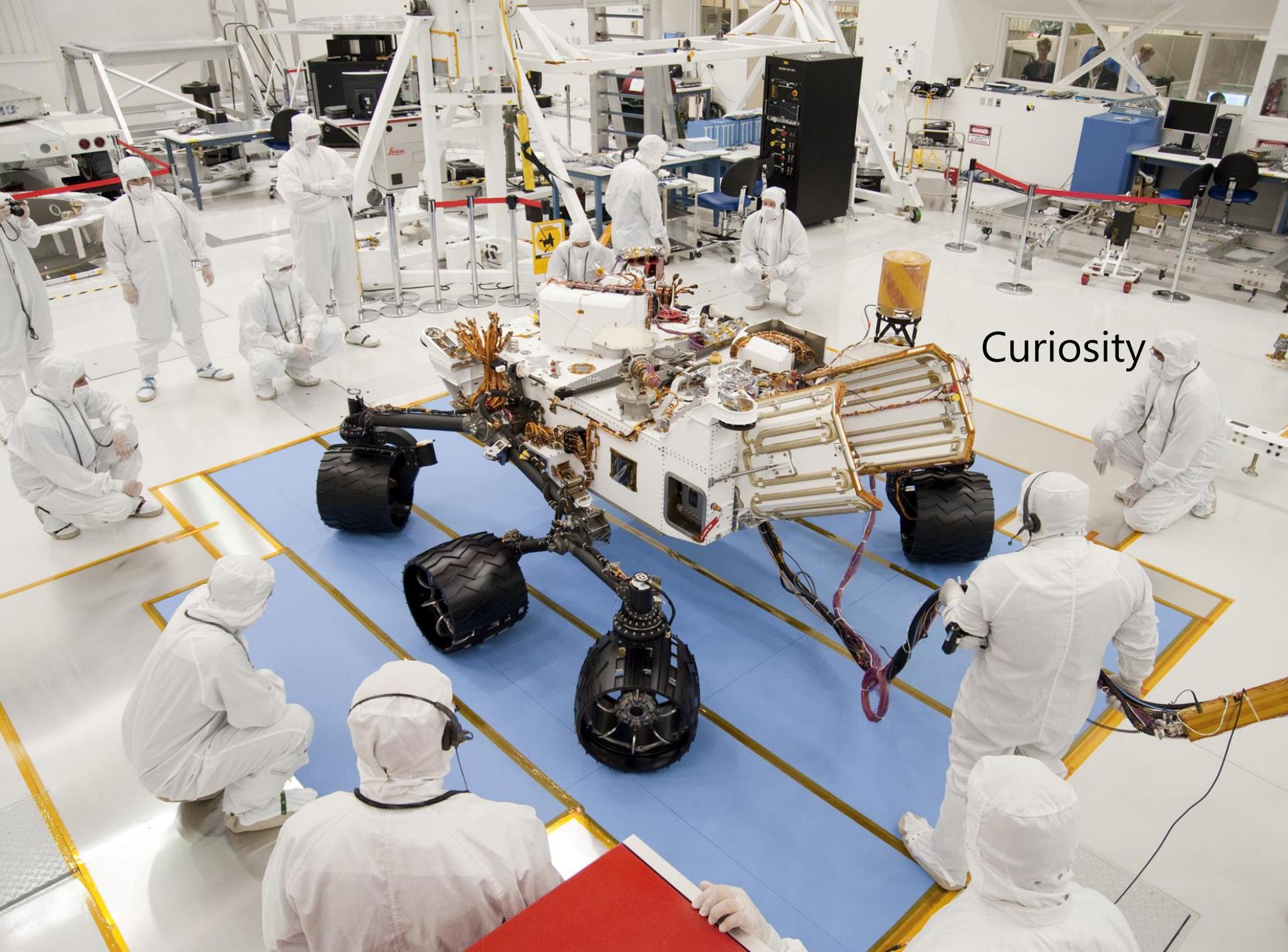
# Insight



To design, build, and operate these spacecraft requires an enormous human undertaking.

# Juno





Curiosity

And with that, the complete tracking of systems, failures, and anomalous behavior of spacecraft.

This requires some data... quite a bit of it!

# Charles Elachi Mission Control Center



# A Data Scientist's Snapshot of the Laboratory

- The Jet Propulsion Laboratory has existed since 1936.
- Now many data systems such as the Deep Space Network (DSN).
- Scope and volume of scientific data is large (NISAR will produce 3-5 TB daily).



A single DSN 70-meter radio antenna in Goldstone, CA

# DSN Now

Live stream of DSN telemetry streams



Jet Propulsion Laboratory | California Institute of Technology

## DEEP SPACE NETWORK NOW

LAST UPDATED: APR 18 9:16 PM (UTC)

[DSN home](#) 

### DAWN

 **MADRID**

APR 18  
11:16 PM

			
63	65	54	55

 **GOLDSTONE**

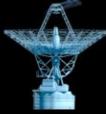
APR 18  
2:16 PM

				
14	15	24	25	26

**M010 MRO**      **TESS**      **M010 MRO MVN**      **TESS**

 **CANBERRA**

APR 19  
7:16 AM

			
43	34	35	36

TARGET

## DAWN



[VIEW ANTENNA](#)   [VIEW SPACECRAFT](#)   [VIEW WORLD MAP](#)

DAWN

SPACECRAFT

NAME  
Dawn

RANGE  
329.05 million km

ROUND-TRIP LIGHT TIME  
36.58 minutes

[+ more detail](#)      [credits](#)   [contact us](#)

# A Data Scientist's Snapshot of the Laboratory

- Internal data historically less prevalent; data exists within organizations and groups.
- Considerable effort is being made to improve internal data collection and build data-driven services that can improve our internal processes and decision making.



a present-day view of JPL

# User Evaluation & Technology Infusion Office

- A collection of data scientists, cloud engineers, software developers, and data visualization gurus.
- Our broad role as a team is to infuse new technologies into the way we do things at JPL.



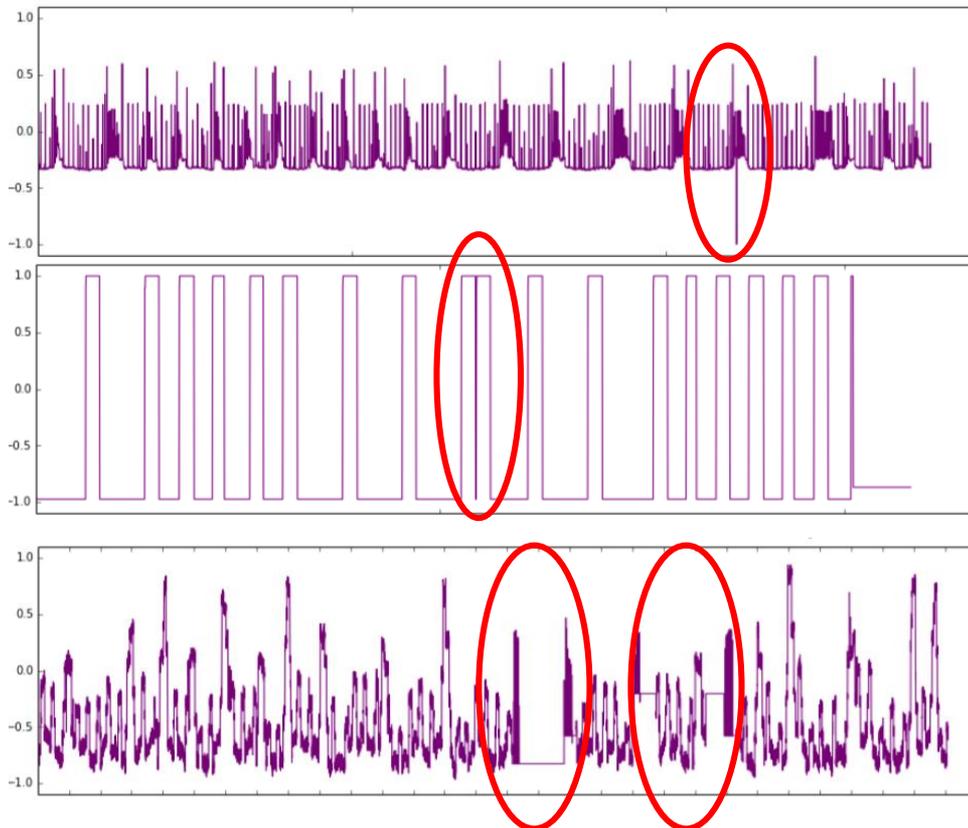
the office

We do some cool things!

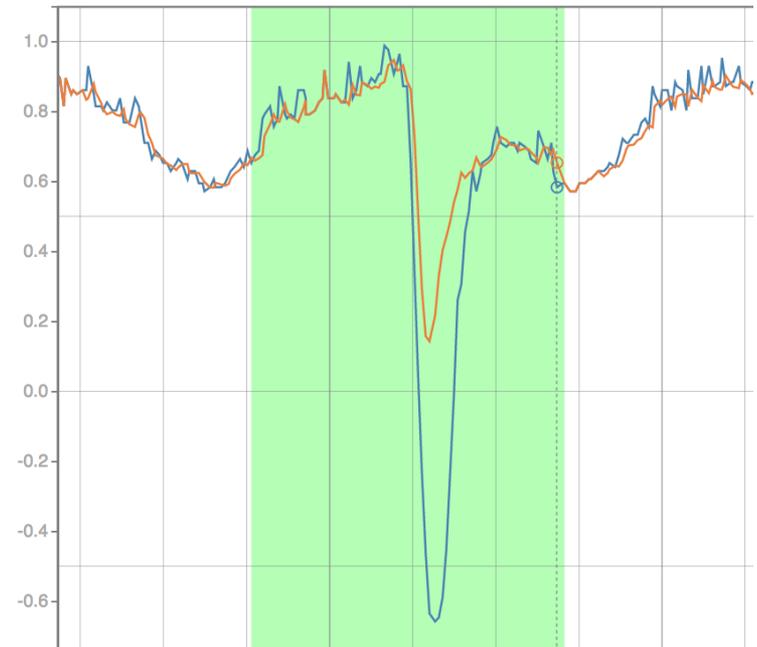
# Piloting the use of **LSTM networks to detect anomalous behavior** aboard spacecraft.

# LSTM Anomaly Detection

Aboard the SMAP spacecraft in Earth orbit



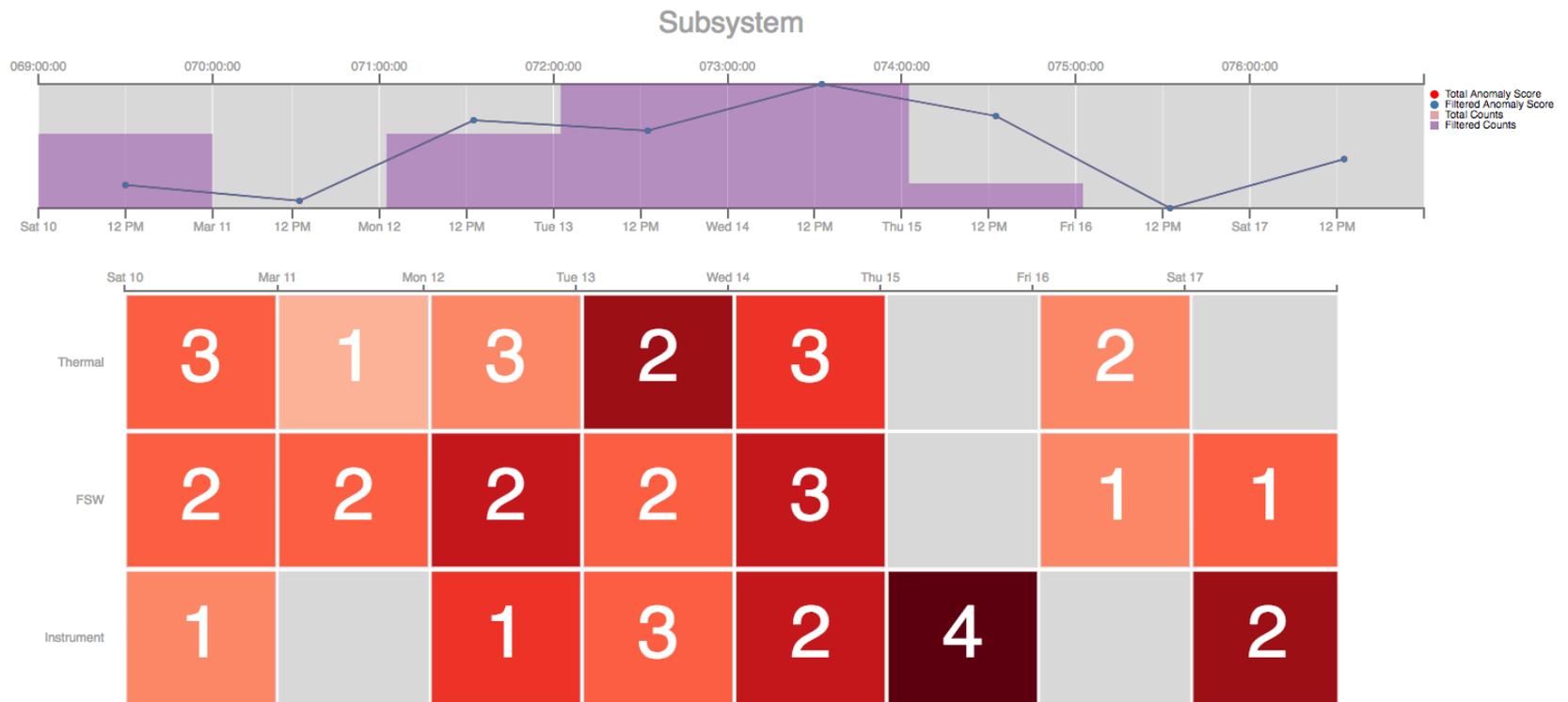
various types of anomalies (point, contextual, collective)



an anomalous event for a single spacecraft telemetry stream (channel)

# LSTM Anomaly Detection

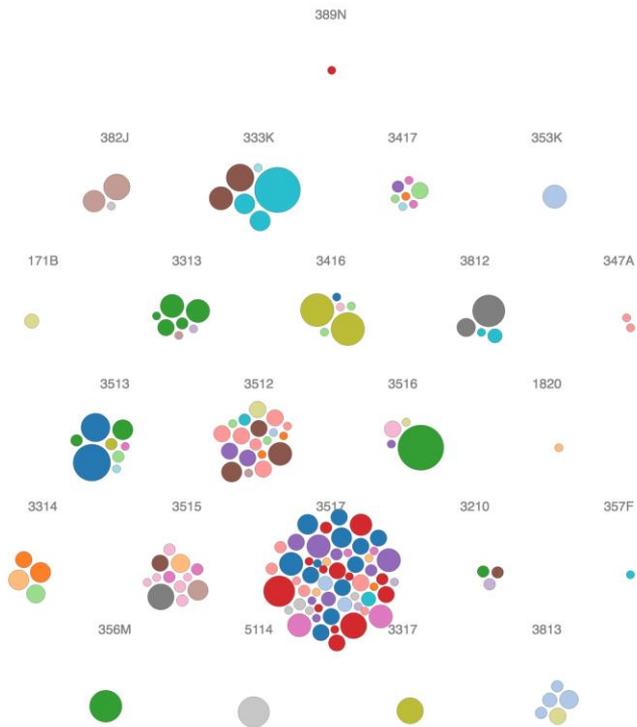
User-interface allows operator validation and exploration



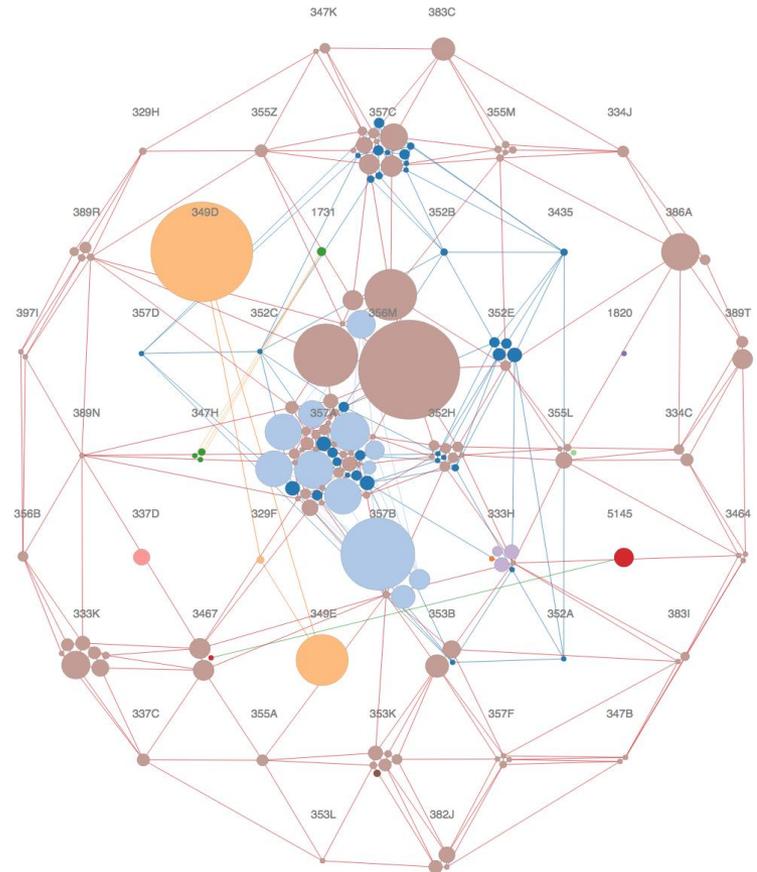
LSTM anomaly detection D3 user interface

Developing **data visualization and exploration interfaces** that allow for user labeling of data.

# Interactive Visualization



a generalized real-time stream of information



user-driven relationship exploration

## **2. Recommendations Made Easier**

# What is a recommender system?

- Recommender systems seek to predict some value a user would give to a particular item.
  - e.g. ratings, items to purchase, etc.

# What is a recommender system?

- Recommender systems seek to predict some value a user would give to a particular item.
  - e.g. ratings, items to purchase, etc.
- Two broad approaches:
  - **Collaborative Filtering:** provide users recommendations based on their similarity to others.
  - **Content-Based Filtering:** provide users recommendations based on their previous values for an item.

# What is a recommender system?

- Recommender systems seek to predict some value a user would give to a particular item.
  - e.g. ratings, items to purchase, etc.
- Two broad approaches:
  - **Collaborative Filtering:** provide users recommendations based on their similarity to others.
  - **Content-Based Filtering:** provide users recommendations based on their previous values for an item.
- Recommender systems are everywhere... Here's a few well-known examples.

# Recommender Systems in Practice

Amazon, YouTube, and Netflix

- Amazon Online Store
  - **Frequently Bought Together:** recommends associated products through market basket analysis or similar means.
  - **Recommendations for You in [category]:** recommends individual products to you based on your past purchases and the purchases of others similar to you.

# Recommender Systems in Practice

Amazon, YouTube, and Netflix

- Amazon Online Store
  - **Frequently Bought Together:** recommends associated products through market basket analysis or similar means.
  - **Recommendations for You in [category]:** recommends individual products to you based on your past purchases and the purchases of others similar to you.
- YouTube
  - **Recommended Videos:** recommends video content based off your recent history.

# Recommender Systems in Practice

Amazon, YouTube, and Netflix

- Amazon Online Store
  - **Frequently Bought Together:** recommends associated products through market basket analysis or similar means.
  - **Recommendations for You in [category]:** recommends individual products to you based on your past purchases and the purchases of others similar to you.
- YouTube
  - **Recommended Videos:** recommends video content based off your recent history.
- Netflix
  - **Recommended Videos in [category]:** recommends video content within categories, such as *trending*.

# The Problem Failure Reporting System (PRS)

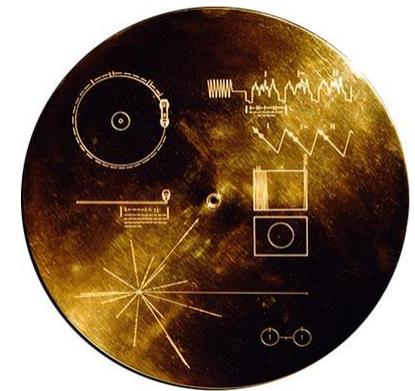
A time capsule of challenges and solutions

- Internal tool that allows engineers to submit Problem Failure Reports (PFRs) and Incident Surprise, Anomaly reports (ISAs).
  - 1. Document pre-launch test failures and post-launch operational anomalies experienced by spacecraft.
  - 2. Serve as both a record of past problems and of past solutions to the problems described.

# The Problem Failure Reporting System (PRS)

A time capsule of challenges and solutions

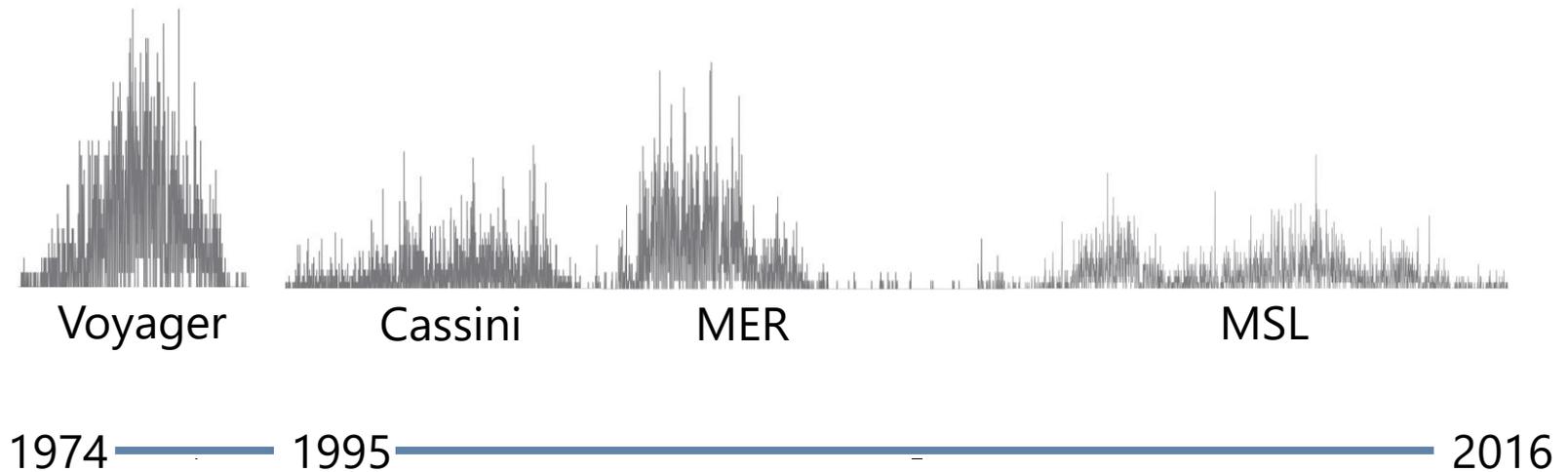
- Internal tool that allows engineers to submit Problem Failure Reports (PFRs) and Incident Surprise, Anomaly reports (ISAs).
  - 1. Document pre-launch test failures and post-launch operational anomalies experienced by spacecraft.
  - 2. Serve as both a record of past problems and of past solutions to the problems described.
- Reports offer **complete record of JPL spacecraft anomalies**, characteristics, and solutions spanning more than 40 years.



Voyager Golden Record

# The Problem Failure Reporting System (PRS)

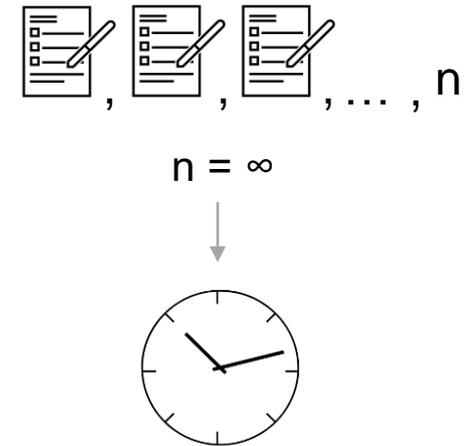
Problem Failure Reports (PFRs) over the years



# The Problem Failure Reporting System (PRS)

Important data but very time-consuming to populate

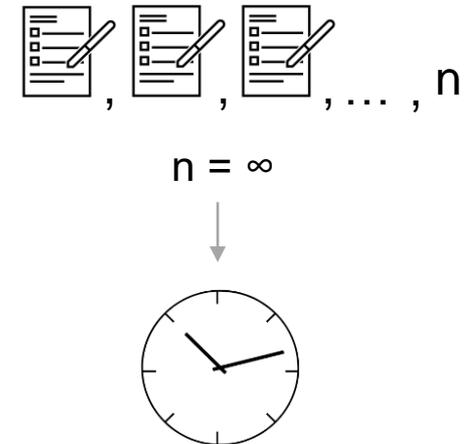
- Submitting reports is a time-consuming process.
- PRS forms contain over 40 fields, most of which are categorical. Some fields contain up to more than 50 classes.
- Data is very imbalanced.



# The Problem Failure Reporting System (PRS)

Important data but very time-consuming to populate

- Submitting reports is a time-consuming process.
- PRS forms contain over 40 fields, most of which are categorical. Some fields contain up to more than 50 classes.
- Data is very imbalanced.
- Goal is to both improve the user experience and reduce the time needed to fill out PRS reports.
- Our solution is to *provide form field recommendations to users dynamically as they populate the form.*



# What Does the Solution “Look Like”

A simple example

## Title

e.g. CSSR Prof Test Failure

---

## Description

e.g. During post proof test disassembly of the CSSR, the set screws...

---

## Project Name

---

## Specific Environment

---

## Problem Failure Noted During

---

## Problem Type

---

# What Does the Solution “Look Like”

## Overview

- *Henosis* is a cloud-based, open-sourced Python package that provides recommendations in a flexible and extensible framework.

# What Does the Solution “Look Like”

## Overview

- *Henosis* is a cloud-based, open-sourced Python package that provides recommendations in a flexible and extensible framework.
- Two broad development goals:
  - **Generalizability**
  - **Ease of Use** (user-driven design)

# What Does the Solution “Look Like”

## Overview

- *Henosis* is a cloud-based, open-sourced Python package that provides recommendations in a flexible and extensible framework.
- Two broad development goals:
  - **Generalizability**
  - **Ease of Use** (user-driven design)
- Open sourcing and generalizability were important goals from early on in the project.
  - Goal is to integrate this capability to other lab applications.
  - Our role to the public as an FFRDC is important → unique opportunity to promote open source tools

# What Does the Solution “Look Like”

## Overview

- *Henosis* is a cloud-based, open-sourced Python package that provides recommendations in a flexible and extensible framework.
  - Two broad development goals:
    - **Generalizability**
    - **Ease of Use** (user-driven design)
- 
- Open sourcing and generalizability were important goals from early on in the project.
    - Goal is to integrate this capability to other lab applications.
    - Our role to the public as an FFRDC is important → unique opportunity to promote open source tools

# What is scikit-learn?

- A collection of simple efficient tools for data mining, modeling, and analysis.



# What is scikit-learn?

- A collection of simple efficient tools for data mining, modeling, and analysis.
- Accessible to everyone that uses Python for data science.



# What is scikit-learn?

- A collection of simple efficient tools for data mining, modeling, and analysis.
- Accessible to everyone that uses Python for data science.
- Open source, community-maintained, easy to use, and popular.
  - Lots of available coding examples.
  - Core members actively maintain and update code-base.
  - Integrates easily into many existing Python workflows.



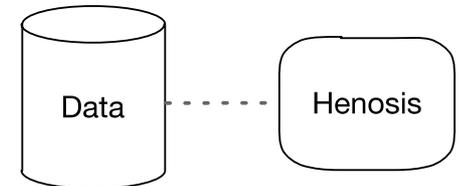
# Generalizable & Flexible

- Framework can be used with any *categorical* scikit-learn predictive model.



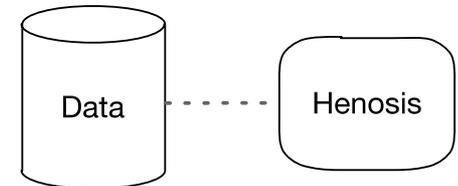
# Generalizable & Flexible

- Framework can be used with any *categorical* scikit-learn predictive model.
- Agnostic to data source → data sources reside outside the system.
  - “function tagging” allows us to bring in outside data or prep data → explain in later slides



# Generalizable & Flexible

- Framework can be used with any *categorical* scikit-learn predictive model.
- Agnostic to data source → data sources reside outside the system.
  - “function tagging” allows us to bring in outside data or prep data → explain in later slides
- Use of Elasticsearch and Amazon S3 allows for deployment at scale, use of cloud services.
  - *Both are required to use Henosis → other options for future.*



# Easy to Use

- User-Driven Development (UDD) was our approach from the start.

```
from Henosis.model import Data, Models  
from Henosis.server import Server
```

clean and easy imports

# Easy to Use

- User-Driven Development (UDD) was our approach from the start.
- Goal was to allow easy use by both data scientists and developers querying the framework for recommendations.

```
from Henosis.model import Data, Models  
from Henosis.server import Server
```

clean and easy imports

# Easy to Use

- User-Driven Development (UDD) was our approach from the start.
- Goal was to allow easy use by both data scientists and developers querying the framework for recommendations.
  - Developers simply query endpoint for recommendations. Different applications → unique but similar endpoints
    - System knows what to recommend through “missing tag”
  - Data scientists only call Henosis for splitting data or working with models
    - Models defined exactly as in scikit-learn

```
from Henosis.model import Data, Models
from Henosis.server import Server
```

clean and easy imports

<https://<host>/api/v0.1/recommend>

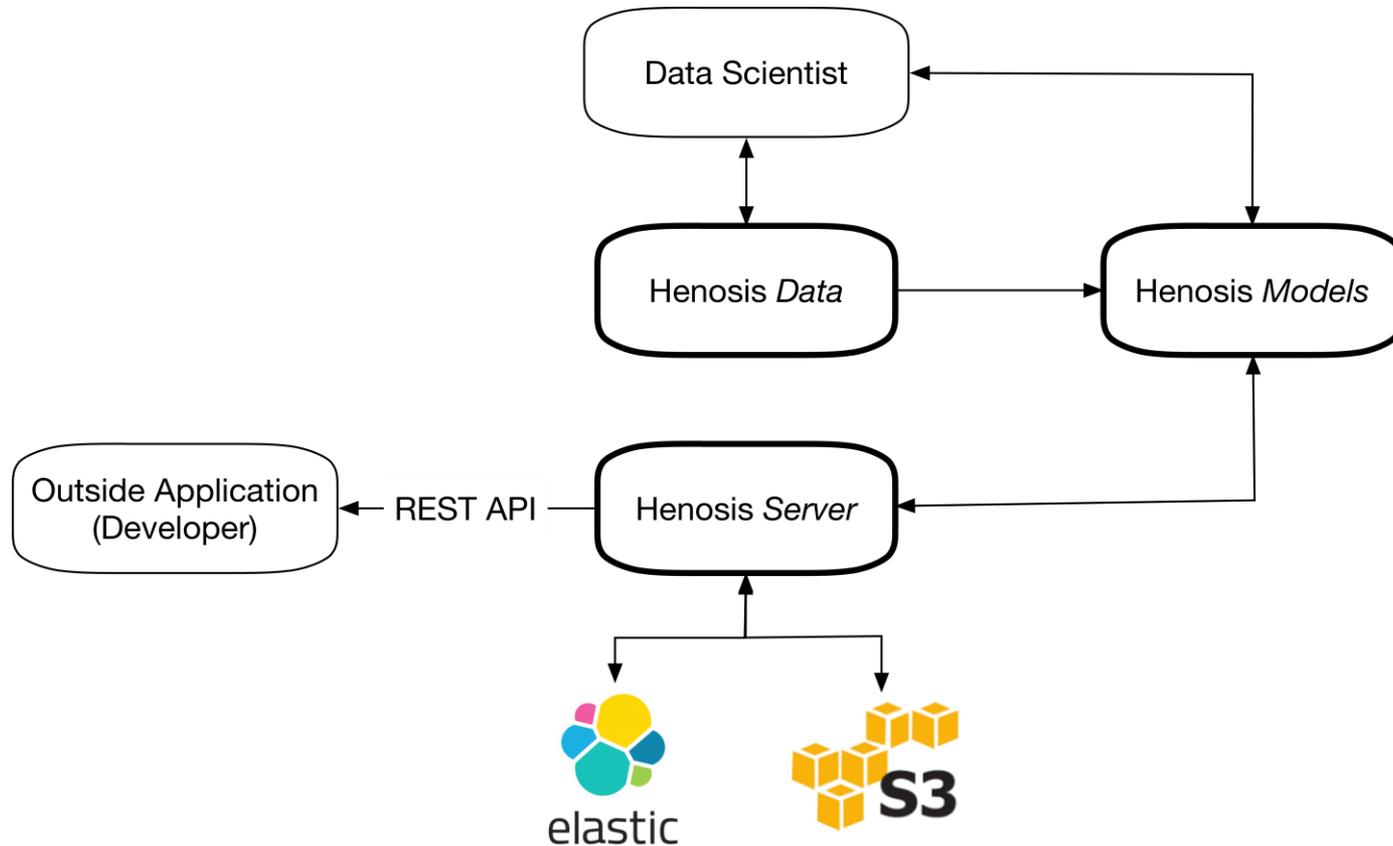
example endpoint

{title: “ODSC”, difficulty: 999}

example request data

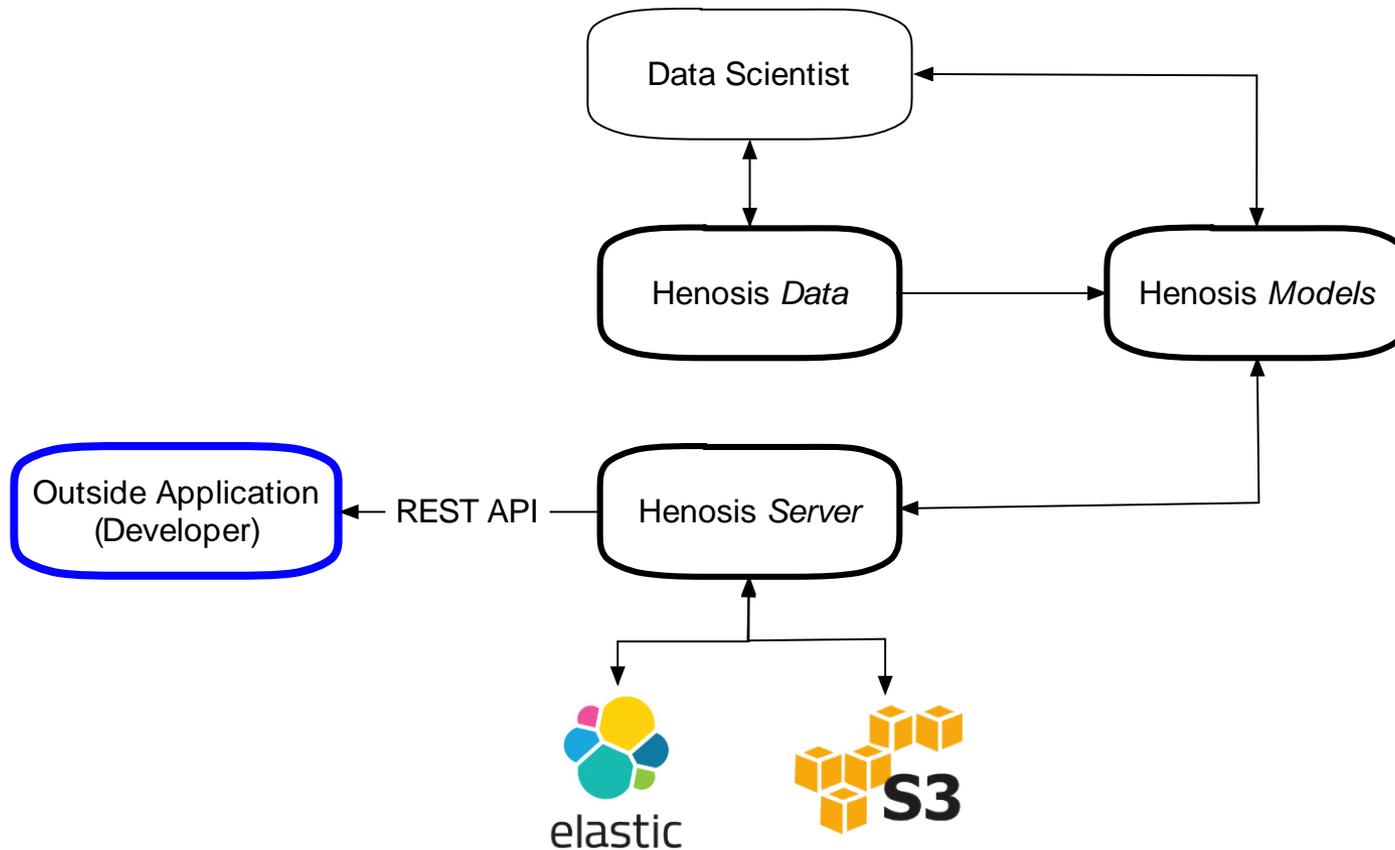
# What Does the Solution “Look Like”

Framework architecture



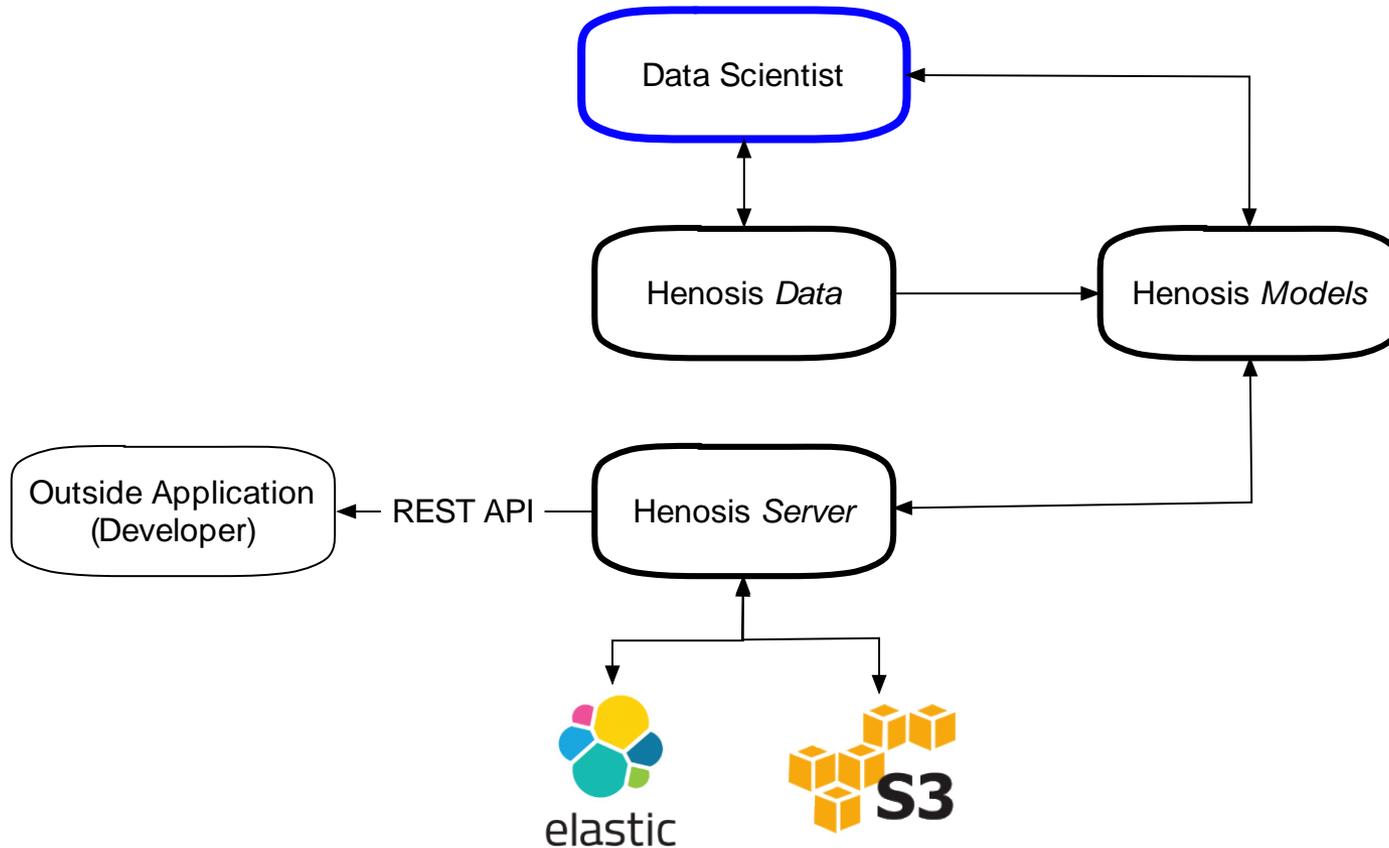
# What Does the Solution “Look Like”

Two broad user groups



# What Does the Solution “Look Like”

Two broad user groups



# Useful Features

- 1. Top-N Recommendations**
- 2. Confidence Thresholding**
- 3. Averaging Model Predictions**
- 4. Bagging (bootstrap aggregation)**
- 5. Built-in Server and API**

# Useful Features: Top-N Recommendations

- Providing only the most likely class for a given categorical field limits usability in some use cases.
- Bucketing the top  $n$  predictions decreases the likelihood of providing incorrect recommendations → increases likelihood of appropriate response
- Bucketing also allows for a little bit more “slack” for tough modeling problems.

[ 'In Doors', 'Cryogenic Testing', 'Orbit' ]  
[ 0.35 , 0.1 , 0.030 ]

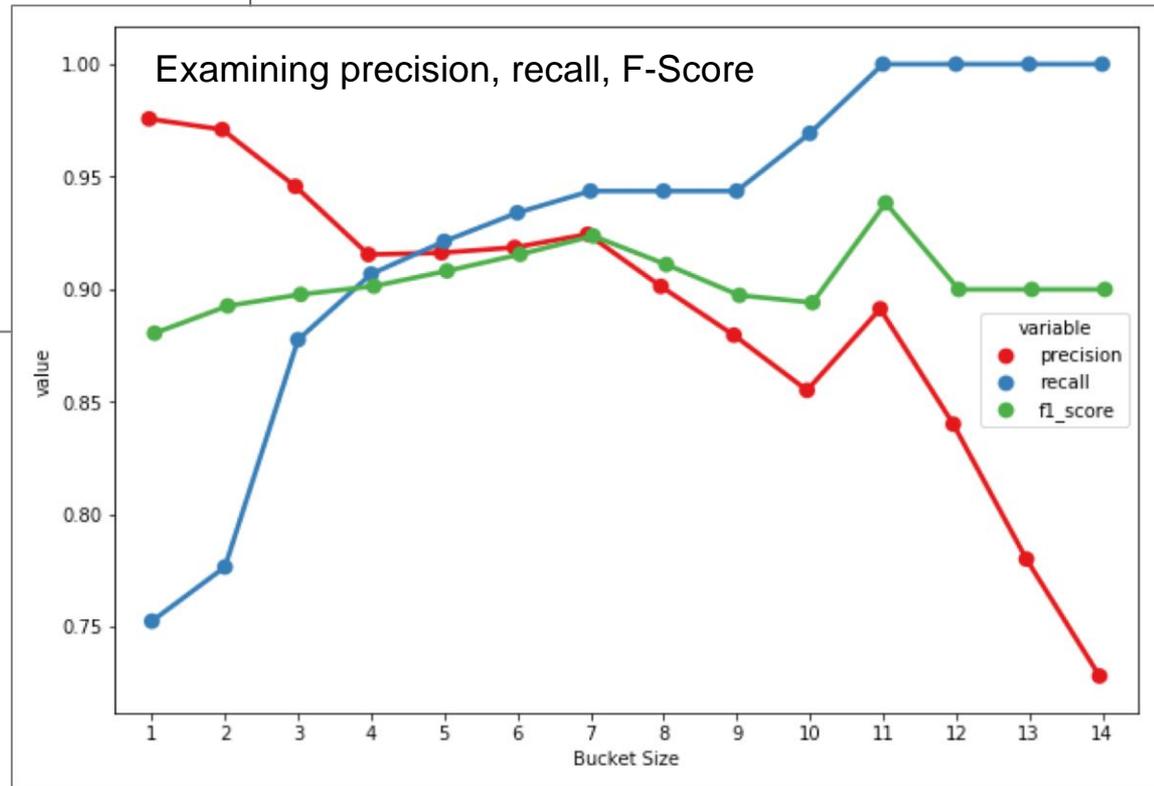
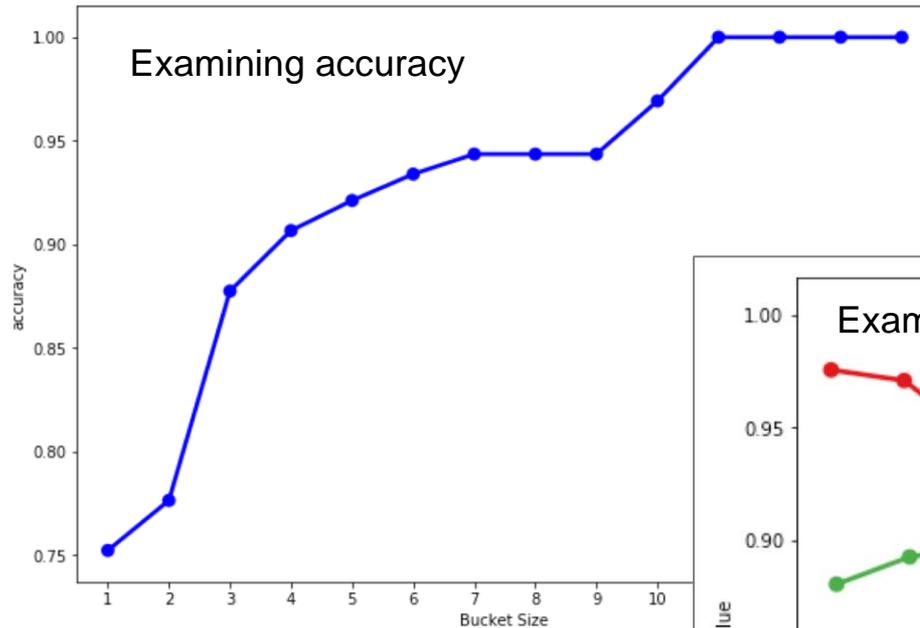
Top-N =	1	2	3
Likelihood =	0.35	+ 0.45	+ 0.48

## Specific Environment

---

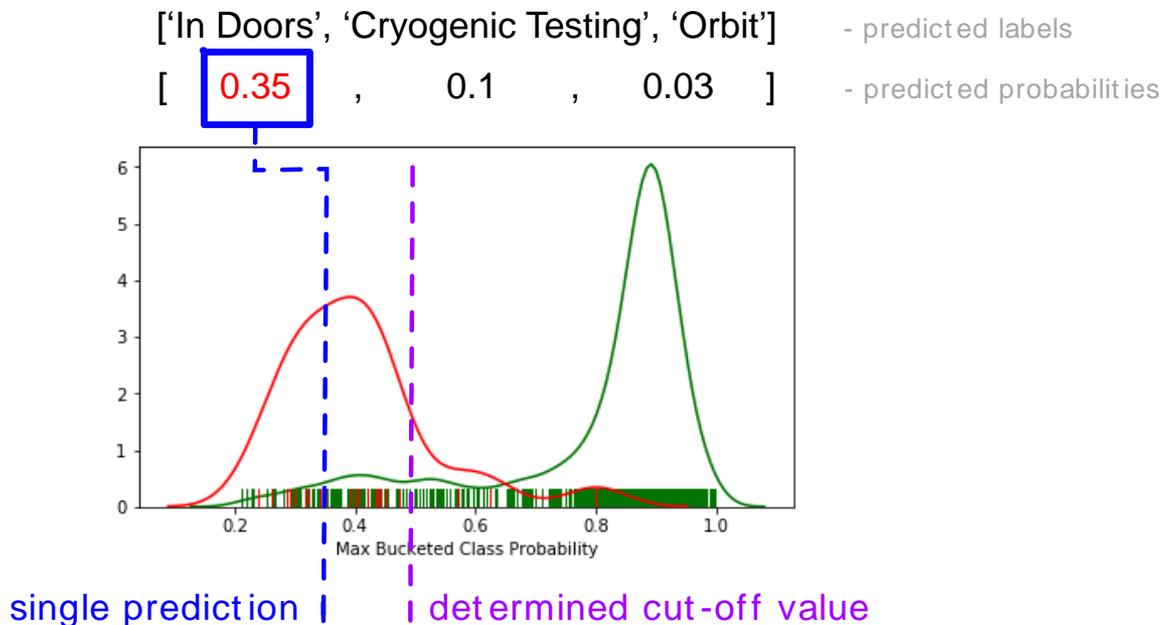
Recommended: Ambient - Clean Room; Ambient - In Doors; Low Temp, Vacuum;

# Useful Features: Top-N Recommendations



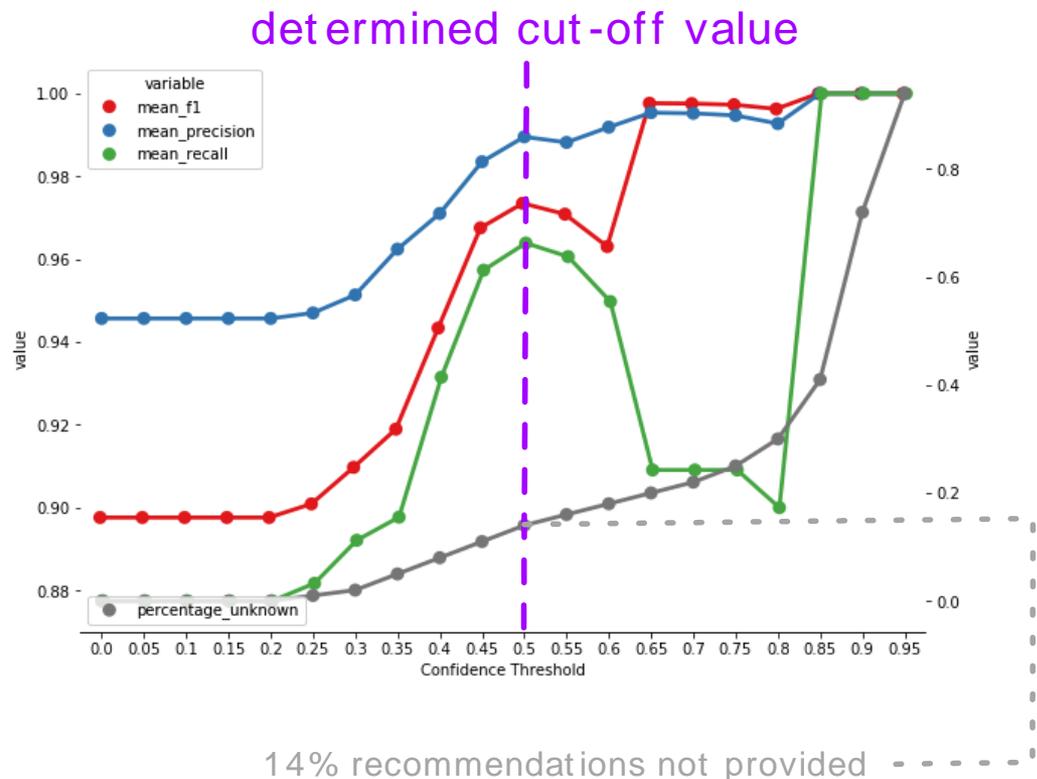
# Useful Features: Confidence Thresholding

- Idea is to provide recommendations only when confident of the result.
- Examine max bucketed class probability between true and false recommendations.



# Useful Features: Confidence Thresholding

- How do precision, recall, F-score improve as the percentage of withheld recommendations increases?
- Easy to specify → threshold is an optional parameter when deploying model.



# Useful Features: Averaging Model Predictions

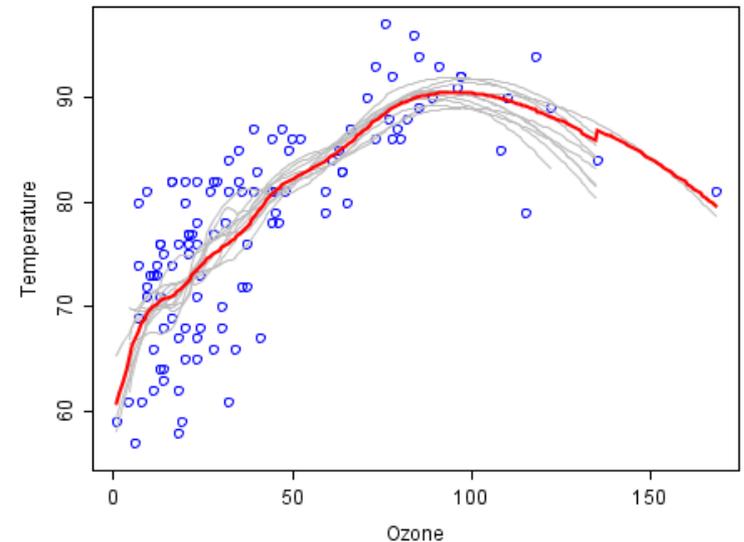
- Intuition: using multiple types of models better captures data distribution and relationships  
→ improves performance
- Henshin averages predicted probabilities for each class across models → provides ranked result
- Using more than one model per field allows for multiple model types, data sources.

$$\begin{array}{r} \text{['In Doors', 'Cryogenic Testing', 'Orbit']} \\ [ \text{0.35} , \text{0.1} , \text{0.030} ] \\ + \\ \text{['In Doors', 'Cryogenic Testing', 'Orbit']} \\ [ \text{0.25} , \text{0.85} , \text{0.10} ] \\ \div \\ \text{Mean} = \quad 0.3 \quad \boxed{\text{0.475}} \quad 0.065 \end{array}$$

# Useful Features: Bagging Model Predictions

(also known as bootstrap aggregation)

- Intuition: sampling data with replacement → reduces overfitting
- By randomly sampling before each model is fit, can easily achieve bagging.
- Bagging can improve performance when classes in the dependent variable are sparsely represented.

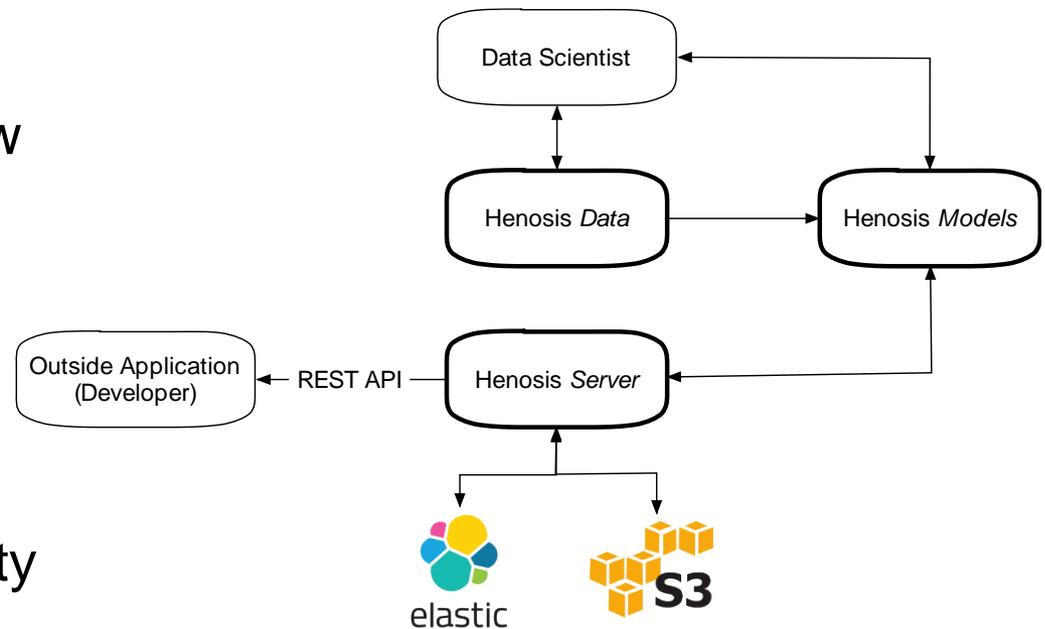


# Useful Features: Function Tagging

- What if I want to clean the text from a form field?
- What if I want to grab new data from somewhere besides the form for my models?

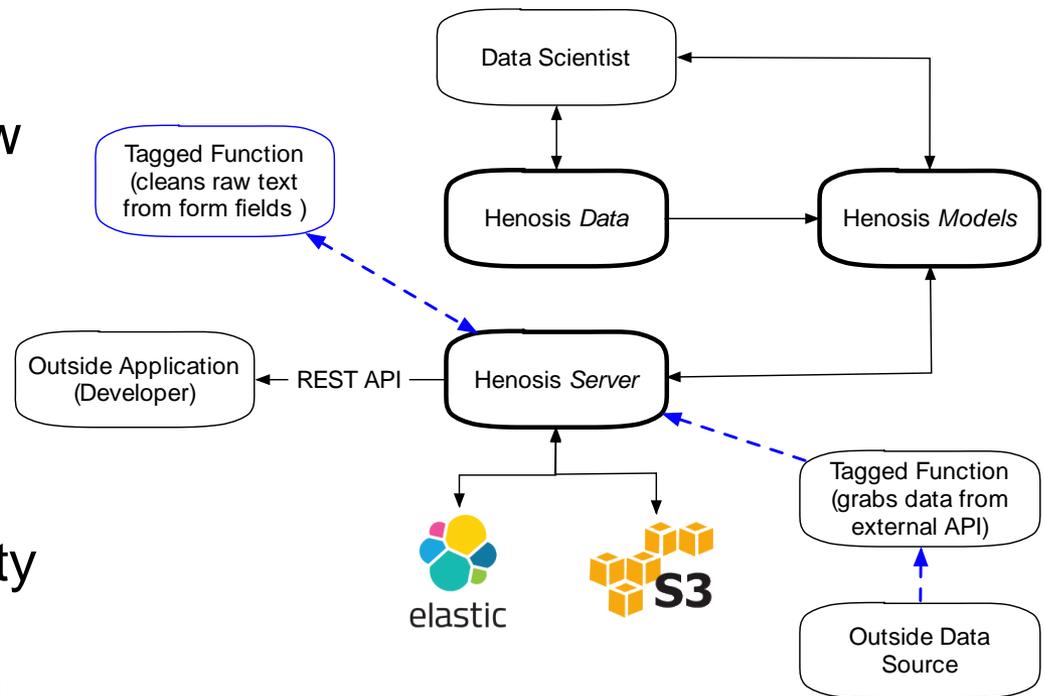
# Useful Features: Function Tagging

- What if I want to clean the text from a form field?
- What if I want to grab new data from somewhere besides the form for my models?
- “Function tagging” allows for near-full customizability of data sources, data cleaning procedures, and variable generation.



# Useful Features: Function Tagging

- What if I want to clean the text from a form field?
- What if I want to grab new data from somewhere besides the form for my models?
- “Function tagging” allows for near-full customizability of data sources, data cleaning procedures, and variable generation.



# Useful Features: Function Tagging

(some examples)

## *Text Cleaning*

1. Grab field “title” from API request.

## *Grab Data from an API*

1. Grab field “ID” from API request.

# Useful Features: Function Tagging

(some examples)

## *Text Cleaning*

1. Grab field “title” from API request.
2. Clean the text by removing stopwords, lemmatizing, etc.

## *Grab Data from an API*

1. Grab field “ID” from API request.
2. Using ID, query HR database for demographic information.

# Useful Features: Function Tagging

(some examples)

## *Text Cleaning*

1. Grab field “title” from API request.
2. Clean the text by removing stopwords, lemmatizing, etc.
3. Insert this cleaned text as a new field called “cleanText”.

## *Grab Data from an API*

1. Grab field “ID” from API request.
2. Using ID, query HR database for demographic information.
3. Insert collected demographic information as new fields (age, team, etc.).

# Useful Features: Function Tagging

(some examples)

## *Text Cleaning*

1. Grab field “title” from API request.
2. Clean the text by removing stopwords, lemmatizing, etc.
3. Insert this cleaned text as a new field called “cleanText”.
4. Deployed model uses “cleanText” to provide recommendations.

## *Grab Data from an API*

1. Grab field “ID” from API request.
2. Using ID, query HR database for demographic information.
3. Insert collected demographic information as new fields (age, team, etc.).
4. Deployed model uses demographic information to provide recommendations.

# Useful Features: Function Tagging

(some examples)

## *Text Cleaning*

1. Grab field “title” from API request.
2. Clean the text by removing stopwords, lemmatizing, etc.
3. Insert this cleaned text as a new field called “cleanText”.
4. Deployed model uses “cleanText” to provide recommendations.
5. Returned model-specific recommendation has used data from outside the original API request.

## *Grab Data from an API*

1. Grab field “ID” from API request.
2. Using ID, query HR database for demographic information.
3. Insert collected demographic information as new fields (age, team, etc.).
4. Deployed model uses demographic information to provide recommendations.
5. Returned model-specific recommendation has used data from outside the original API request.

# Useful Features: Built-In Server and API

- Henosis comes built in with a simple WSGI + Flask server for providing recommendations.
- API has endpoints:
  - /recommend (provides recommendations)
  - /models (provides model information)
  - /requestlog (provides logs from each request made to the system)
- Running the server simply consists of an import, specification of a configuration file, and calling function within the Server class.



# Useful Features: Built-In Server and API

/recommend

- This API endpoint is the “bread & butter” of the API.
- Query is a flat JSON file containing (key, value) pairs where:
  - **Key:** indicates variable name, should be the same as training set
  - **Value:** a raw value to use in making predictions
    - unique “missing tag” is used by system to determine which fields need recommendations

```
query                                     response
{                                         {
  'variableOne': '999999',                "variableOne": "Proton Beam",
  'variableTwo': 'Spaceman',              "variableTwo": "Spaceman",
  'variableThree': '999999',              "variableThree": "Liquid Nitrogen Resistant Gloves"
}
```

# Useful Features: Built-In Server and API

/models

- This endpoint returns information about models stored and deployed in the system.
- Can retrieve all models or only models that meet specific condition.
  - e.g. **deployed** = **true**

```
response
{
  "callCount": 30,
  "dependent": "variableTwo",
  "deployed": true,
  "encoderPath": "encoder_variableTwo_1.pickle",
  "encoderType": "CountVectorizer",
  "testF1": 0.9008452056839288,
  "testPrecision": 0.9044056750340173,
  "testRecall": 0.9019607843137255,
  "independent": [
    {
      "generator_path": "clean_text.pickle",
      "inputs": [
        "title"
      ],
      "name": "cleanText"
    }
  ],
}
```

# Useful Features: Built-In Server and API

/requestlog

- This endpoint returns information about models stored and deployed in the system.
- Can retrieve all request logs or only request logs that meet specific condition.
  - e.g. **responseStatusCode** = 200

```
response
{"sessionId": "a1c5b9a7f01d4f30a6cb13b2390fc2ac",
 "sessionExpireDate": "2018-02-07T10:51:12",
 "timeIn": "2018-02-21T05:42:50",
 "timeOut": "2018-02-21T05:42:52",
 "timeElapsed": 2.2941131591796875,
 "responseStatusCode": 200,
 "responseDescription": "200 OK",
 "recommendations": {
  "projectName": [
    "projectOne",
  ]
},
 "missingFields": [
  "projectName" ],
 "modelsQueried": [
  "be4d26d624a6408abd105b015e0526b3" ],
```

# Summary of Overviewed Features

- Henosis is a general, cloud-based recommender framework.  
Goal → improve the usability and ease of use of deploying recommender systems for categorical fields.

# Summary of Overviewed Features

- Henosis is a general, cloud-based recommender framework.  
Goal → improve the usability and ease of use of deploying recommender systems for categorical fields.
- Can bucket recommendations to provide users with several recommendations.
- Can threshold recommendations based on confidence, and fetch or process data from outside sources using tagged functions.
- Comes with WSGI + Flask server for providing recommendations.

# Summary of Overviewed Features

- Henosis is a general, cloud-based recommender framework.  
Goal → improve the usability and ease of use of deploying recommender systems for categorical fields.
- Can bucket recommendations to provide users with several recommendations.
- Can threshold recommendations based on confidence, and fetch or process data from outside sources using tagged functions.
- Comes with WSGI + Flask server for providing recommendations.
- Other useful features in system, such as (but not limited to):
  - preloading models into memory and periodically refreshing those models
  - the ability to provide a single recommendation through majority vote (no probabilities)
  - simple API authorization
  - ability to track sessions in the using the request log

# Henosis in Practice

## A simple modeling workflow example

```
from Henosis.model import Data, Models

# read in data from a csv
d = Data()
d.load(csv_path='file.csv')
print(d.all.head())

# split data
d.test_train_split(
    d.all[X_vars],
    d.all[y_var],
    share_train=0.8
)

# fit a model (use any categorical scikit-learn model)
# this is where you do your "magic"!
m = Models().SKModel(MultinomialNB(alpha=0.25))
m.train(d)
m.test(d)

# print results
print(m.train_results)
print(m.test_results)
```

```
from Henosis.utils import Connect
s = Connect().config(config_yaml_path='config.yaml')

# store the model in S3 and Elasticsearch
m.store(
    server_config=s,
    model_path='model_' + y_var + '_1.pickle',
    encoder_path='encoder_' + y_var + '_1.pickle',
    encoder=count_vect
)

# deploy a model for use
m.deploy(
    server_config=s,
    deploy=True
)
```

# Henosis in Practice

A simple server deployment example

```
from Henosis.server import Server

# run the server
s = Server().config(config_yaml_path='config.yaml')
s.run()
```

# Henosis in Practice

## The configuration file (config.yaml)

```
# Elasticsearch
elasticsearch:
  host: 'https://<host>:<port>'
  verify: False
  auth: !!python/tuple ['<username>', '<password>']
  models:
    index: '/model'
  request_log:
    index: '/requestlog'

# AWS S3
aws_s3:
  region: '<region>'
  bucket: '<bucket>'
  key: '<aws_key>'
  secret: '<aws_secret>'

# Api
api:
  host: 'http://localhost:5005'
  index: '/api'
  version: '/v0.1'
  missing: '999999'
  session_expiration: 10
  auth: !!python/tuple ['<username>', '<password>']

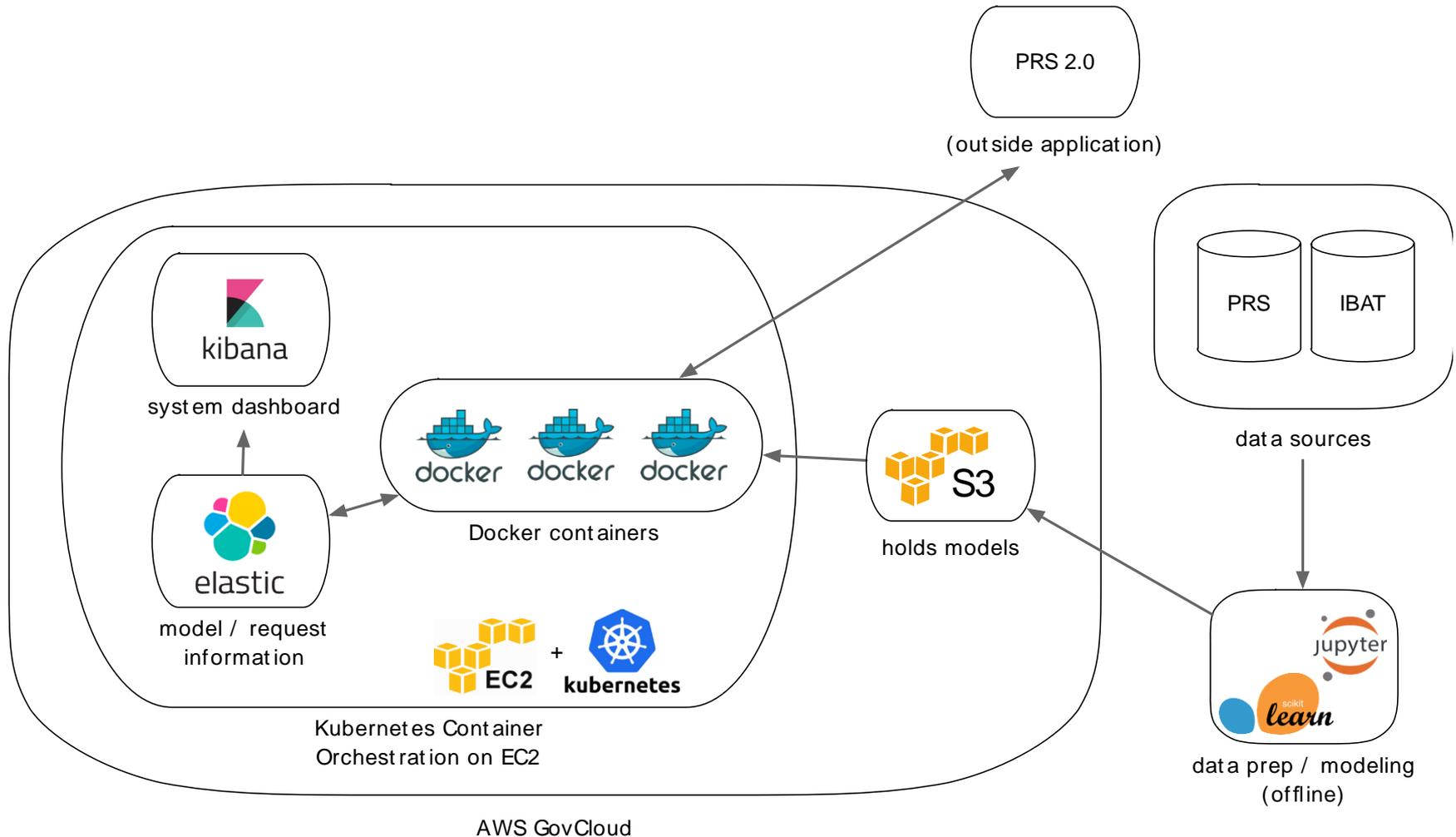
# Models
models:
  preload_pickles: True
  refresh_pickles: 1440 # in minutes
  predict_probabilities: True
  top_n: 3
```

# 3. Implementation & Proof of Concept

PRS 2.0 deployment architecture builds on top of the open-source framework.

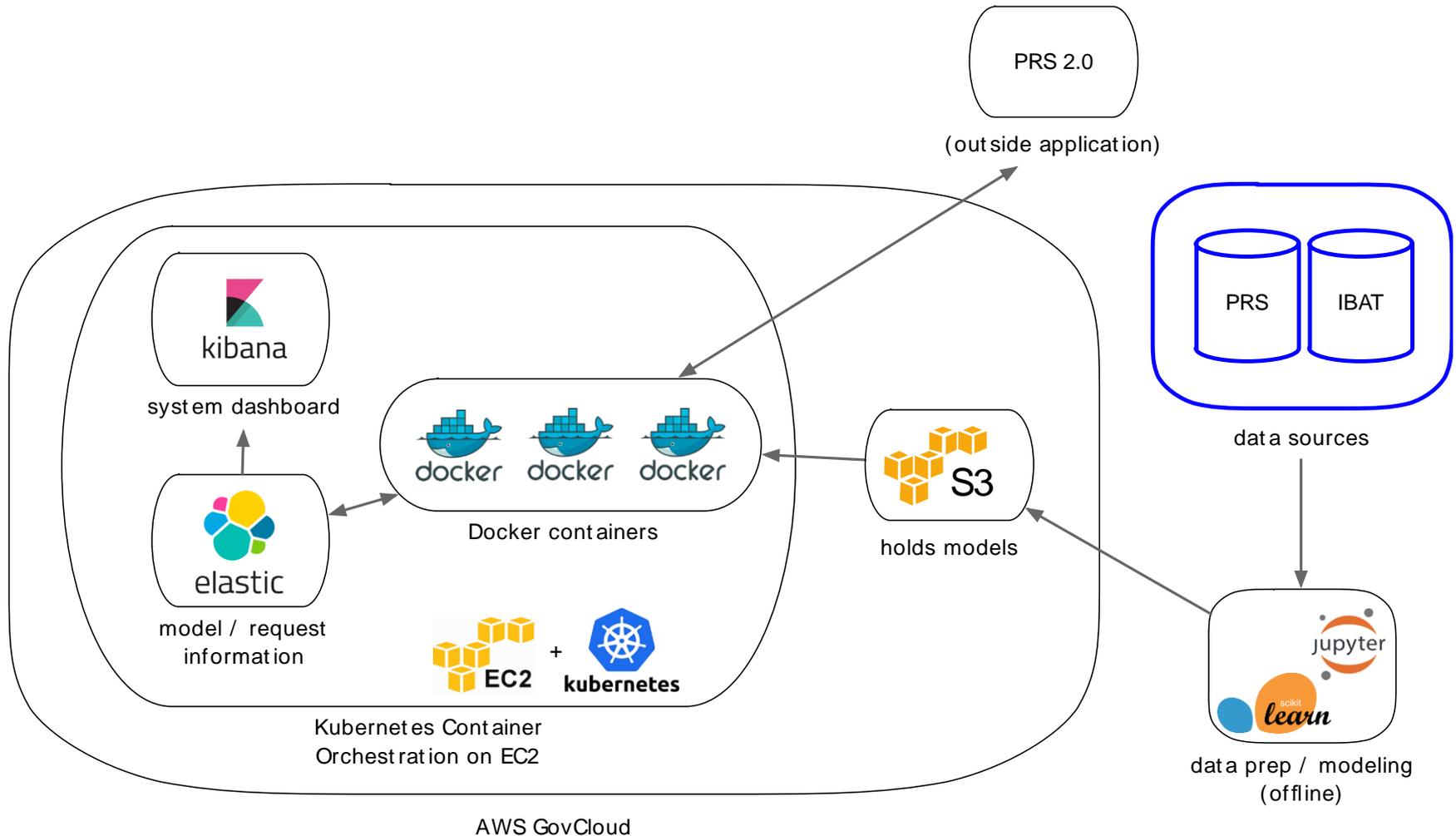
# The PRS 2.0 Architecture

The complete picture



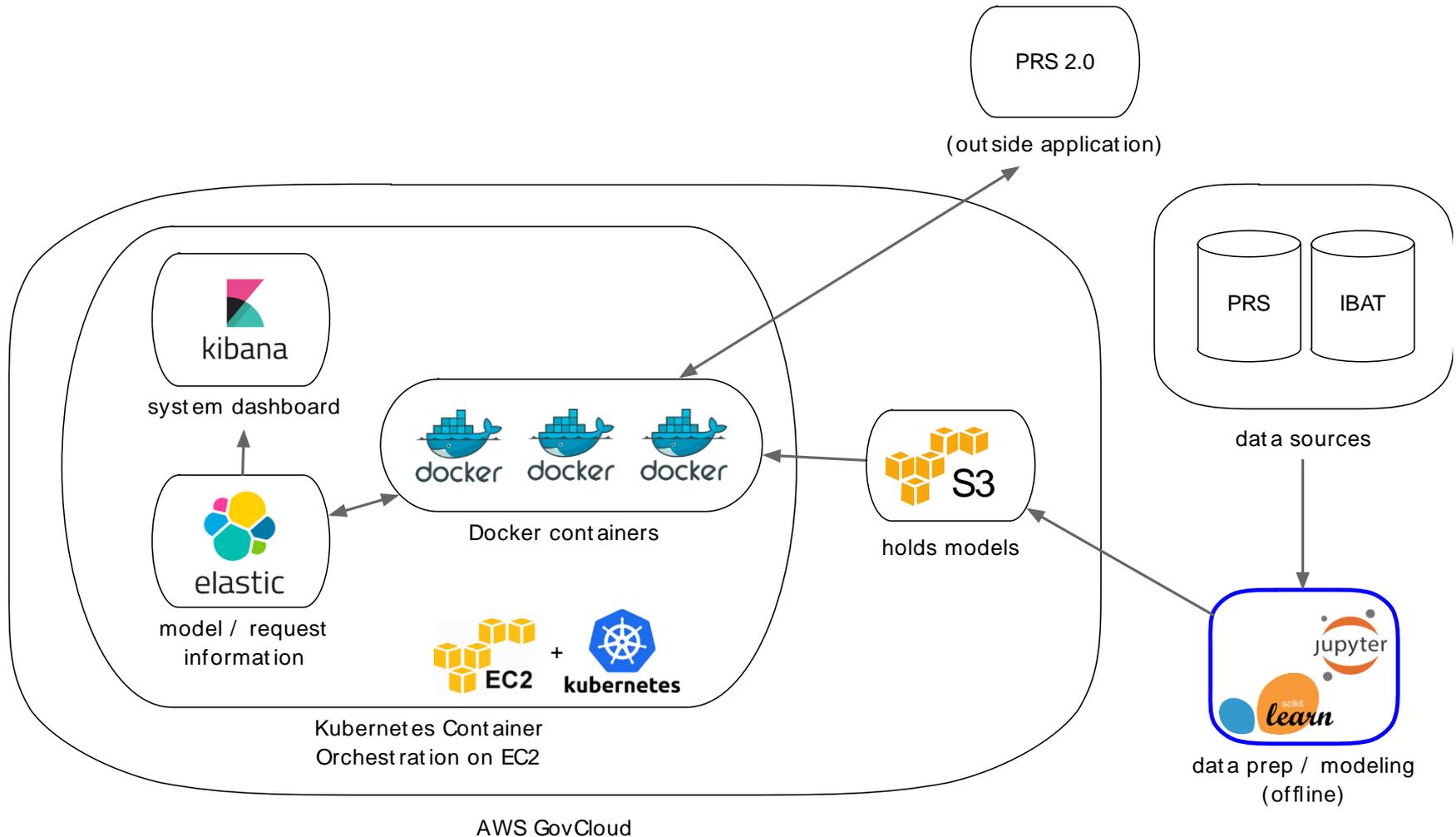
# The PRS 2.0 Architecture

Steps to deployment: grab data



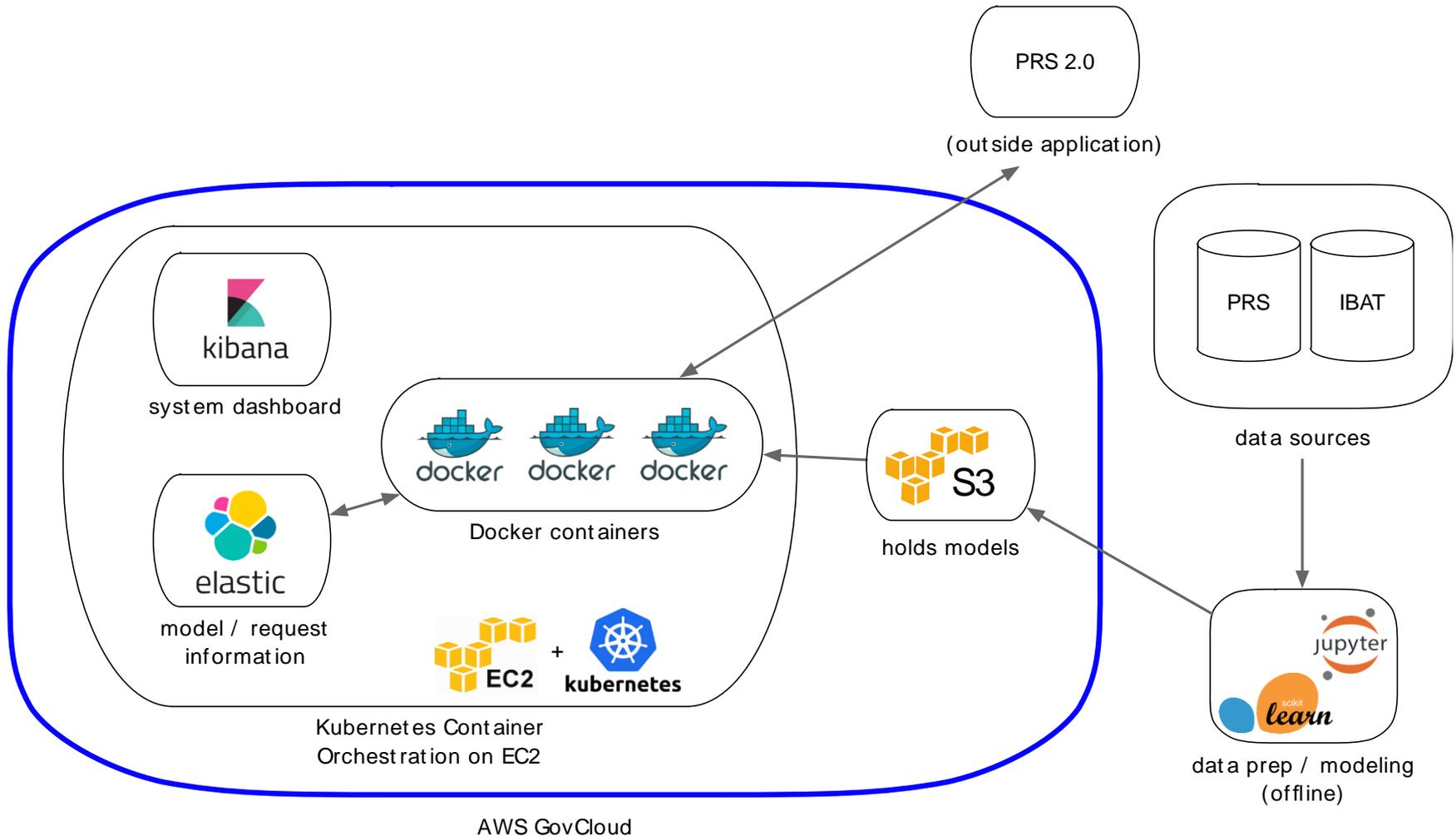
# The PRS 2.0 Architecture

Steps to deployment: work your magic!



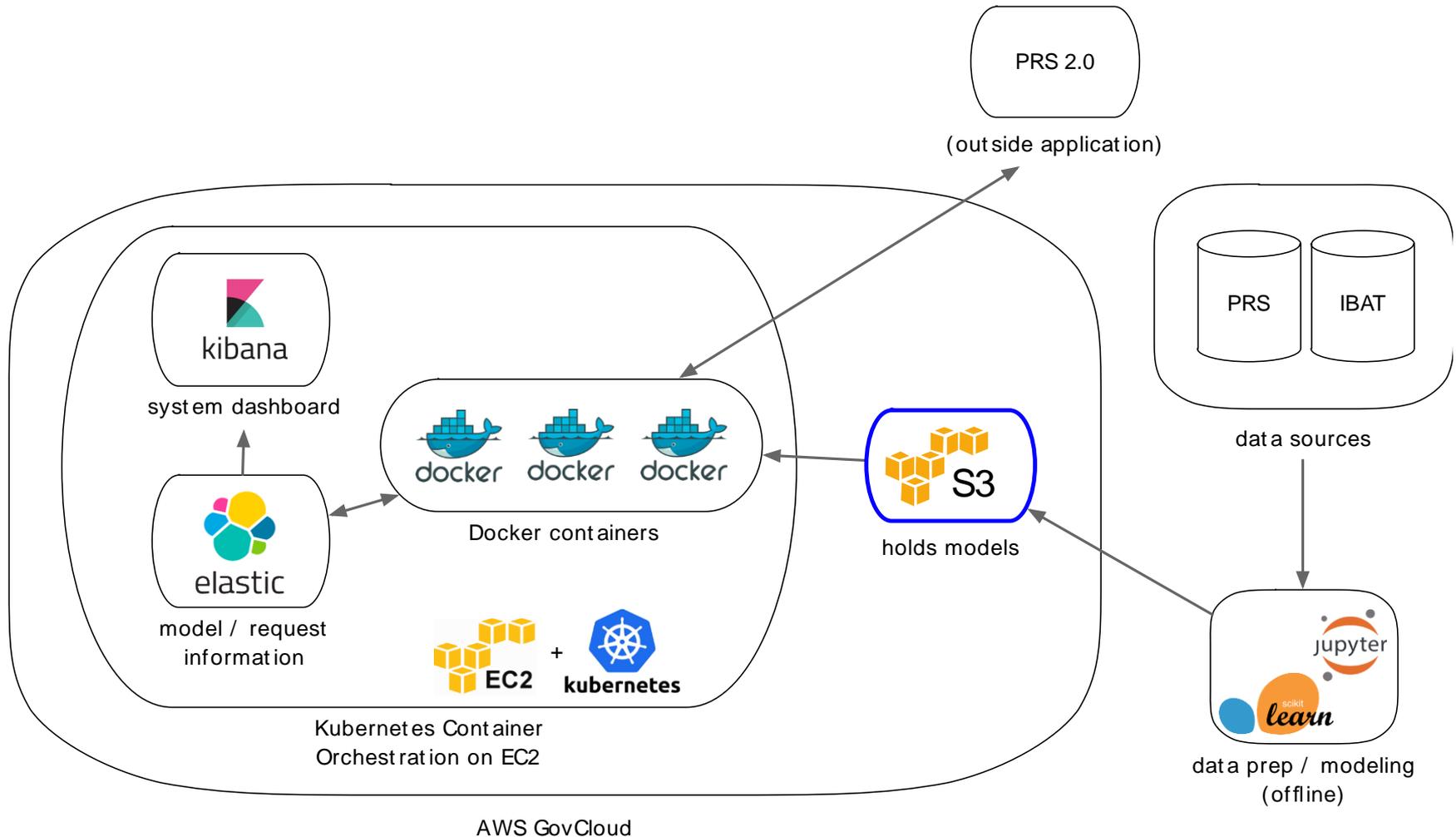
# The PRS 2.0 Architecture

Steps to deployment: push to cloud



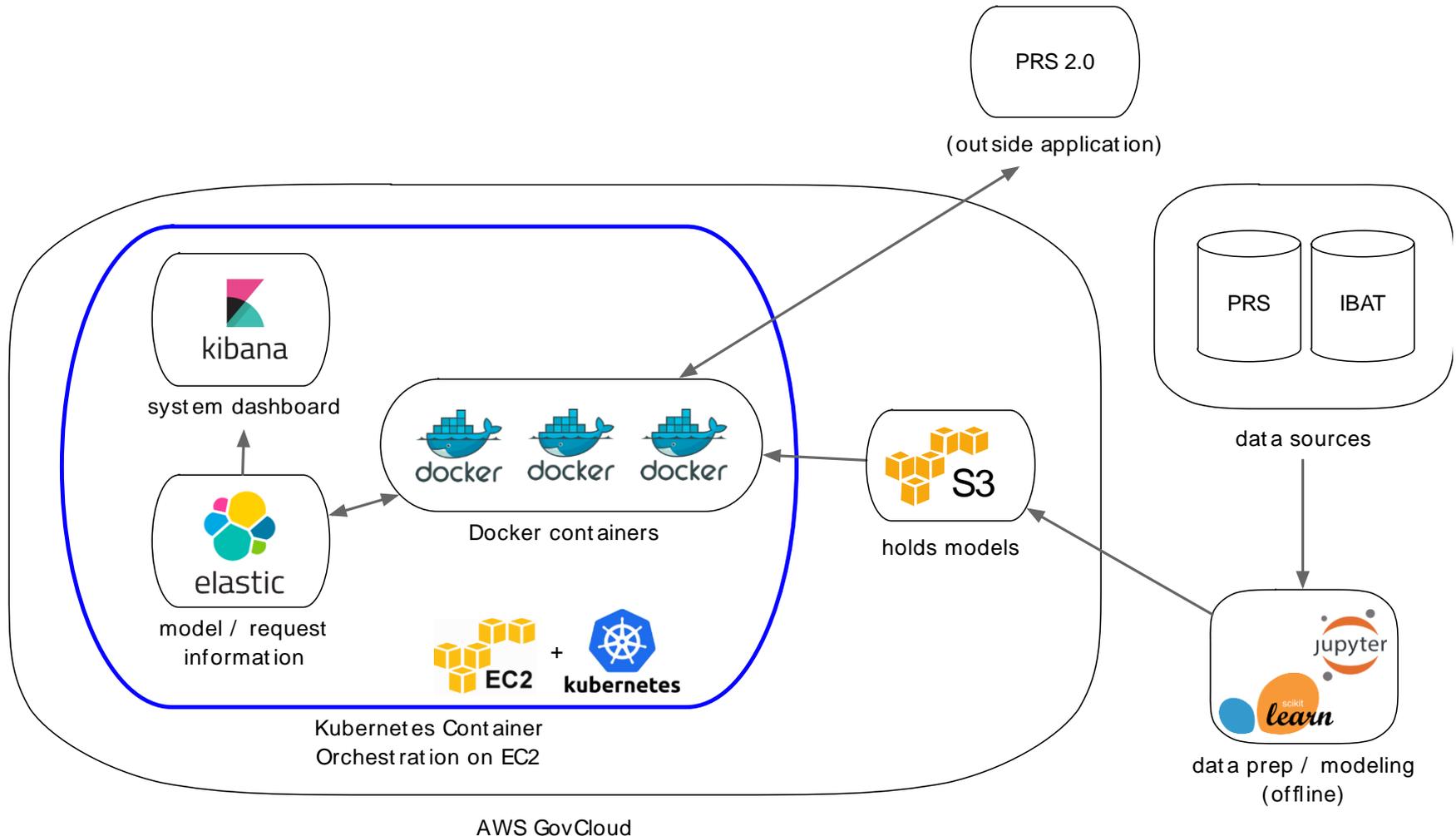
# The PRS 2.0 Architecture

Steps to deployment: store models



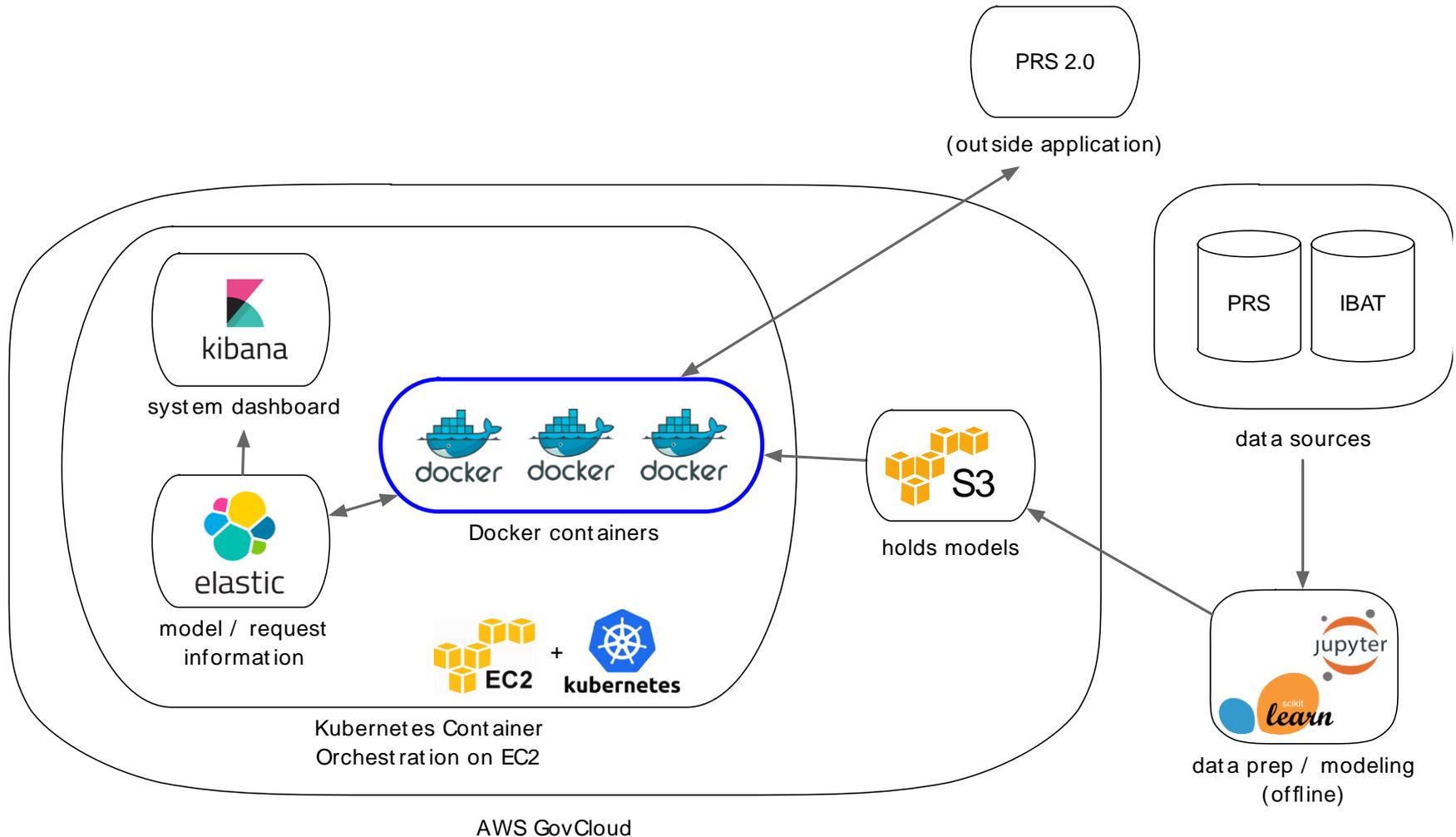
# The PRS 2.0 Architecture

Steps to deployment: containerize



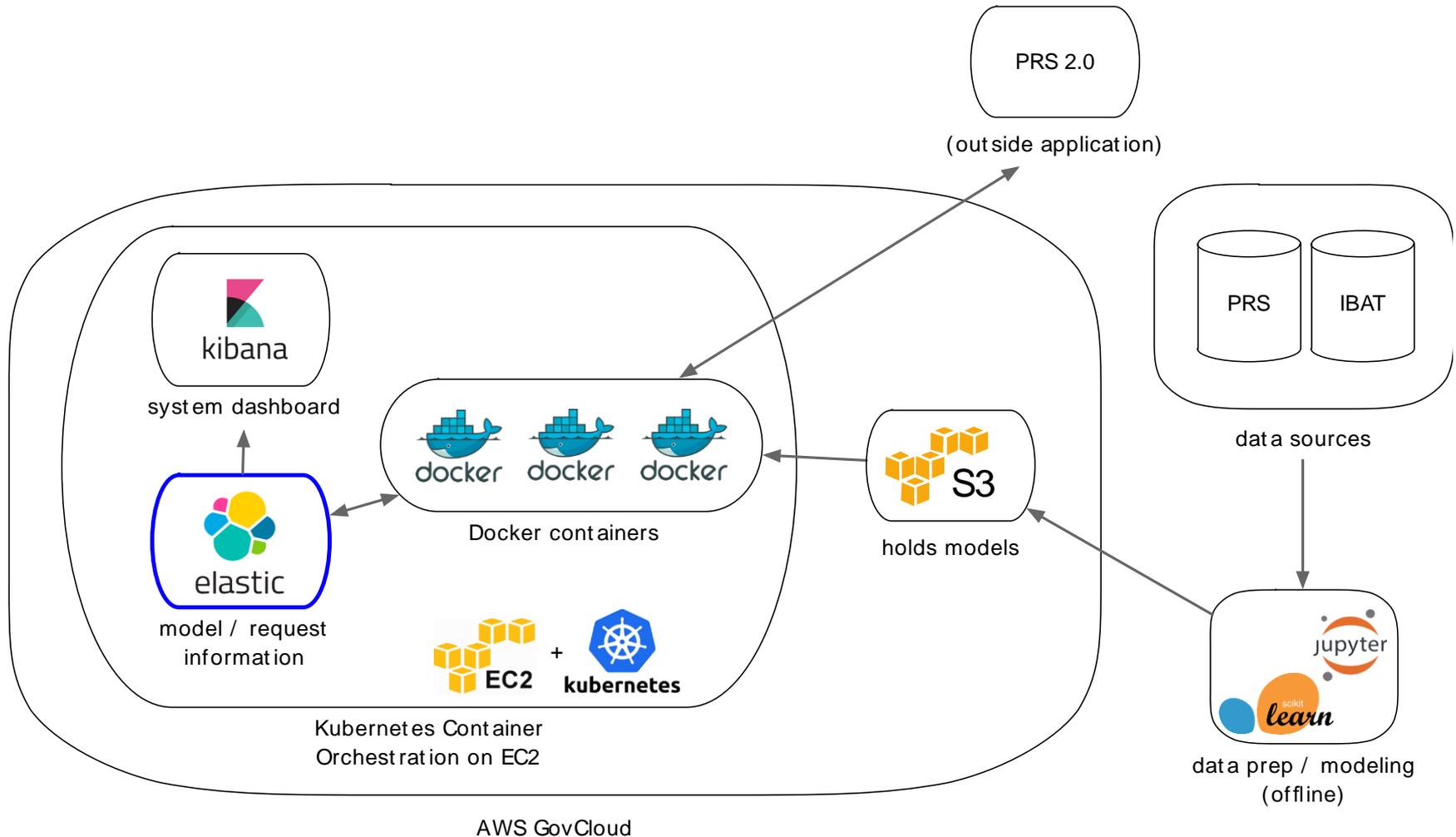
# The PRS 2.0 Architecture

Steps to deployment: use Docker



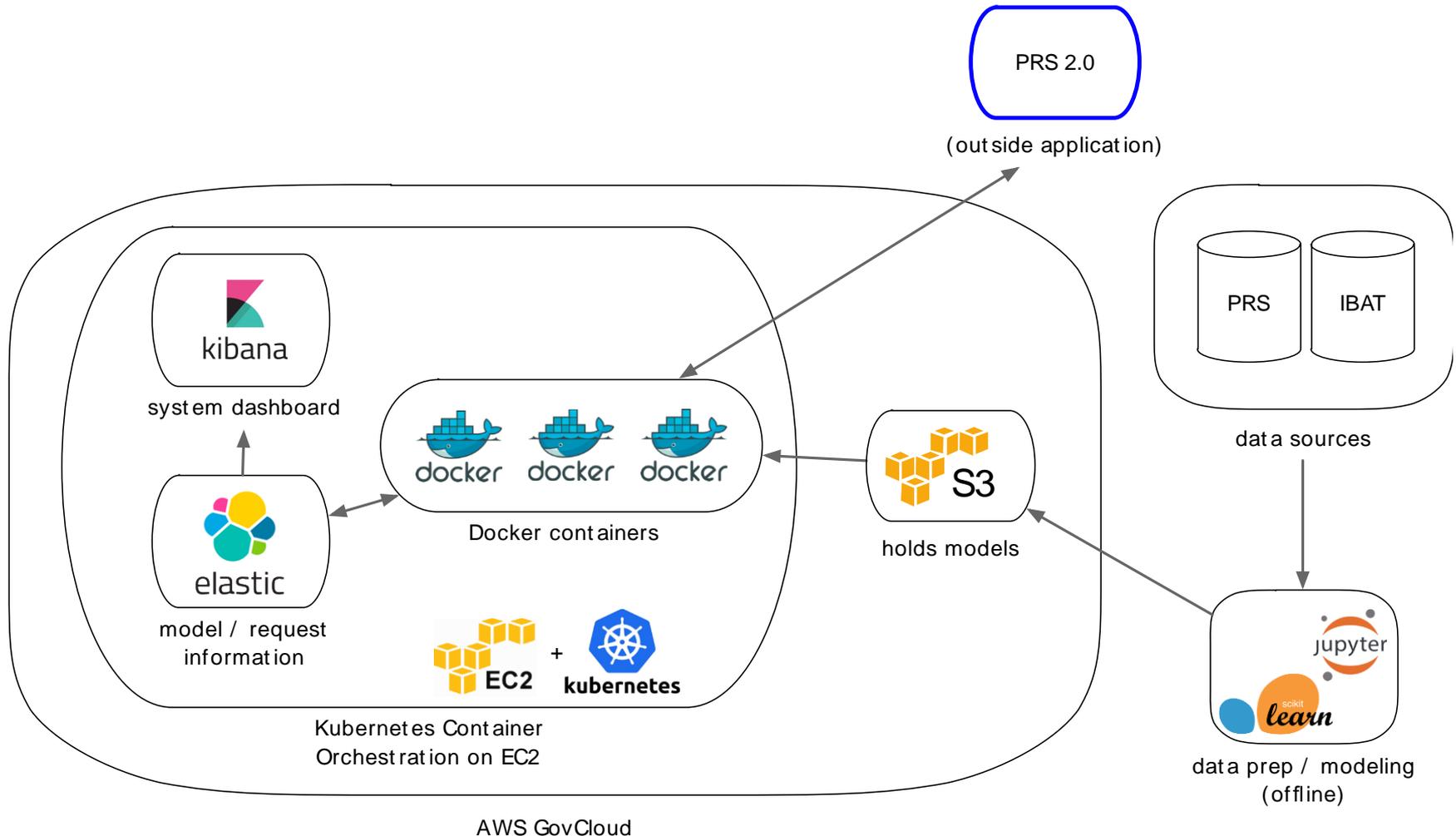
# The PRS 2.0 Architecture

Steps to deployment: models added to index



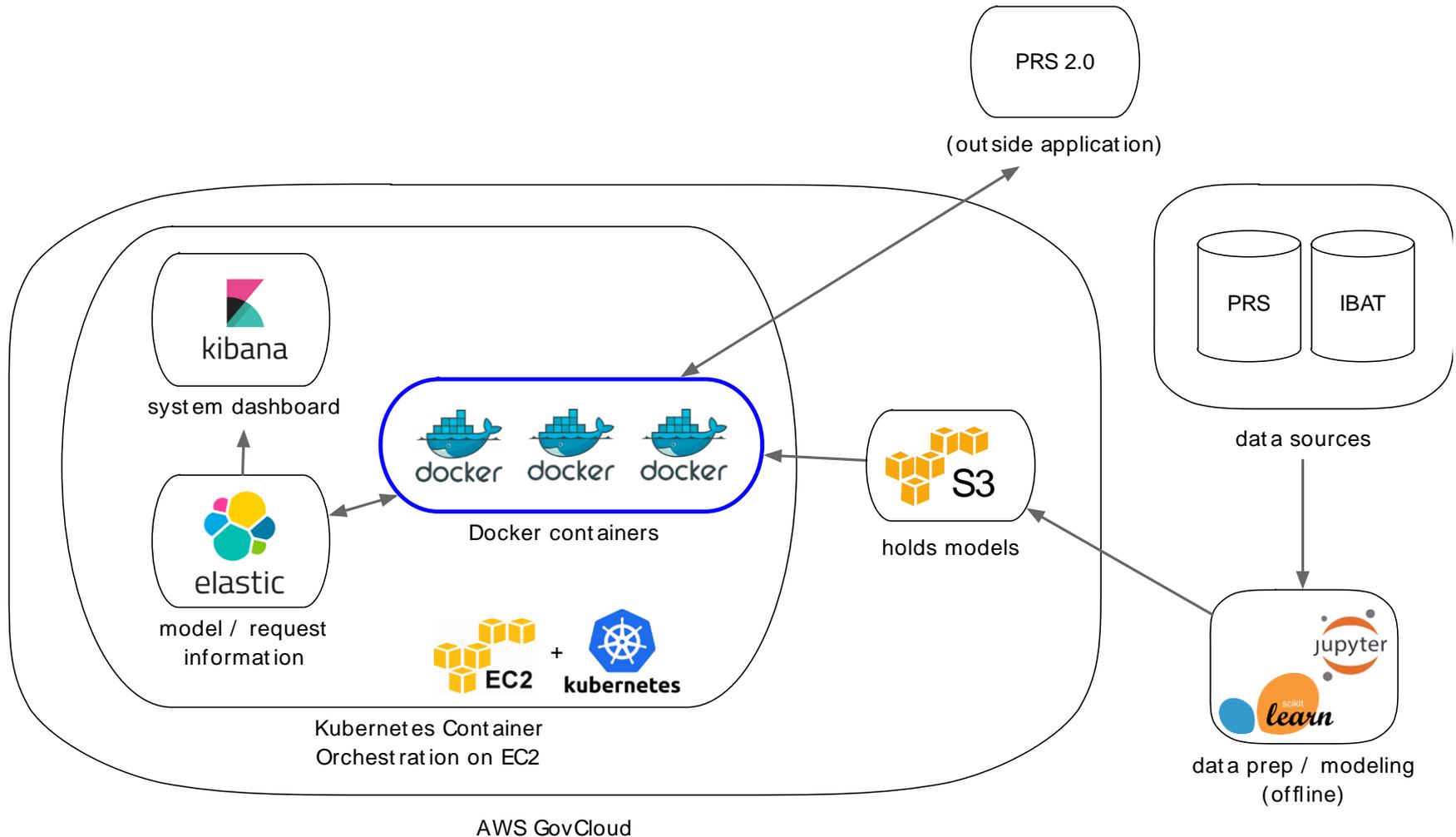
# The PRS 2.0 Architecture

Retrieving recommendations: make request



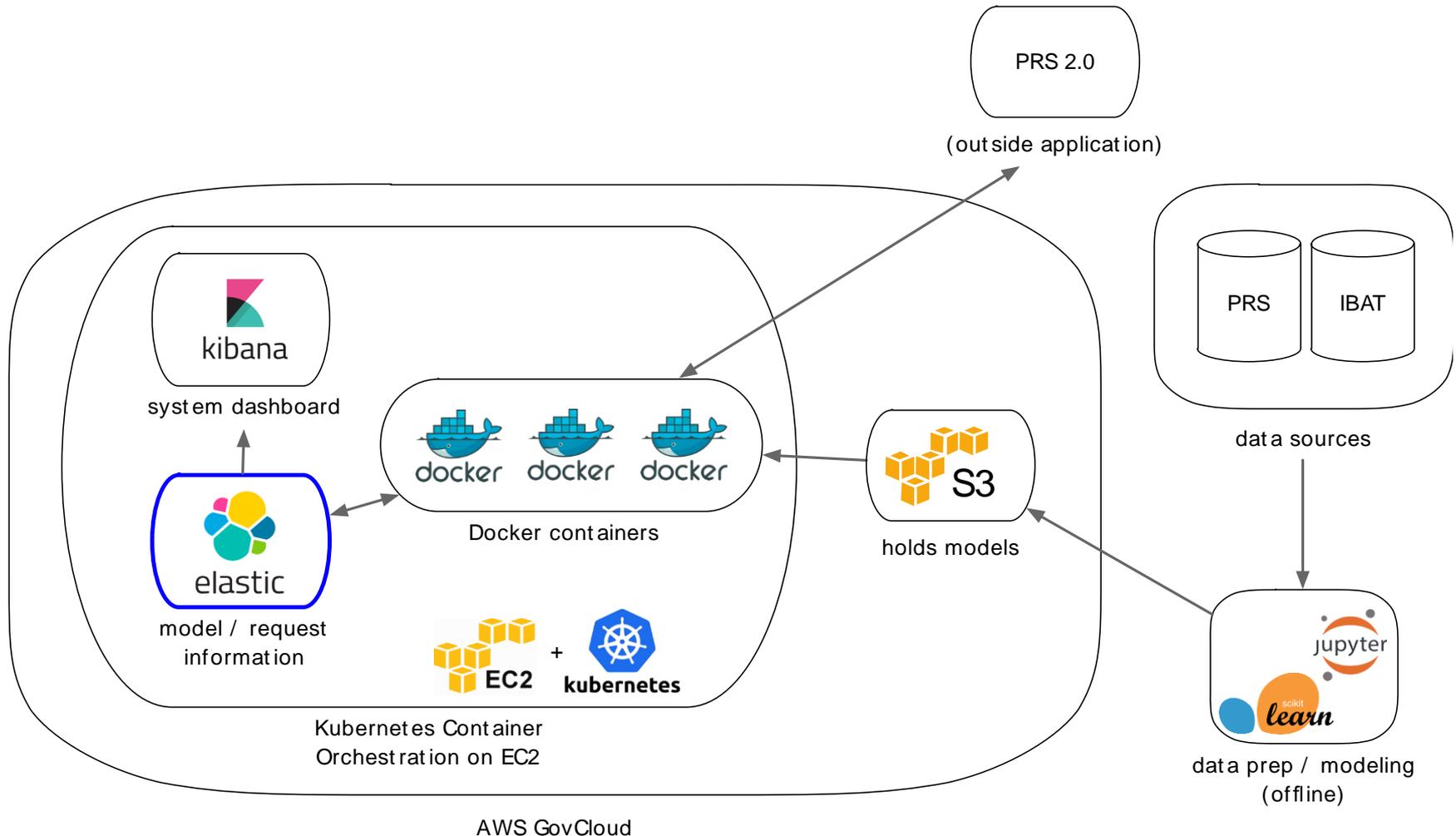
# The PRS 2.0 Architecture

Steps to deployment: Henosis processes



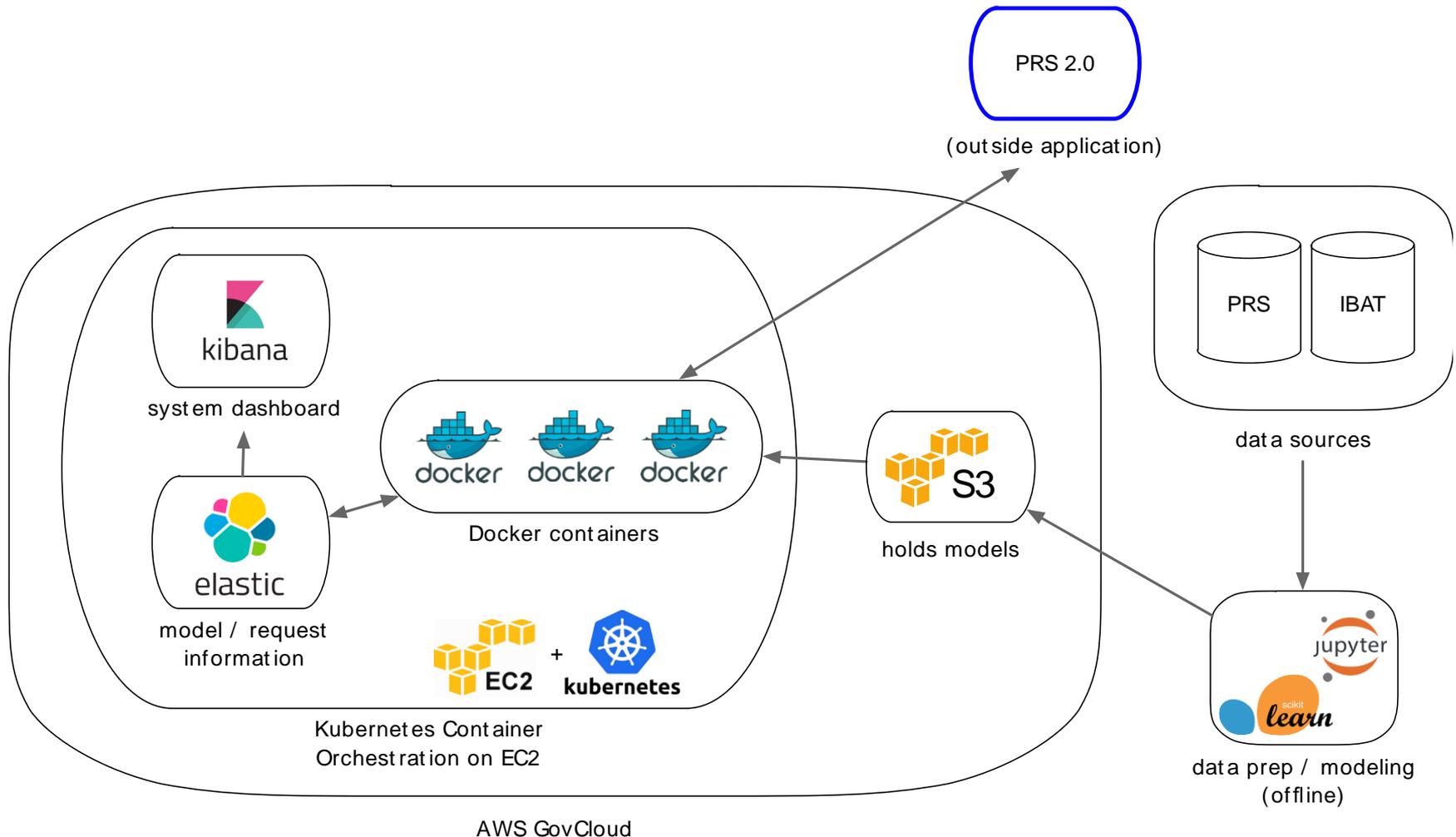
# The PRS 2.0 Architecture

Steps to deployment: logs added to index



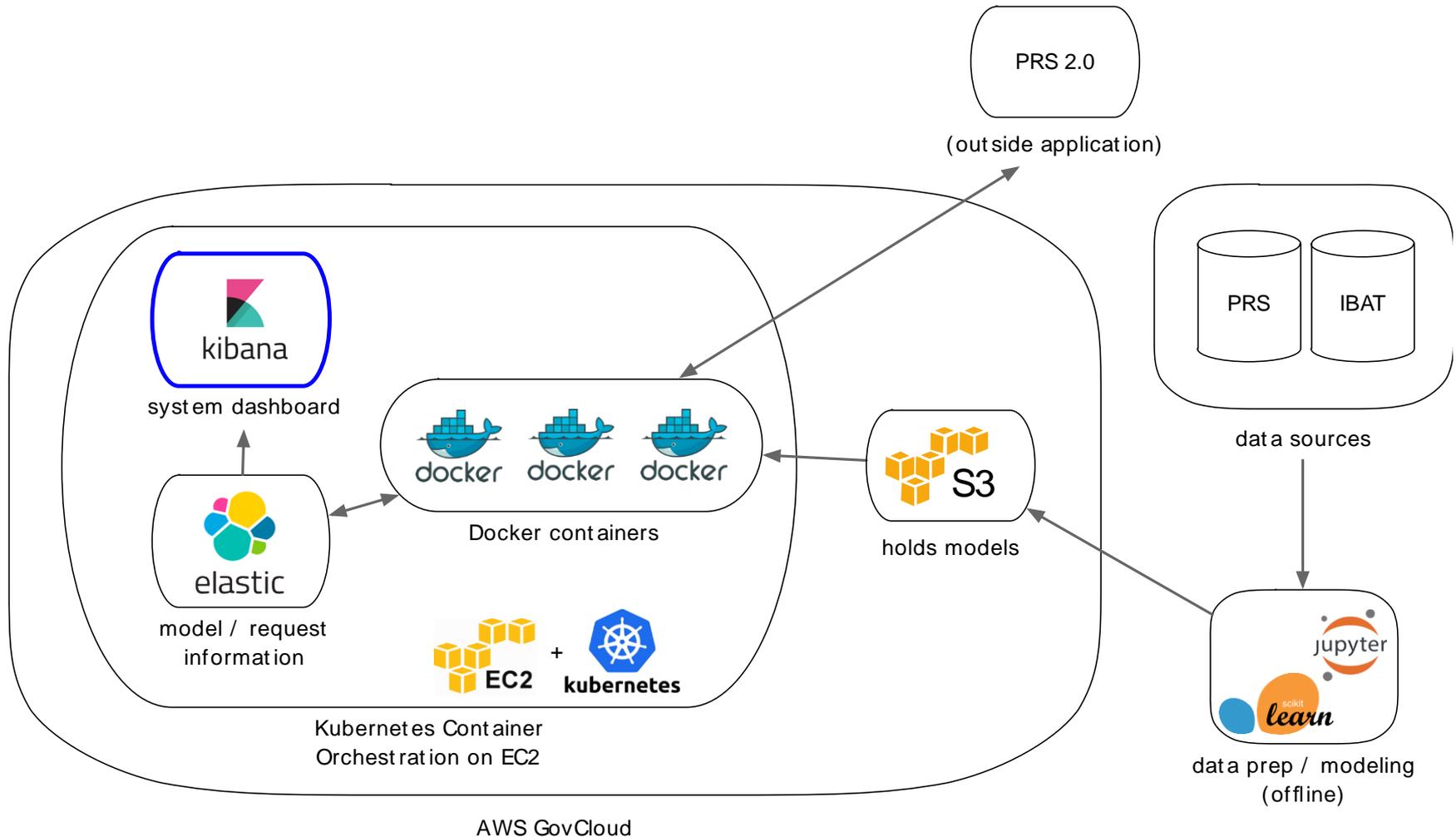
# The PRS 2.0 Architecture

Retrieving recommendations: receive



# The PRS 2.0 Architecture

Viewing system performance: Kibana



# PRS 2.0 Stress (Load) Testing

Testing scalability and reliability

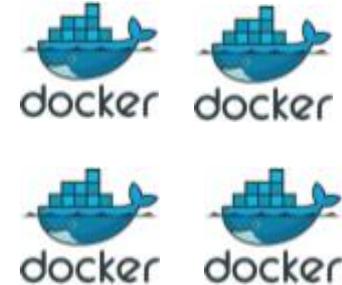
- Used Python library called *Locust* to perform stress testing of PRS 2.0 recommender system.



# PRS 2.0 Stress (Load) Testing

Testing scalability and reliability

- Used Python library called *Locust* to perform stress testing of PRS 2.0 recommender system.
- Six deployed models, one for each field.
- 4 Docker containers:
  - 8GB ram each
  - M4.4xlarge AWS EC2 instances
- Generated dummy data and bombarded system with requests.

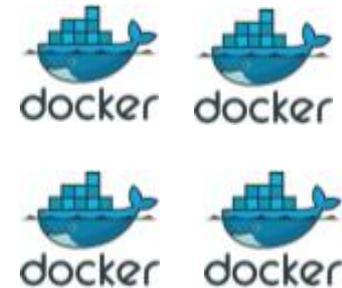


4 Docker containers

# PRS 2.0 Stress (Load) Testing

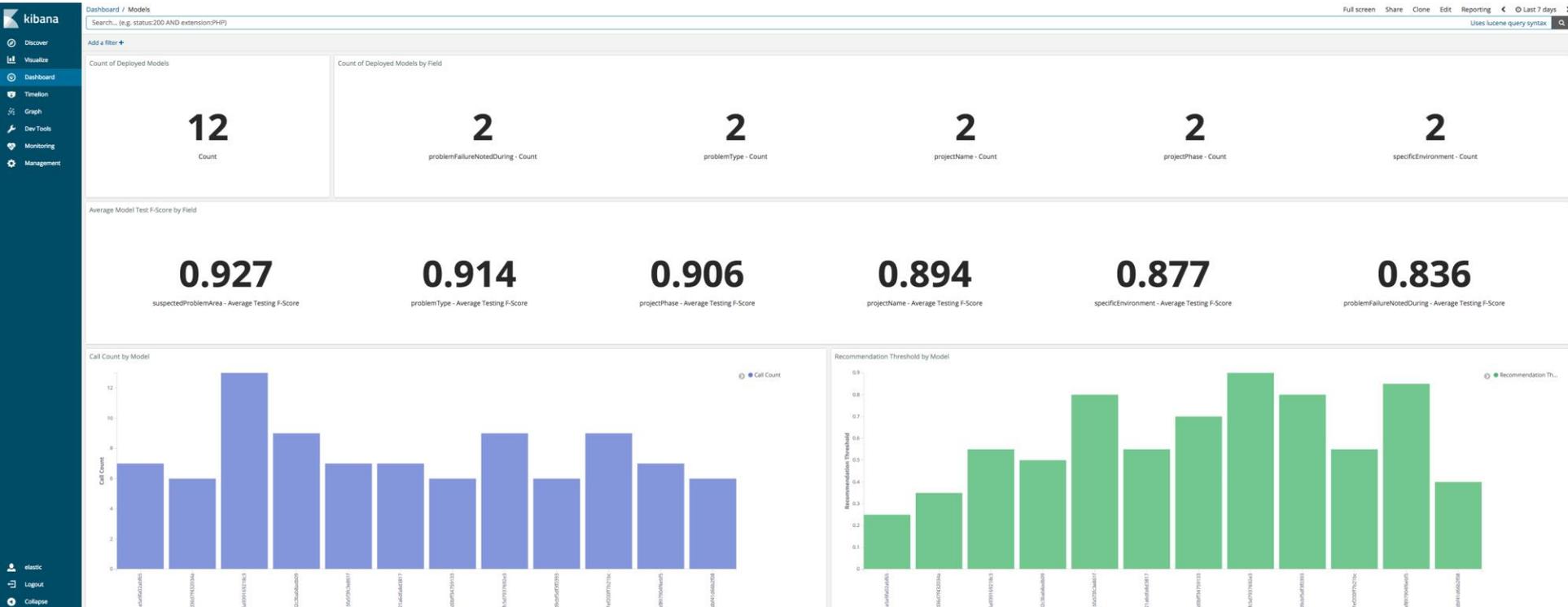
Testing scalability and reliability

- Used Python library called *Locust* to perform stress testing of PRS 2.0 recommender system.
- Six deployed models, one for each field.
- 4 Docker containers:
  - 8GB ram each
  - M4.4xlarge AWS EC2 instances
- Generated dummy data and bombarded system with requests.
- Successfully handled 800 simultaneous users averaging ~ 12 requests (recommendations) per second.



4 Docker containers

# Kibana



- Kibana allows for the exploration & visualization of data in Elasticsearch.
- We're using Kibana in our deployment to show:
  - Information about models (above) and associated fields
  - Request logs, such as session actions, latency, models queried, etc.

## 4. Wrap-Up

We've learned a lot of lessons along the way.

**L1.** Start with the simplest end-to-end pipeline,  
and build up from there.

**L2. Fit into and compliment your users' workflow.**

**L3. Be general as possible, allow for extensibility.**

**L4. Plan for failure. Build redundancy into your recommender system deployment.**

**L5.** Keep latency in mind → use fast models and functions!

**L6.** Illustrate through real-time example and demonstration to solidify concepts.

# Future Work

- It's an **open source project**. We welcome pull requests, open issues, or ideas for future work!
  - (there are several open issues on Github if you're feeling inclined)

# Future Work

- It's an **open source project**. We welcome pull requests, open issues, or ideas for future work!
  - (there are several open issues on Github if you're feeling inclined)
- Add the ability to support regression models.

# Future Work

- It's an **open source project**. We welcome pull requests, open issues, or ideas for future work!
  - (there are several open issues on Github if you're feeling inclined)
- Add the ability to support regression models.
- Explore the use of collaborative filtering and association rules (market basket analysis) in the framework.

# Future Work

- It's an **open source project**. We welcome pull requests, open issues, or ideas for future work!
  - (there are several open issues on Github if you're feeling inclined)
- Add the ability to support regression models.
- Explore the use of collaborative filtering and association rules (market basket analysis) in the framework.
- Better track individual recommendation requests to quantify which recommendations are used, when, and from which model(s).

# Future Work

- It's an **open source project**. We welcome pull requests, open issues, or ideas for future work!
  - (there are several open issues on Github if you're feeling inclined)
- Add the ability to support regression models.
- Explore the use of collaborative filtering and association rules (market basket analysis) in the framework.
- Better track individual recommendation requests to quantify which recommendations are used, when, and from which model(s).
- A/B testing with PRS 2.0 to measure time savings, dollar impact.

# Future Work

- It's an **open source project**. We welcome pull requests, open issues, or ideas for future work!
  - (there are several open issues on Github if you're feeling inclined)
- Add the ability to support regression models.
- Explore the use of collaborative filtering and association rules (market basket analysis) in the framework.
- Better track individual recommendation requests to quantify which recommendations are used, when, and from which model(s).
- A/B testing with PRS 2.0 to measure time savings, dollar impact.
- Many others!

# Some Acknowledgements

- **Ian Colwell**: for the very consistent and informative feedback on framework development. Such an awesome project partner!
- This work would not be possible without the generous support from the Office of Safety and Mission Success (5X).
  - **Leslie Callum**: for organizing information and people between 5X and our office (no small feat!)
  - **Harold Schone**: for believing in and pushing for using data science in internal applications on the lab.
- **Kyle Hundman**: for technical expertise and feedback.
- **Paul Ramirez**: for spurring ideas and further thinking.
- **Kevin Reelfs** and **Tom Soderstrom**: for continued support, interest, and advocating of our team's projects.

## **Give it a try!**

*Framework:* <https://www.github.com/vc1492a/henosis>

*Documentation:* <https://www.henosis.io>

## **Let's chat!**

[vconstan@jpl.nasa.gov](mailto:vconstan@jpl.nasa.gov)



**Jet Propulsion Laboratory**  
California Institute of Technology

---

[jpl.nasa.gov](http://jpl.nasa.gov)

# Appendix

# Why is it called Henosis?

