Continuous Delivery for Machine Learning

Open Data Science Conference Europe
September 17, 2020

Christoph Windheuser - cwindheu@thoughtworks.com
David Johnston - dajohnst@thoughtworks.com
Eric Nagler - eric.nagler@thoughtworks.com

ThoughtWorks®

YOUR WORKSHOP TEAM

Christoph Windheuser

Global Head of Artificial Intelligence

cwindheu@thoughtworks.com

David Johnston

Principal Data Scientist

dajohnst@thoughtworks.com

Eric Nagler

Lead Data Engineer

eric.nagler@thoughtworks.com

www.thoughtworks.com

AGENDA

- Introduction and Setup ..... 10 Min
- Discussion of Continuous Delivery for Machine Learning (CD4ML) ..... 45 Min
- Break ..... 10 Min
- Workshop ..... 1h 45 Min
  - Doing the plumbing: Set up the pipeline and see if it is working ..... 20 Min
  - Data Science: Develop the model and test it (with TDD) ..... 50 Min
  - Continuous Deployment: Bring the model into production ..... 20 Min
  - Monitor the model “in the wild” ..... 15 Min
- Questions and Answers ..... 10 Min

ThoughtWorks®

7000 technologists with 40 offices in 14 countries

Partner for technology driven business transformation

100+

books written

#1

in Agile and Continuous Delivery

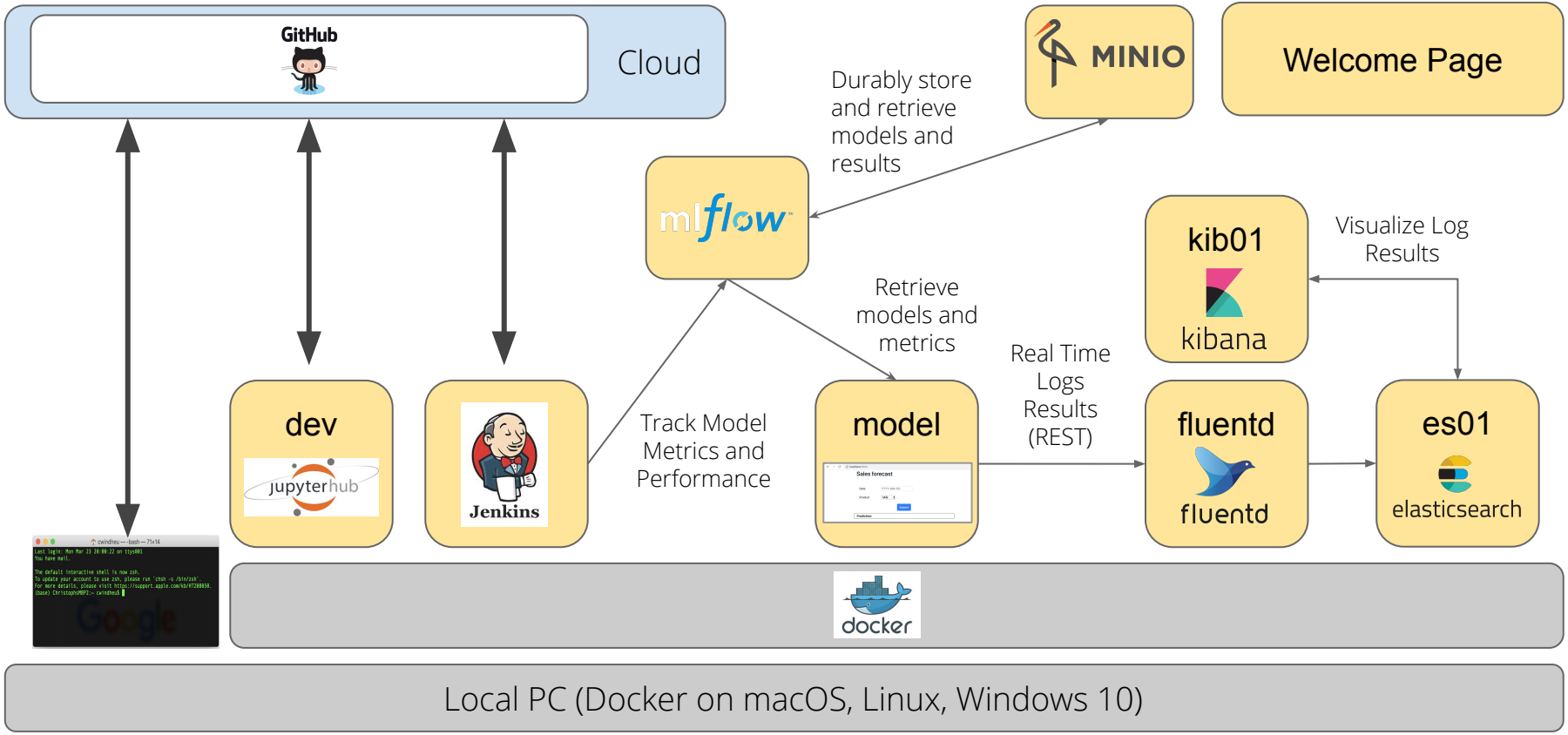
TECHNIQUES

Continuous delivery for machine learning (CD4ML) #8

TRIAL

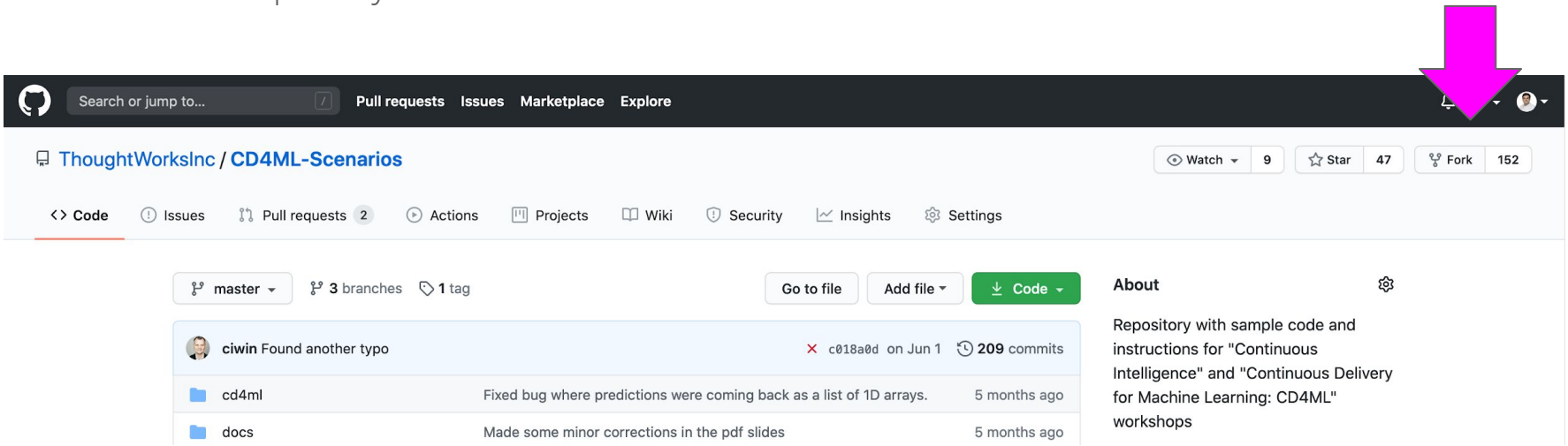
Setting up your local environment

Big Picture of Workshop Services



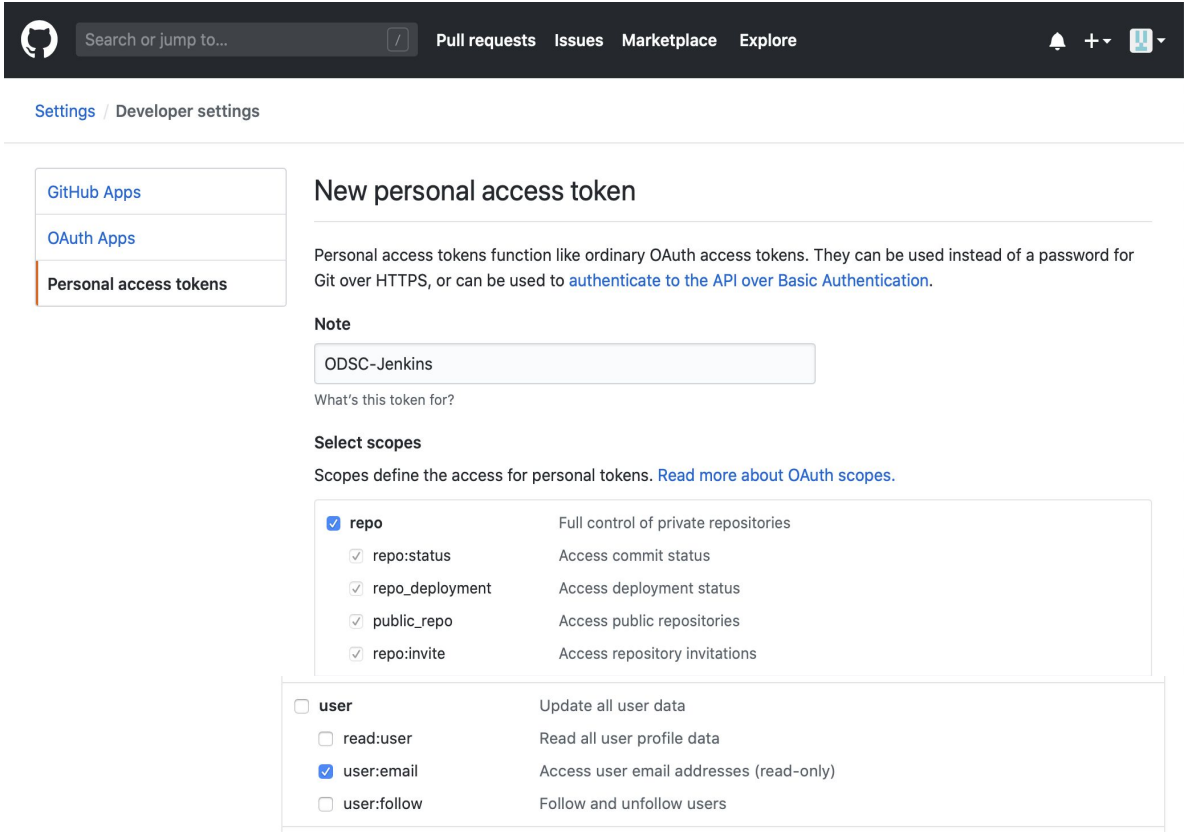
Installation (1)

- Navigate to https://github.com/ThoughtWorksInc/CD4ML-Scenarios
- Fork the repo to your Personal GitHub Account



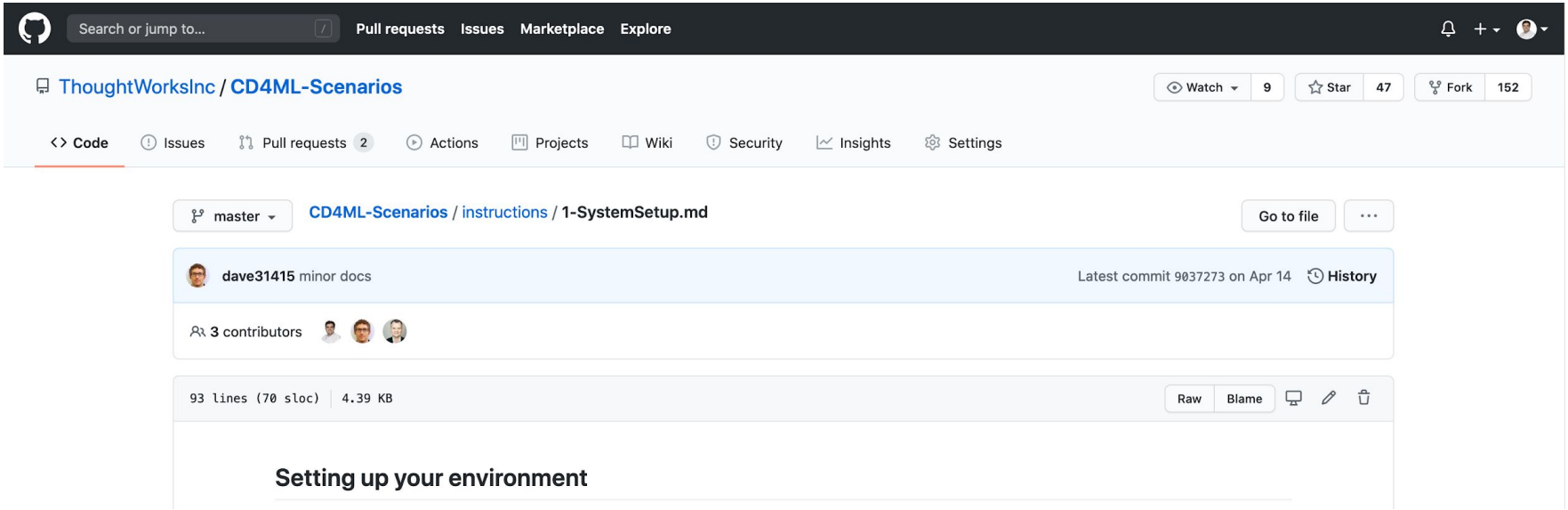
Installation (2)

- Create a github personal access token with the following permissions to check out the code.
- When you click Generate Token a token will be displayed to you. Please securely save this token because it will only be displayed once. Do not



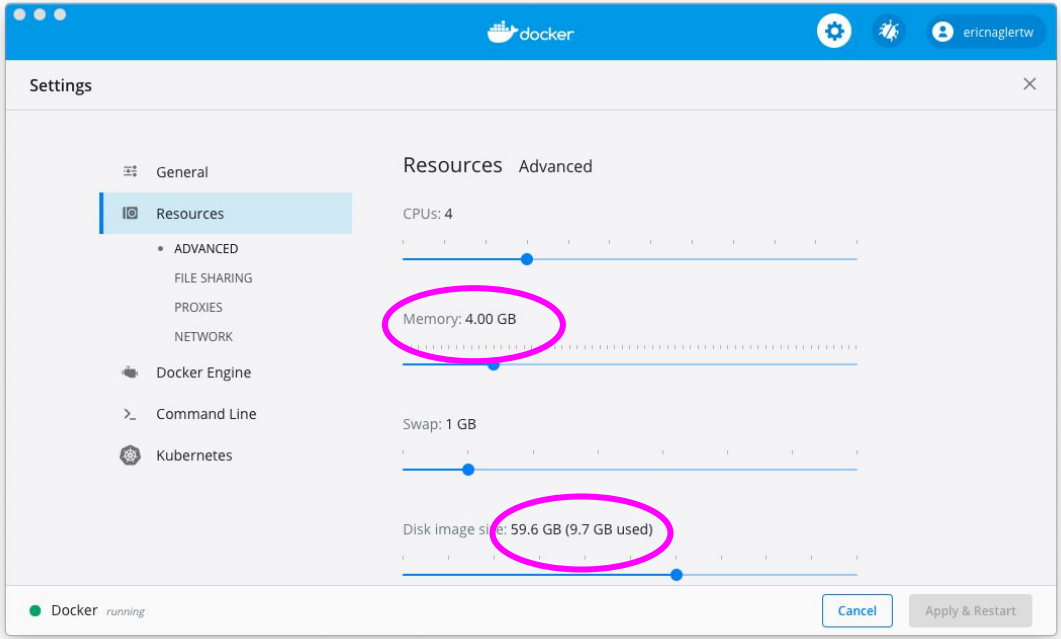
Installation (3)

- Navigate to "<your github username>/CD4ML-Scenarios/instructions/1-SystemSetup.md" to begin setting up your development environment



Installation (4)

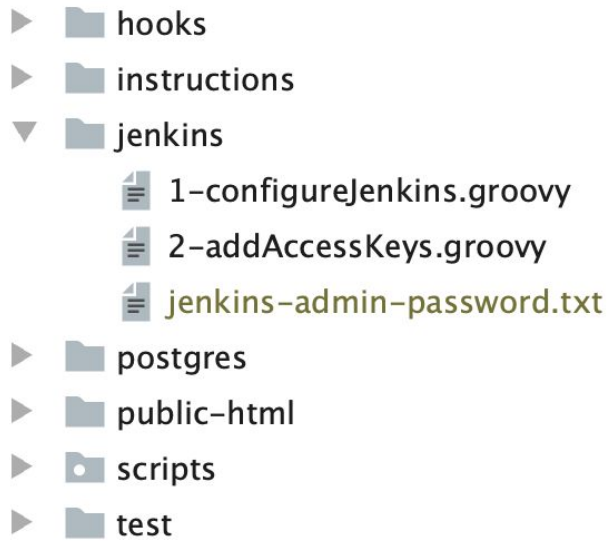
- Increase the RAM allocated to docker to 4 Gigabytes (or more)
- Ensure you have about 20 GB Free on your Docker image



Installation (5)

Configuring your jenkins password

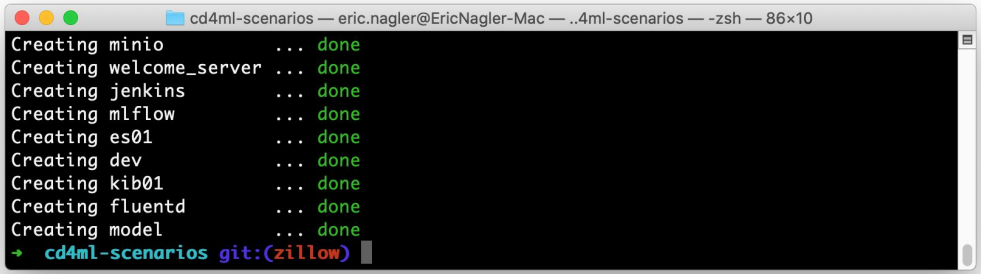
- Create a file in jenkins/jenkins-admin-password.txt
- Open the file and put in a secure password on the first line. Save and close the file.
- When you log into Jenkins your username will be ‘admin’



Installation (6)

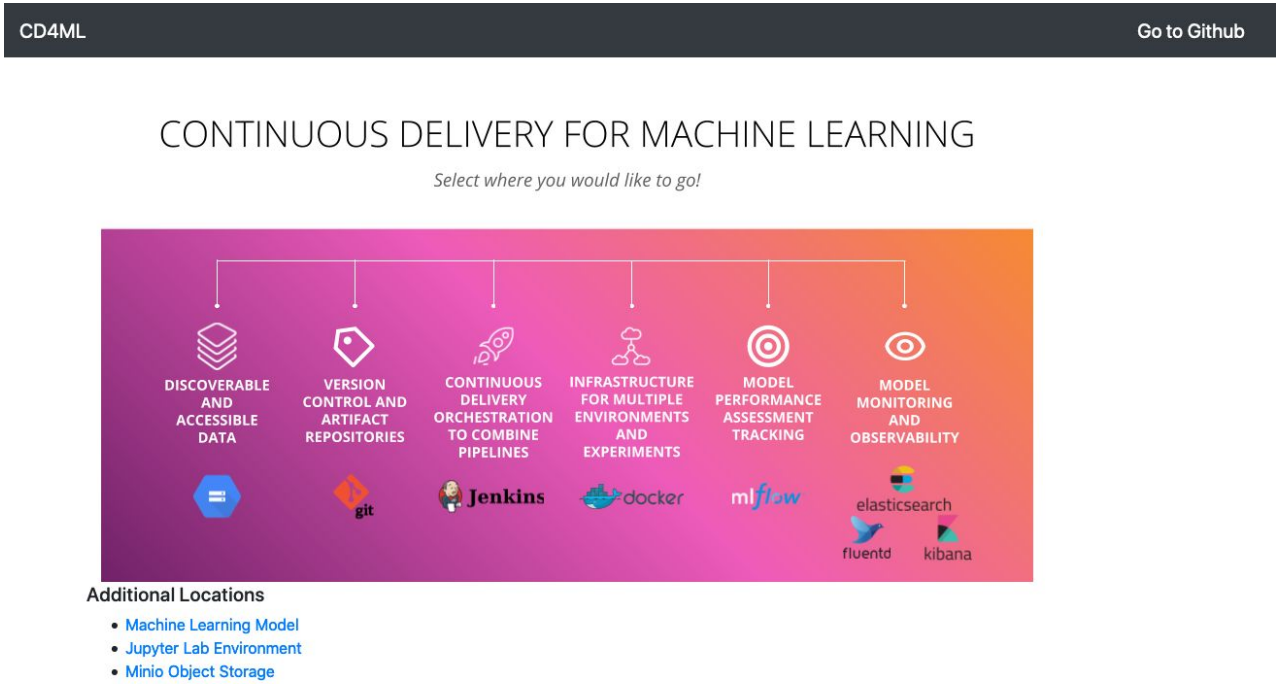
- You choice of code development environment
  - JupyterHub Notebook
    - Ready to go Jupyter Environment with python and JupyterLab ready
    - Follow the “Setting up your local environment (using JupyterLab Development Environment)”
  - Local Machine virtualenv environment
    - Use your preferred tools for python development
    - Follow the “Setting up your local environment (using local machine environment)”
- Start the environment using docker-compose to download the images and software

```
docker-compose up -d --build
```



HOMEPAGE

- Navigate to homepage - http://localhost:3000



Continuous Delivery for Machine Learning (CD4ML)

PRODUCTIONIZING ML IS HARD

HOW DO WE APPLY DECADES OF SOFTWARE DELIVERY EXPERIENCE TO INTELLIGENT SYSTEMS?

Production systems should be:

- Reproducible
- Testable
- Auditable
- Continuously Improving

Machine Learning is:

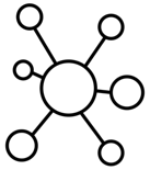
- Non-deterministic (somewhat)
- Hard to test
- Hard to explain
- Hard to improve

CD4ML isn't a technology or a tool; it is a practice and a set of principles. Quality is built into software and improvement is always possible.

But machine learning systems have unique challenges; unlike deterministic software, it is difficult—or impossible—to understand the behavior of data-driven intelligent systems. This poses a huge challenge when it comes to deploying machine learning systems in accordance with CD principles.

MANY SOURCES OF CHANGE

+



+

Data

Schema

Sampling over Time

Volume

...

Model

Research, Experiments

Training on New Data

Performance

...

Code

New Features

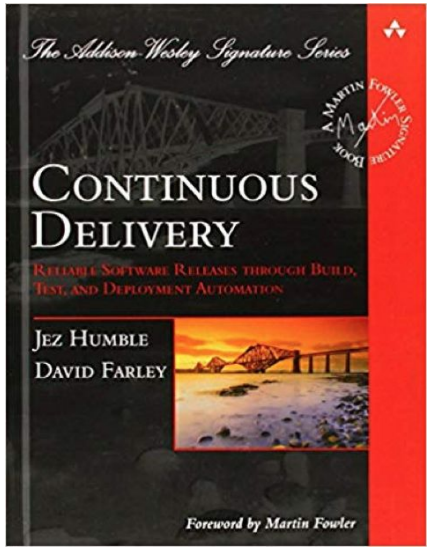
Bug Fixes

Dependencies

...

Icons created by Noura Mbarki and I Putu Kharismayadi from Noun Project

PRINCIPLES OF CONTINUOUS DELIVERY



Published 2010

- → Create a Repeatable, Reliable Process for Releasing Software
- → Automate Almost Everything
- → Build Quality In
- → Work in Small Batches
- → Keep Everything in Source Control
- → Done Means “Released”
- → Improve Continuously

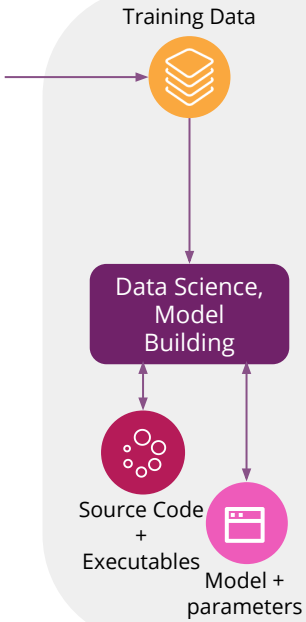
*"Continuous Delivery is the ability to get **changes of all types** — including new features, configuration changes, bug fixes and experiments — into production, or into the hands of users, **safely and quickly** in a **sustainable way.**"*

- Jez Humble & Dave Farley

Continuous Delivery for Machine Learning (CD4ML) is a software engineering approach in which a ***cross-functional team*** *produces machine learning applications* ***based on code, data, and models*** *in* ***small and safe increments*** *that can be* ***reproduced*** *and* ***reliably released*** *at* ***any time****, in* ***short adaptation cycles****.*

HOW DOES IT WORK

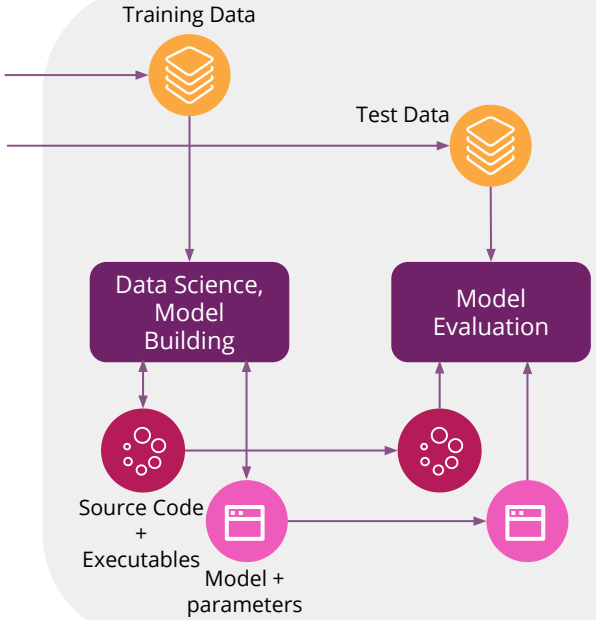
Discoverable and Accessible Data



CD Tools and Repositories

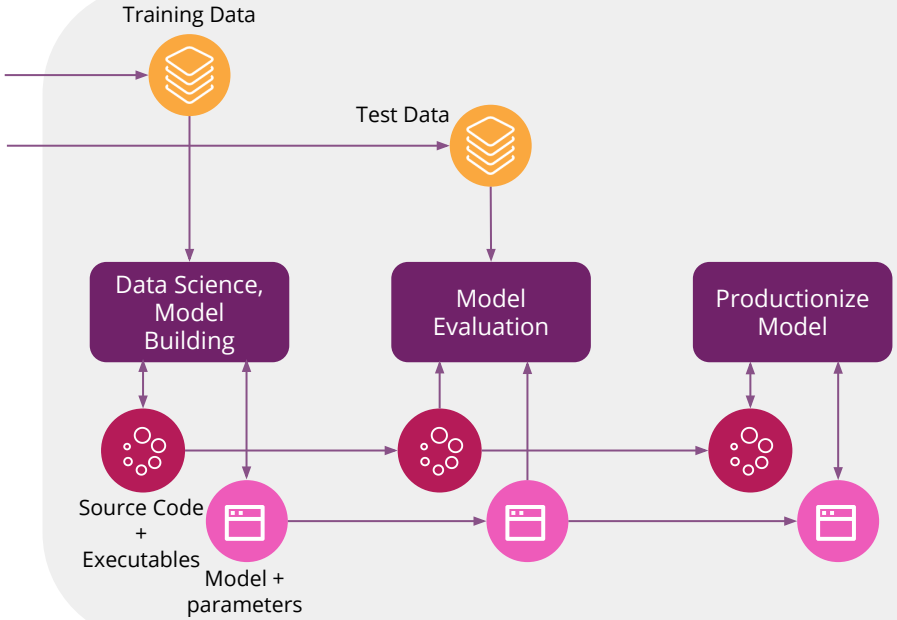
HOW DOES IT WORK

Discoverable and Accessible Data



CD Tools and Repositories

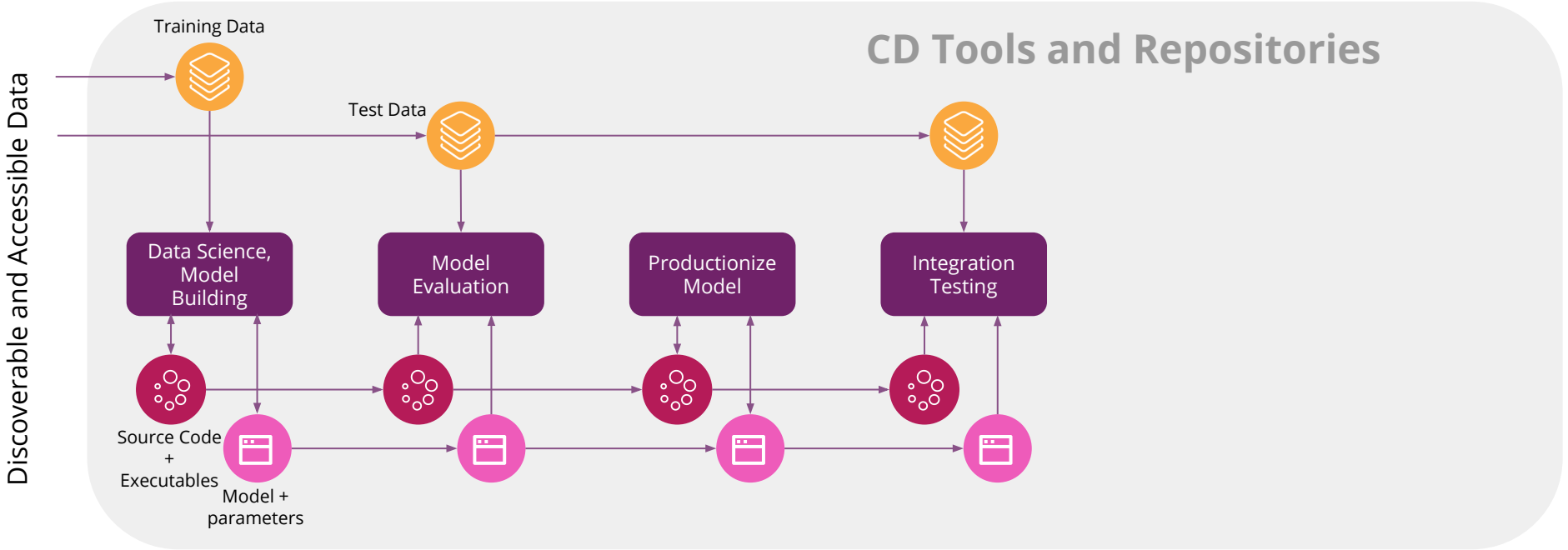
HOW DOES IT WORK



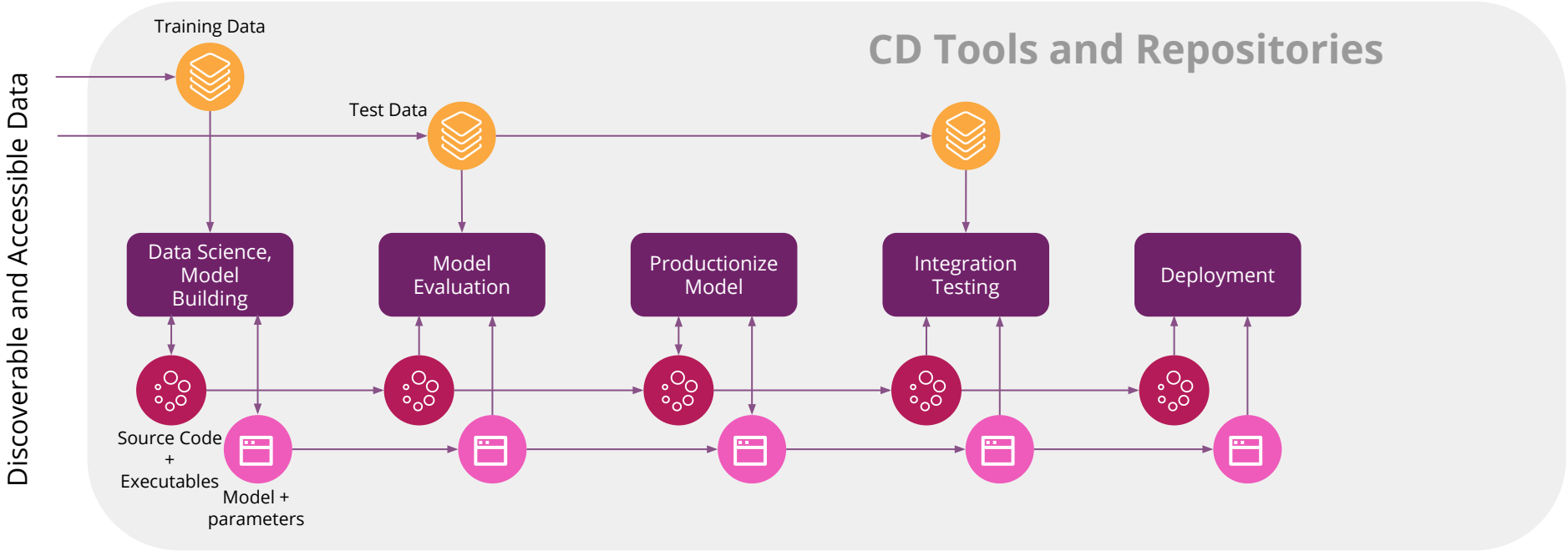
CD Tools and Repositories

Discoverable and Accessible Data

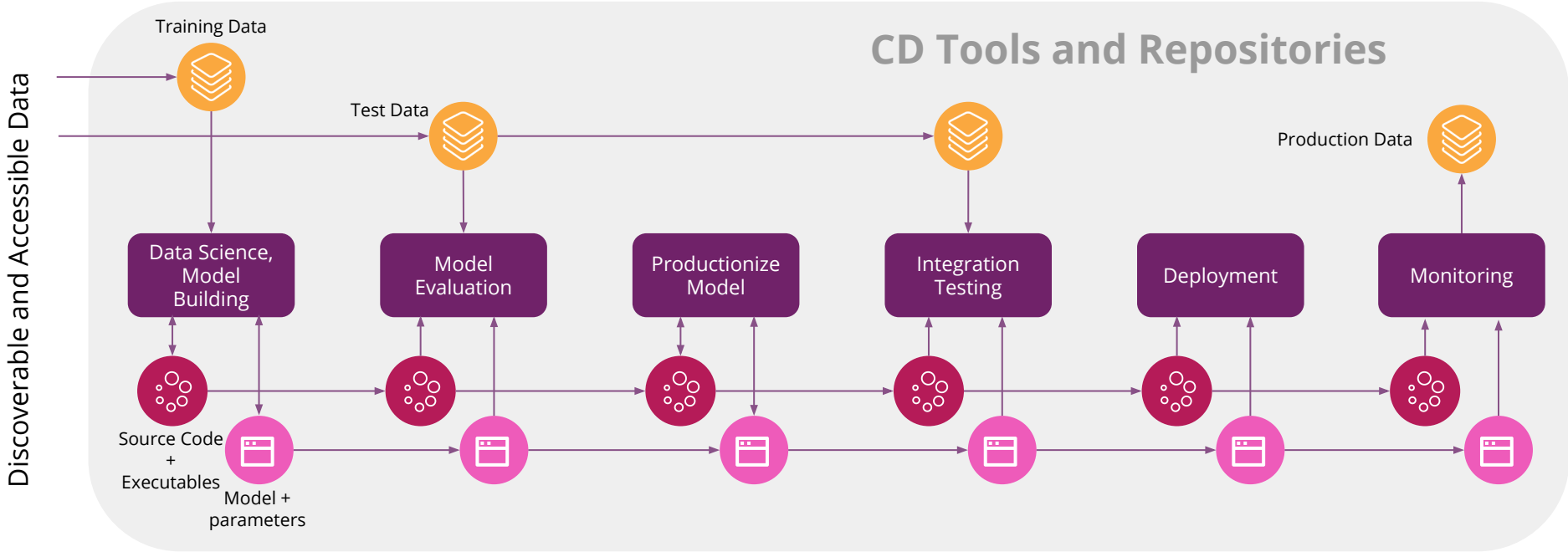
HOW DOES IT WORK



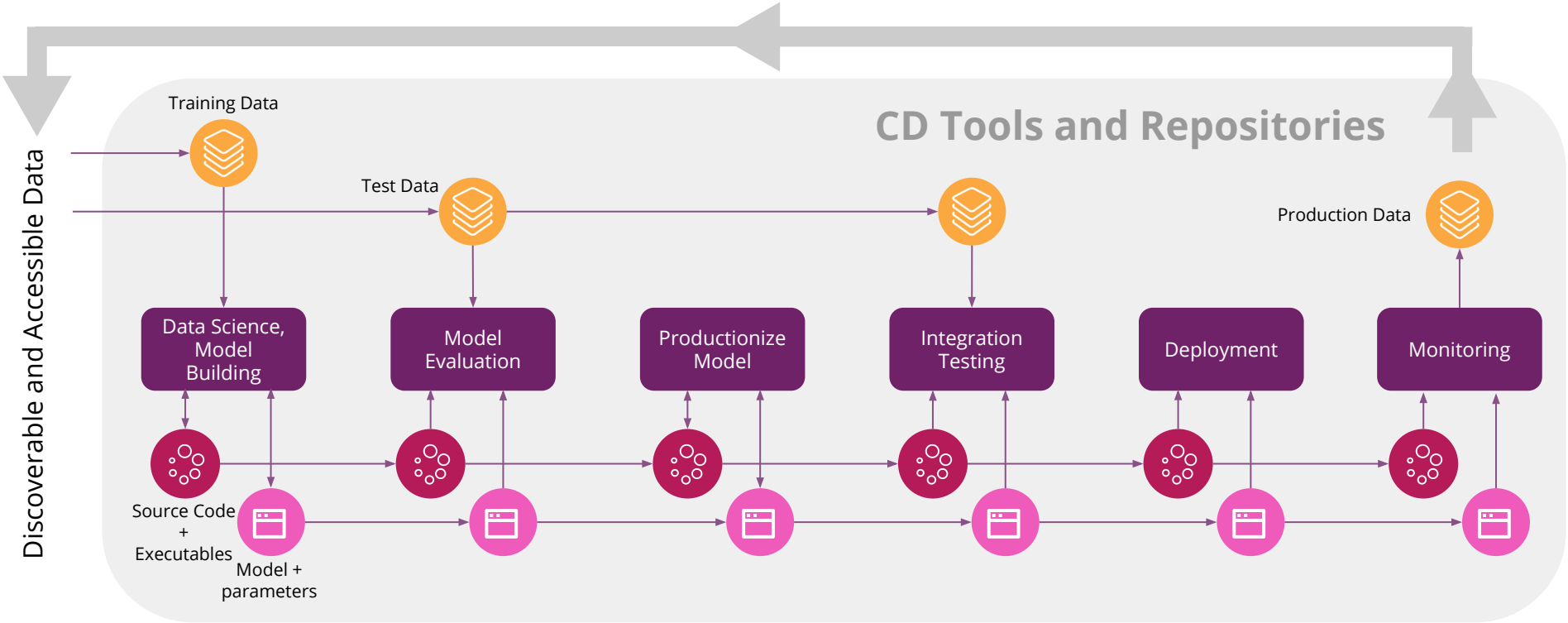
HOW DOES IT WORK



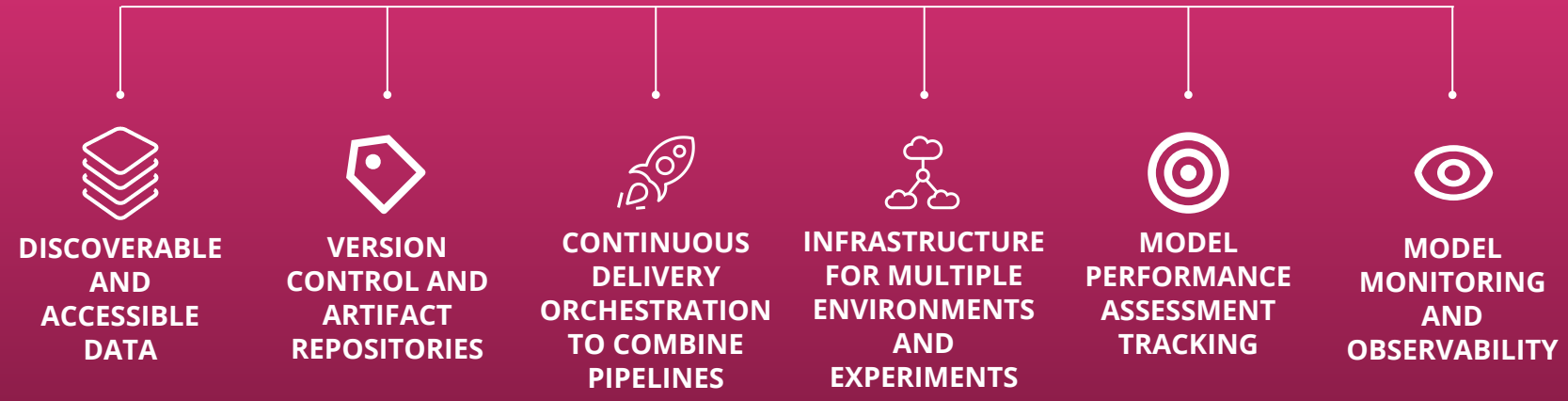
HOW DOES IT WORK



HOW DOES IT WORK



WHAT DO WE NEED IN OUR CD4ML STACK?



THE BUSINESS PROBLEM: PREDICT HOUSE PRICES

Make a pricing prediction based on a multitude of data:

- ZIP code
- Lot size and square feet
- No. of rooms
- Year built
- Style
- Extras (parking, pool, etc.)

Based on publicly available data from the Zillow Group.

Picture from https://www.zillowgroup.com/

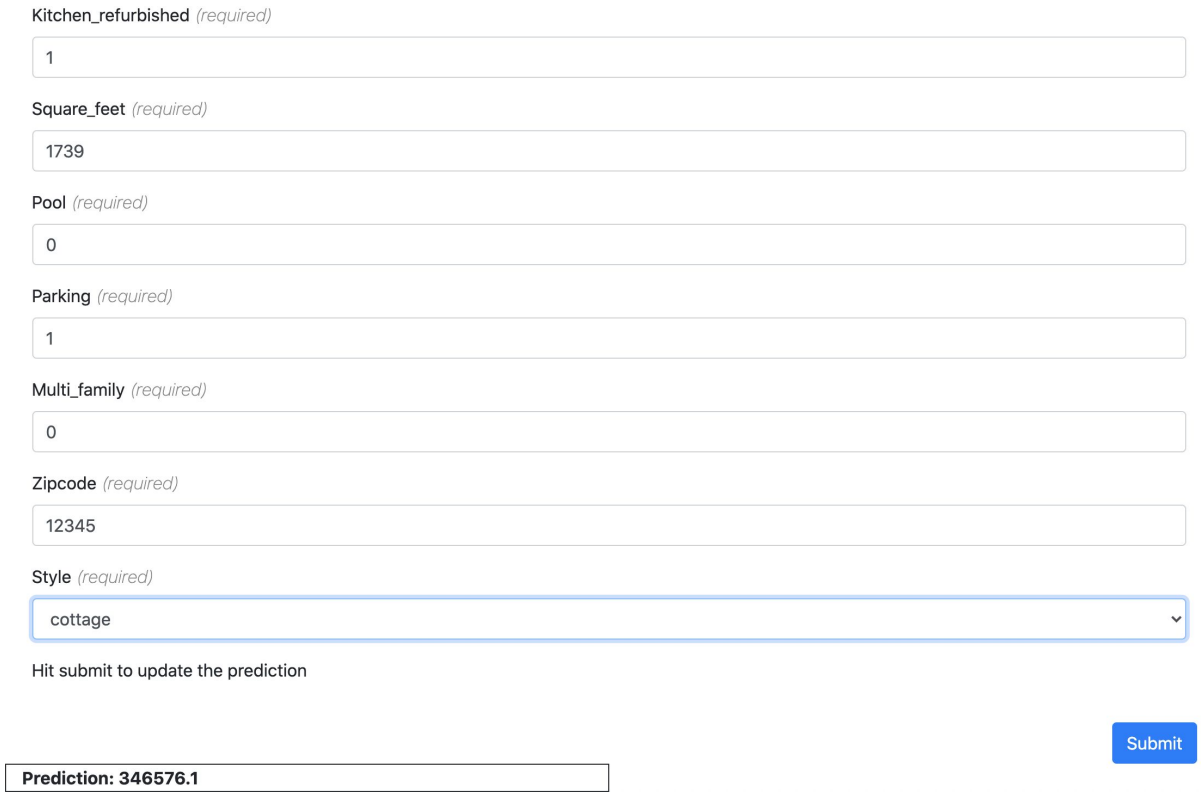
THE DATA

To train our machine learning models, we are using a publicly available data set from the Zillow Group containing:

- You can find the data here
- The original data are from the kaggle platform

WE WILL BUILD A SIMPLIFIED WEB APPLICATION

As a seller or a buyer, I want to get a price estimation based on my preferences.



WHAT WE WILL USE IN THIS WORKSHOP

There are many options for tools and technologies to implement CD4ML

DISCOVERABLE AND ACCESSIBLE DATA

VERSION CONTROL AND ARTIFACT REPOSITORIES

CONTINUOUS DELIVERY ORCHESTRATION TO COMBINE PIPELINES

INFRASTRUCTURE FOR MULTIPLE ENVIRONMENTS AND EXPERIMENTS

MODEL PERFORMANCE ASSESSMENT TRACKING

MODEL MONITORING AND OBSERVABILITY

Break (10 min)

STEP BY STEP

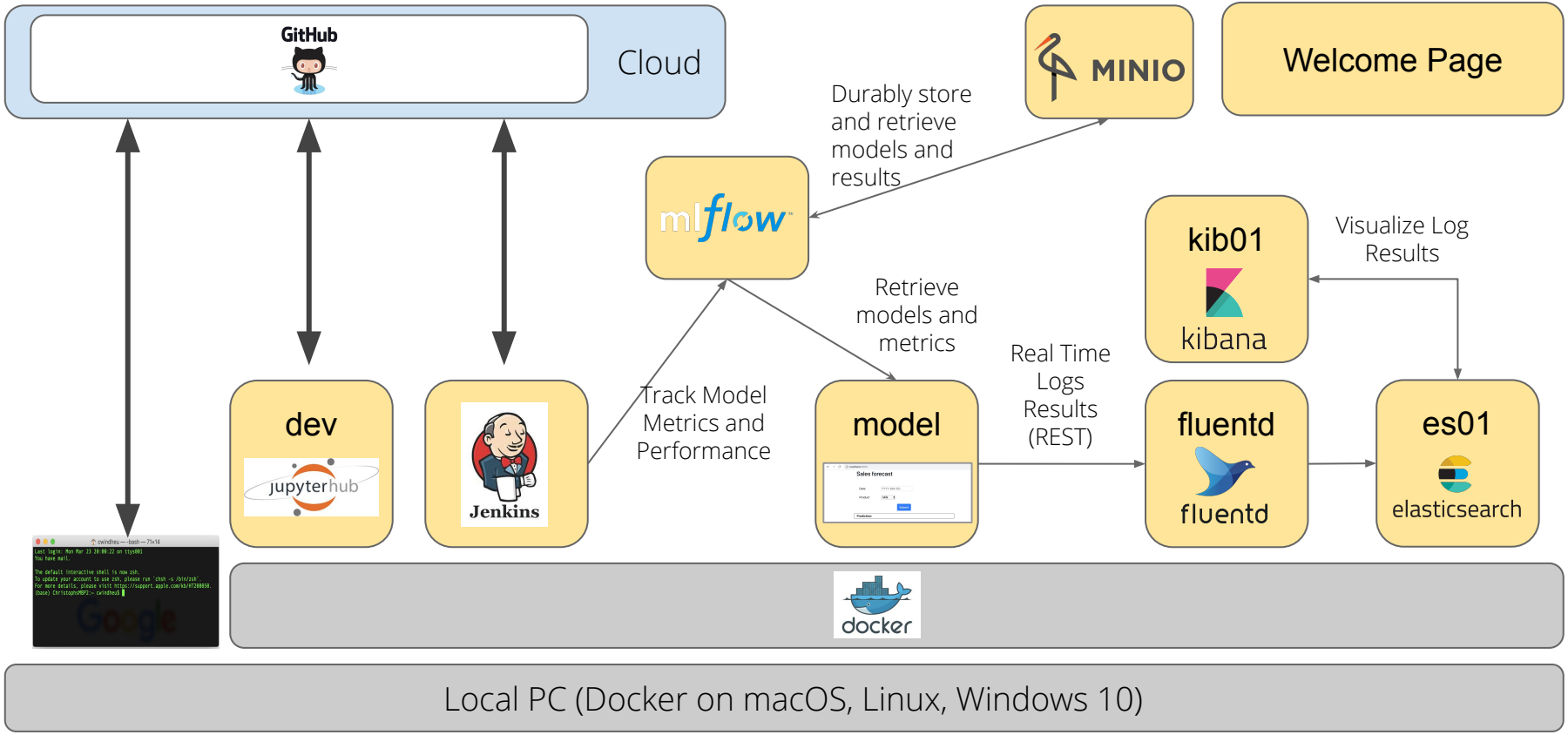
1. **Doing the plumbing:** Set up the pipeline and see if it is working

2. **Data Science:** Develop the model and test it (with TDD)
    a. Run the ML pipeline
    b. Run experiments
    c. Change parameters, improve score and monitor models in ML Flow
    d. Overview of codebase and design
    e. Testing (TDD)

3. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the quality threshold

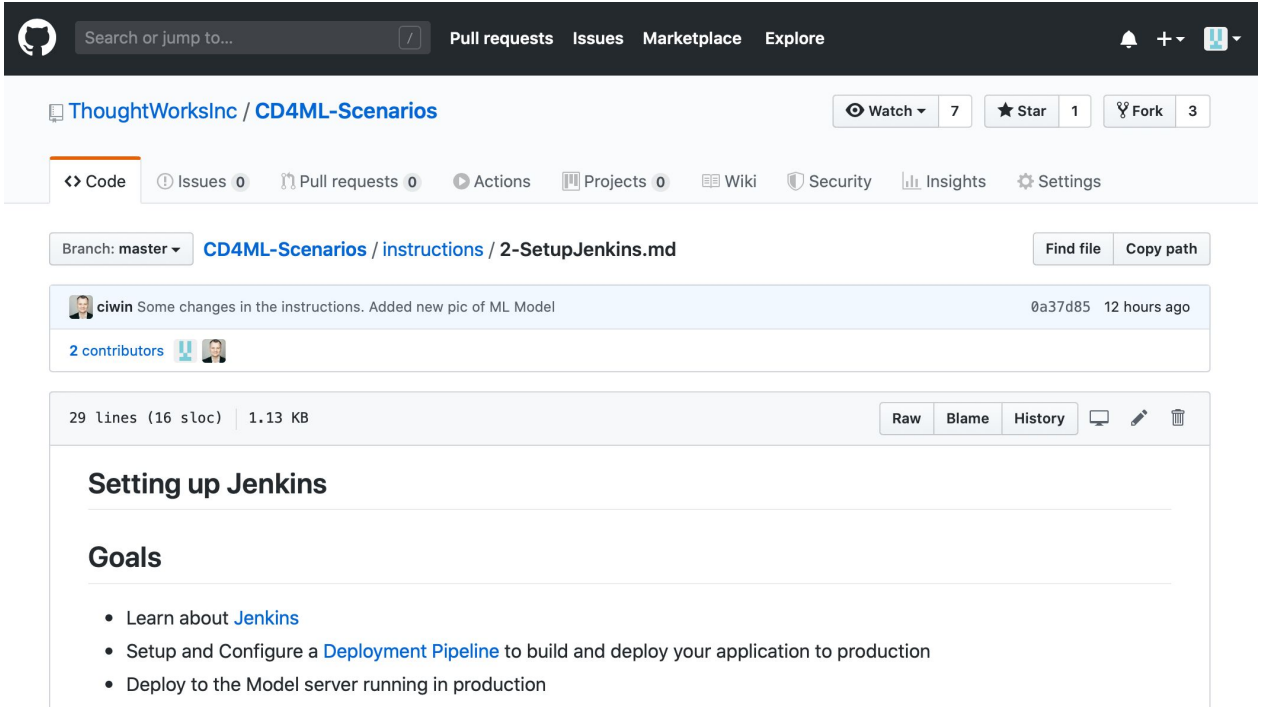
4. **Monitor** the model “in the wild”

The Workshop Infrastructure

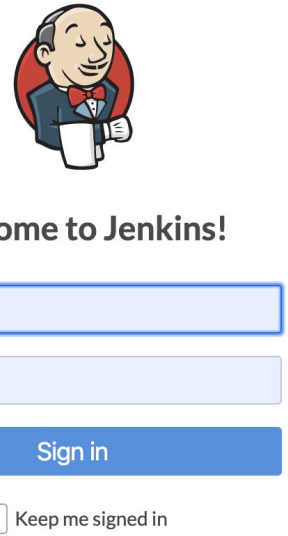


SETUP JENKINS

- Navigate to [instructions/2-SetupJenkins.md]

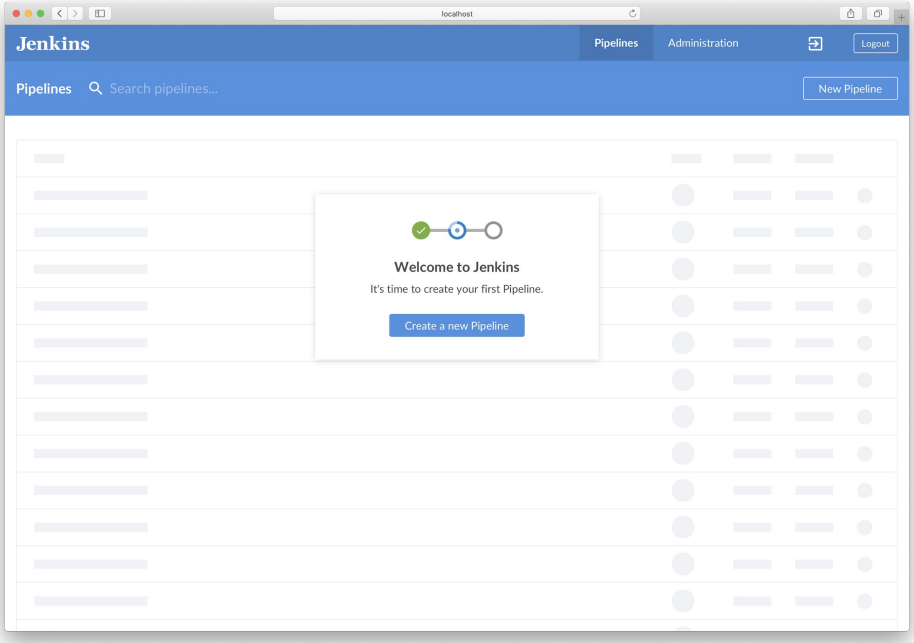


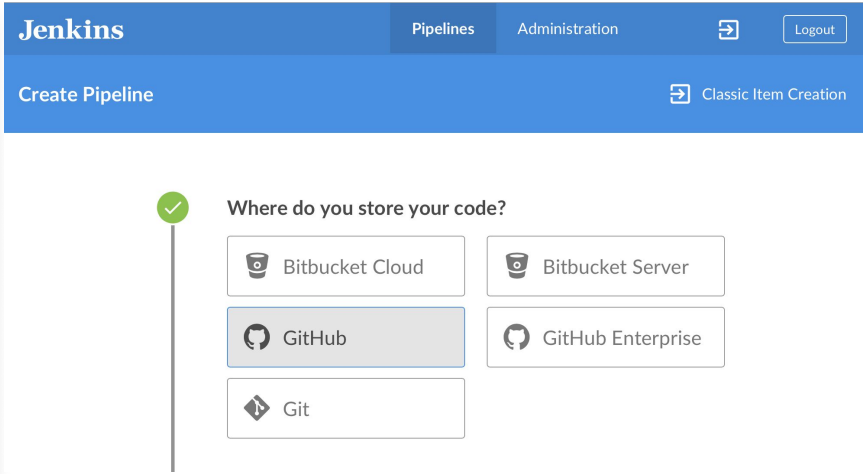
LOG IN TO JENKINS



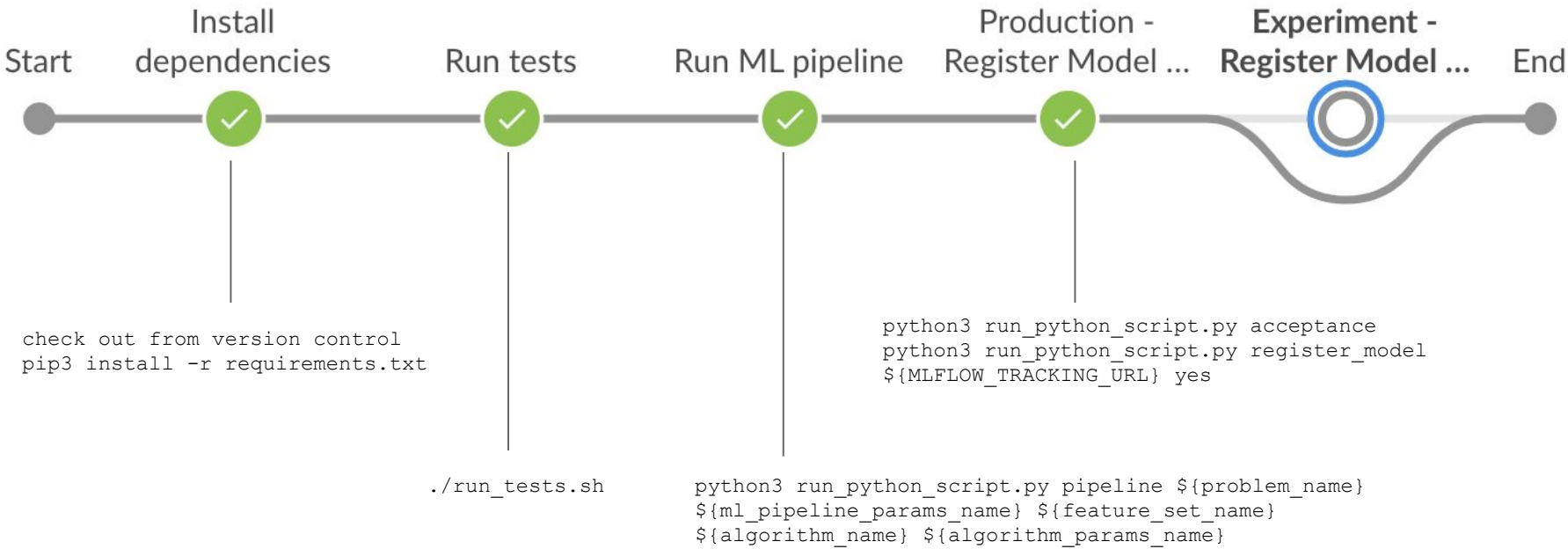
http://localhost:10000/blue

SETUP THE PIPELINE

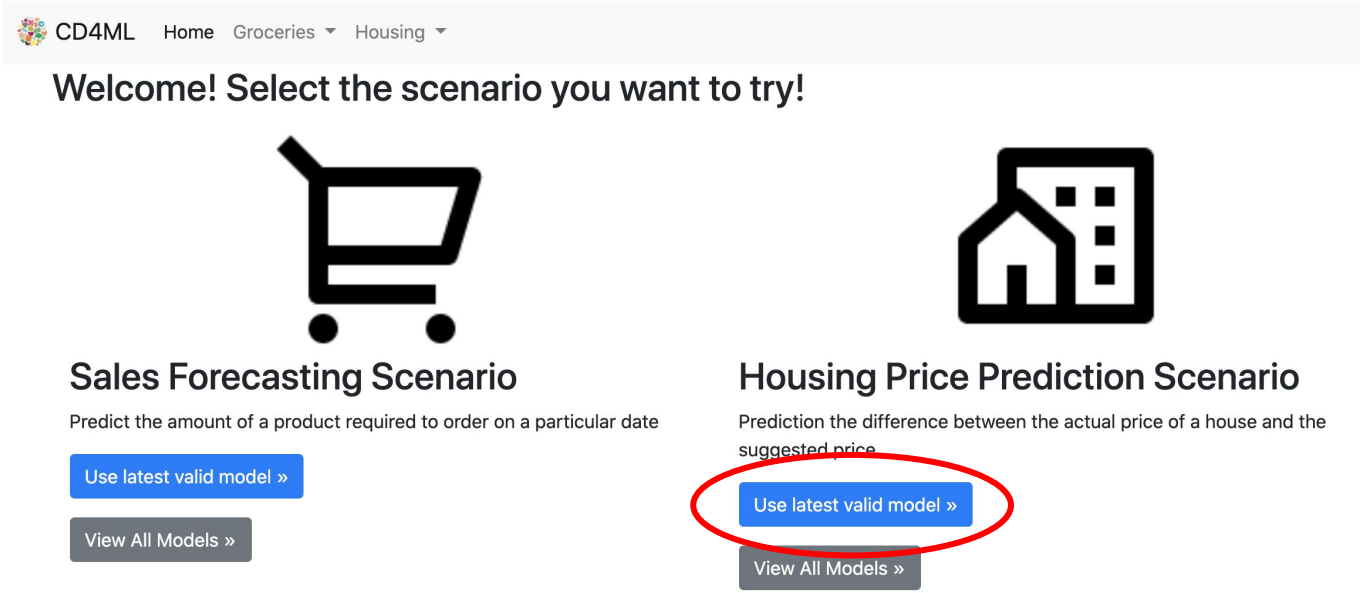




OUR END GOAL JENKINS PIPELINE

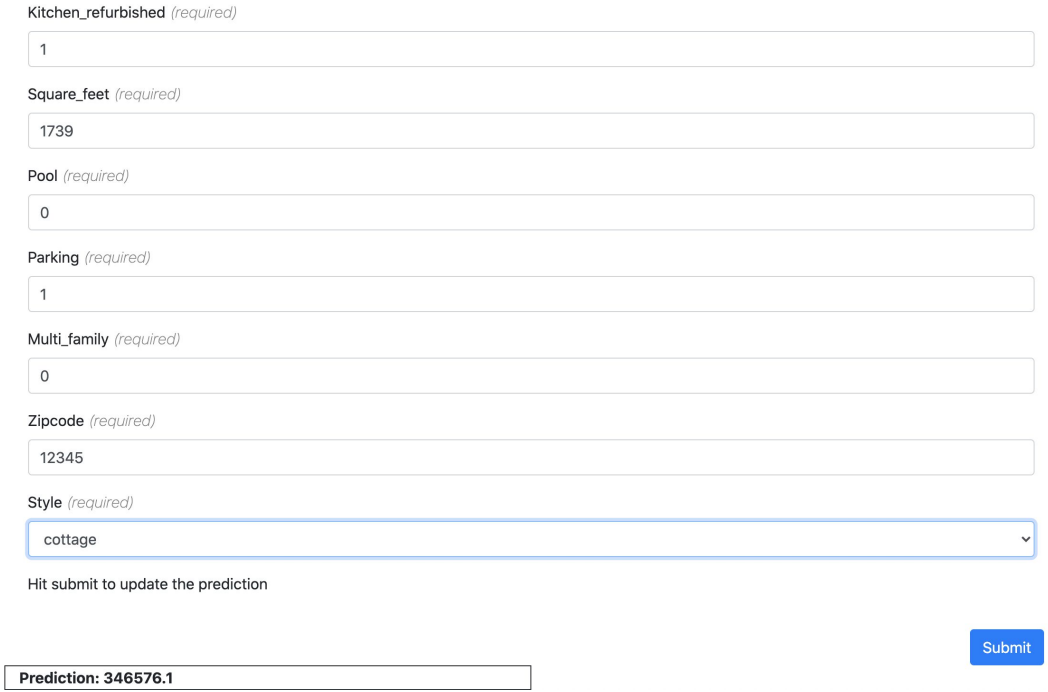


RUN THE WEB APPLICATION (1)



http://localhost:11000/

RUN THE WEB APPLICATION (2)



http://localhost:11000/

STEP BY STEP

1. **Doing the plumbing:** Set up the pipeline and see if it is working

2. **Data Science:** Develop the model and test it (with TDD)
a) Run the ML pipeline
b) Run experiments
c) Change parameters, improve score and monitor models in ML Flow
d) Overview of codebase and design
e) Testing (TDD)

3. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the quality threshold

4. **Monitor** the model “in the wild”

Machine Learning Frameworks Principles

GUIDING PRINCIPLES FOR ML PIPELINES

Attention to good software development principles

The key to quality, flexibility and maintainability is in controlling complexity

- Good design requires modularity and encapsulation
- Integrity of domain boundaries
- Maintainable software requires good code testing not just validation
- Automation, reproducibility including CI/CD

Using Popular ML Frameworks incorrectly can lead to hard-to-maintain software

Common major offenders

- Pandas and other data frame models. Even Spark.
- Notebooks used for pipeline coordination
- GGplot, other visualization libraries which include transformation and aggregations

EXAMPLE OF ABUSING PANDAS

```
def prepare_sales_forecast(idx, sales_hist_dates, orders_all, sales_daily):
    sdate = sales_hist_dates.Date_min[idx]
    sales_default = pd.DataFrame(
        data={'Date': [sdate + timedelta(days=i) for i in range(0, int((last_date - sdate).days))]})
    sales_obs = sales_daily[(sales_daily.location == sales_hist_dates.location[idx]) & (
                sales_daily.factory_code == sales_hist_dates.factory_code[idx])]

    order_cust = orders_all[(orders_all.location == sales_hist_dates.location[idx]) & (
                orders_all.fact_code == sales_hist_dates.fact_code[idx]) & (orders_all.lodge_date < last_date)]
    order_endcust = order_cust[['Date', 'Completion', 'unitSales']]
    order_endcust = order_endcust[(order_endcust.VolUnitsSold > 0)]
    order_endcust_agg = order_endcust.groupby(['component'], as_index=False).agg(
        {'volume': ['mean', 'std', 'count']})
    order_endcust_agg.columns = ['component', 'mean', 'std', 'cnt']
    order_cust = order_endcust.merge(order_endcust_agg, how='left', on=['component'])

    return order_cust, sales_obs, sales_default
```

- Batch oriented
- Hard to make modular, hard to test
- Behavior of library functions hard to predict
- Hard to identify edge cases

Jack of all concerns

EXAMPLE OF ABUSING PANDAS

```
def prepare_sales_forecast(idx, sales_hist_dates, orders_all, sales_daily):
    sdate = sales_hist_dates.Date_min[idx]
    sales_default = pd.DataFrame(
        data={'Date': [sdate + timedelta(days=i) for i in range(0, int((last_date - sdate).days))]})
    sales_obs = sales_daily[(sales_daily.location == sales_hist_dates.location[idx]) & (
            sales_daily.factory_code == sales_hist_dates.factory_code[idx])]

    order_cust = orders_all[(orders_all.location == sales_hist_dates.location[idx]) & (
                orders_all.fact_code == sales_hist_dates.fact_code[idx]) & (orders_all.lodge_date < last_date)]
    order_endcust = order_cust[['Date', 'completion', 'VolUnitsSold']]
    order_endcust = order_endcust[(order_endcust.VolUnitsSold > 0)]
    order_endcust_agg = order_endcust.groupby(['component'], as_index=False).agg(
        {'volume': ['mean', 'std', 'count']})
    order_endcust_agg.columns = ['component', 'mean', 'std', 'cnt']
    order_cust = order_endcust.merge(order_endcust_agg, how='left', on=['component'])

    return order_cust, sales_obs, sales_default
```

Date transformations

Extract data

Feature generation

Column Filtering

Restructuring

Row Filtering

Aggregation

Statistics

Feature generation

Row Filtering

Restructuring

Row Filtering

- Batch oriented
- Hard to make modular, hard to test
- Behavior of library functions hard to predict
- Hard to identify edge cases

EXAMPLE OF CLEAN MODULAR CODE

```
def process(row_in):
    row = {'item_nbr': row_in['item_nbr'],
           'unit_sales': max(0.0, float(row_in['unit_sales'])),
           'date': row_in['date'],
           'year': row_in['year'],
           'month': row_in['month'],
           'day': row_in['day'],
           'class': row_in['class'],
           'family': row_in['family'],
           'perishable': int(row_in['perishable'] == '1'),
           'dayofweek': row_in['dayofweek'],
           'days_til_end_of_data': int(row_in['days_til_end_of_data']),
           'dayoff': int(row_in['dayoff'] == 'True')}

    return row
```

```python
def test_process():
    row_in = {'id': '88219279',
              'date': '2016-08-16',
              'item_nbr': '103520',
              'unit_sales': '10.0',
              'family': 'GROCERY I',
              'class': '1028',
              'perishable': '0',
              'year': '2016',
              'month': '8',
              'day': '16',
              'dayofweek': '1',
              'days_til_end_of_data': '364',
              'dayoff': 'False'}

    row = process(row_in)
    expected = {'item_nbr': '103520',
                'unit_sales': 10.0,
                'date': '2016-08-16',
                'year': '2016',
                'month': '8',
                'day': '16',
                'class': '1028',
                'family': 'GROCERY I',
                'perishable': 0,
                'dayofweek': '1',
                'days_til_end_of_data': 364,
                'dayoff': 0}

    assert row == expected
```

EXAMPLE OF A TEST

Test Driven Development (TDD)

- Write the tests before the code
- Watch the test fail
- Write the code to make the test pass
- Write a new test
- Watch the test fail
- Write code to make all tests pass
- Repeat…

Advantages

- All code functionality will be tested
- Debugging is easier
- Refactoring is easier

SEPARATE STREAMING VERSUS BATCH

With a preference for streaming

Python makes streaming very easy with generators. E.g.

```
stream = csv.DictReader(open(filename, 'r'))
print(next(stream))
```

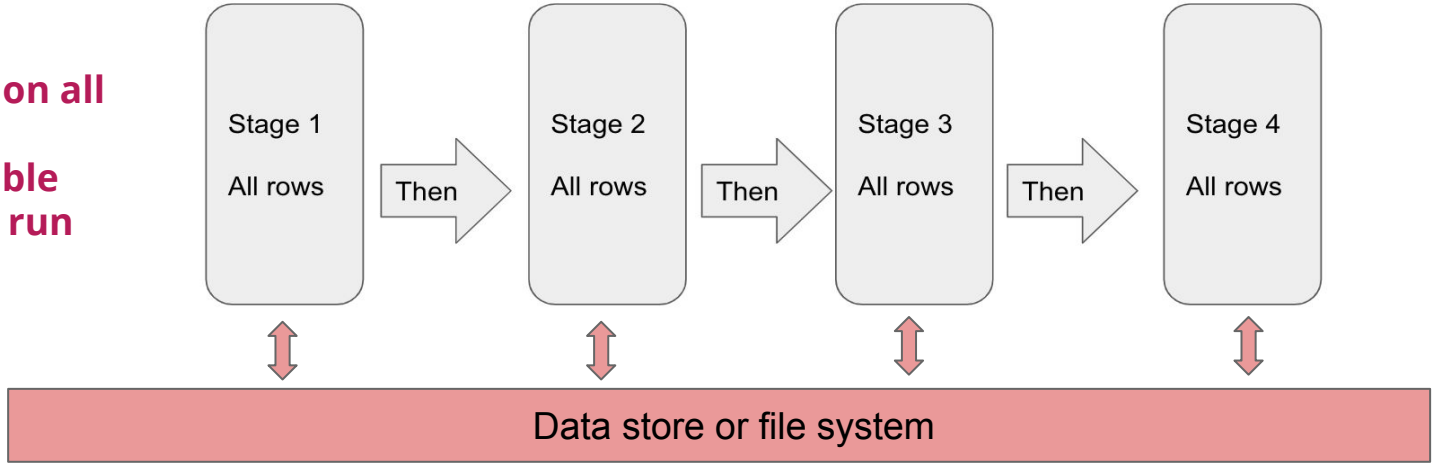
{'id': '88221657', 'date': '2016-08-16', 'item_nbr': '1963838'}

Nice properties of streams

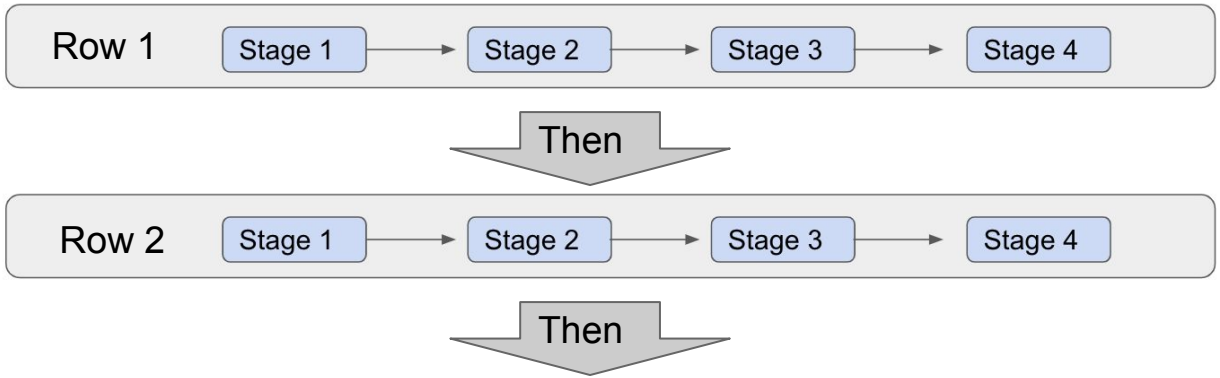
- They are generic. Encapsulates the ultimate source and its properties.
- Stream transformations are simple: stream_2 = (transform(i) for i in stream)
- Encourages modular code and testing and separation of concerns
- Memory efficient
- Leads to higher productivity development / debugging. Depth-first.

Row first, batch stages versus stage-first streaming model

Run Stage 1 on all rows. Write results to table or file. Then run stage 2 etc.



Run all stages on each row before moving on to second row.



Advantages of stage-first streaming model

- Data access / IO can be more easily be separated from the transformation logic. All logic runs the same wherever the stream of rows comes from.
- Data sources can be migrated trivially. File -> DB -> Spark -> Kafka
- Code logic can all be developed from a simulated or reduced-size data set in any environment, even local (i.e. laptop). No fighting the system to develop. Much higher productivity. Separation of concerns.
- Easier to test and debug
  - What if Row # 580,005 fails at Stage 3? Can write a single unit test that fails for that row and fix code to make it pass without having to rerun the whole pipeline and wait hours to reproduce failure. Fast feedback through all stages. Easier to identify root cause of problem.
- Usually more performant. Reduces the tendency to rely on SQL or database to perform transformation logic. In memory processing is usually faster. At least, allows easier performance tuning.

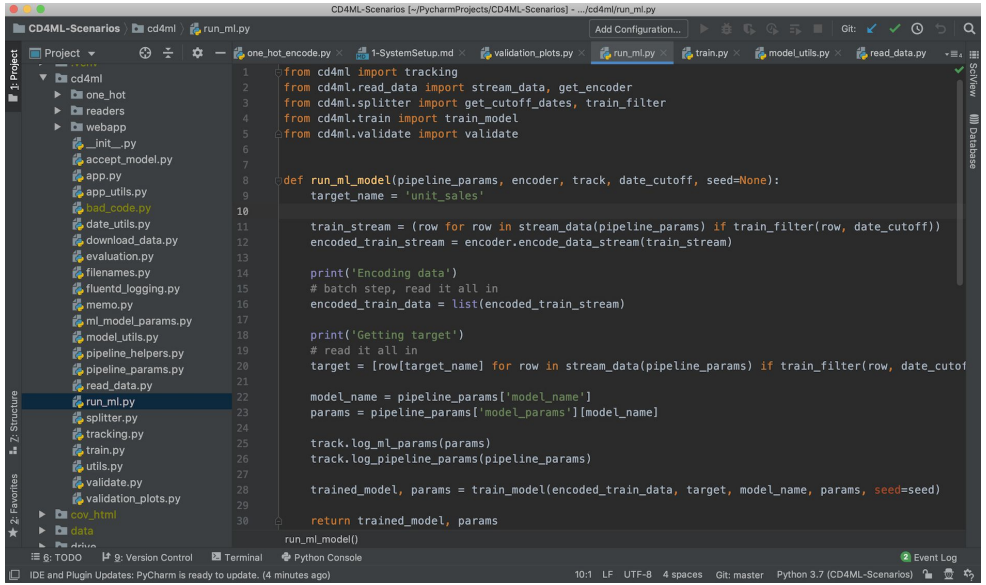
STEP BY STEP

1. **Doing the plumbing:** Set up the pipeline and see if it is working

2. **Data Science:** Develop the model and test it (with TDD)
a) Run the ML pipeline
b) Run experiments
c) Change parameters, improve score and monitor models in ML Flow
d) Overview of codebase and design
e) Testing (TDD)

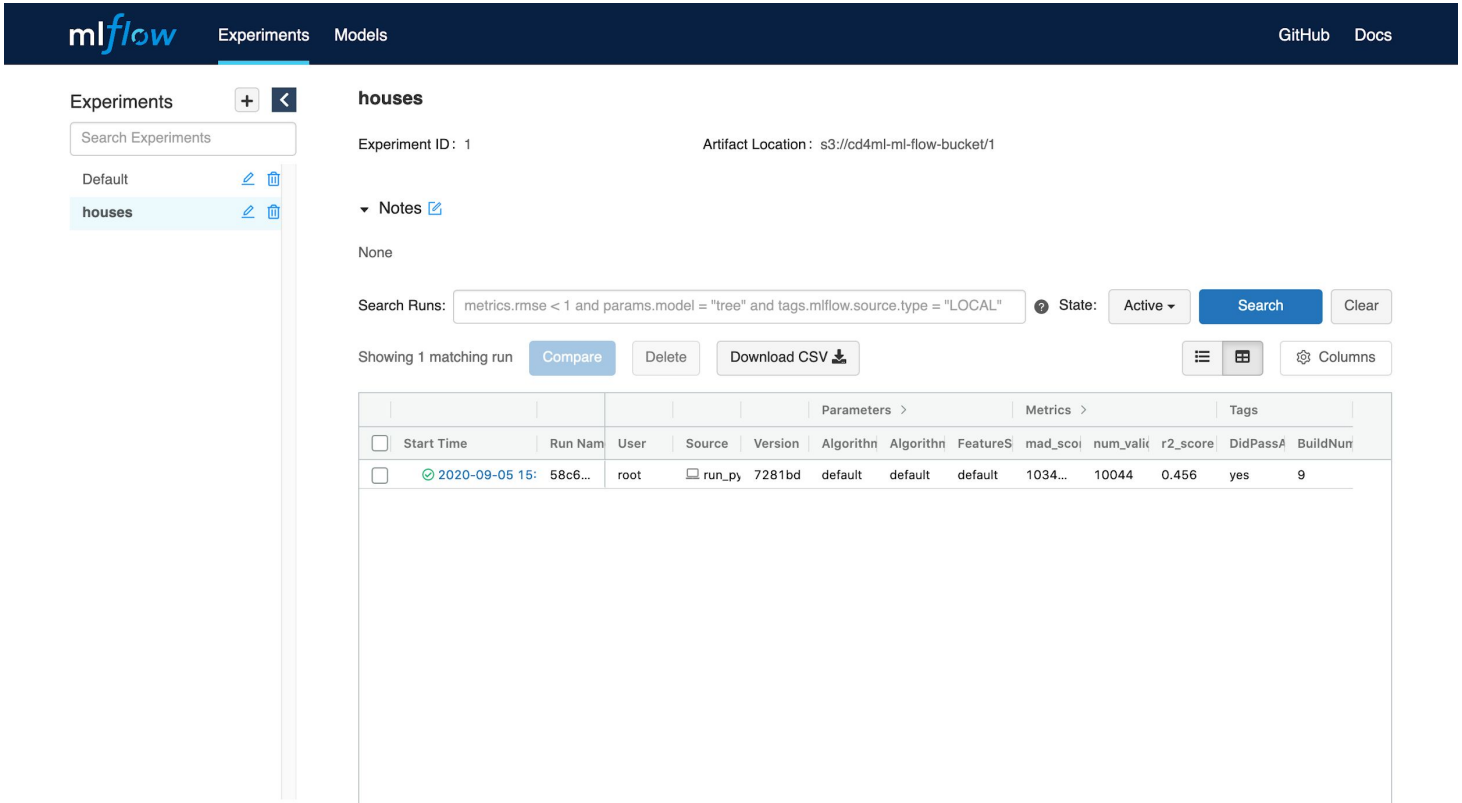
3. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the quality threshold
4. **Monitor** the model “in the wild”

OVERVIEW OF THE CODE BASE AND SCENARIOS



- Structure of code base
- How to run the pipeline After making changes
- How to view results of models and keep track of changes
- Changing algorithms, params etc and not causing chaos

ML Flow



http://localhost:11000/

STEP BY STEP

1. **Doing the plumbing:** Set up the pipeline and see if it is working

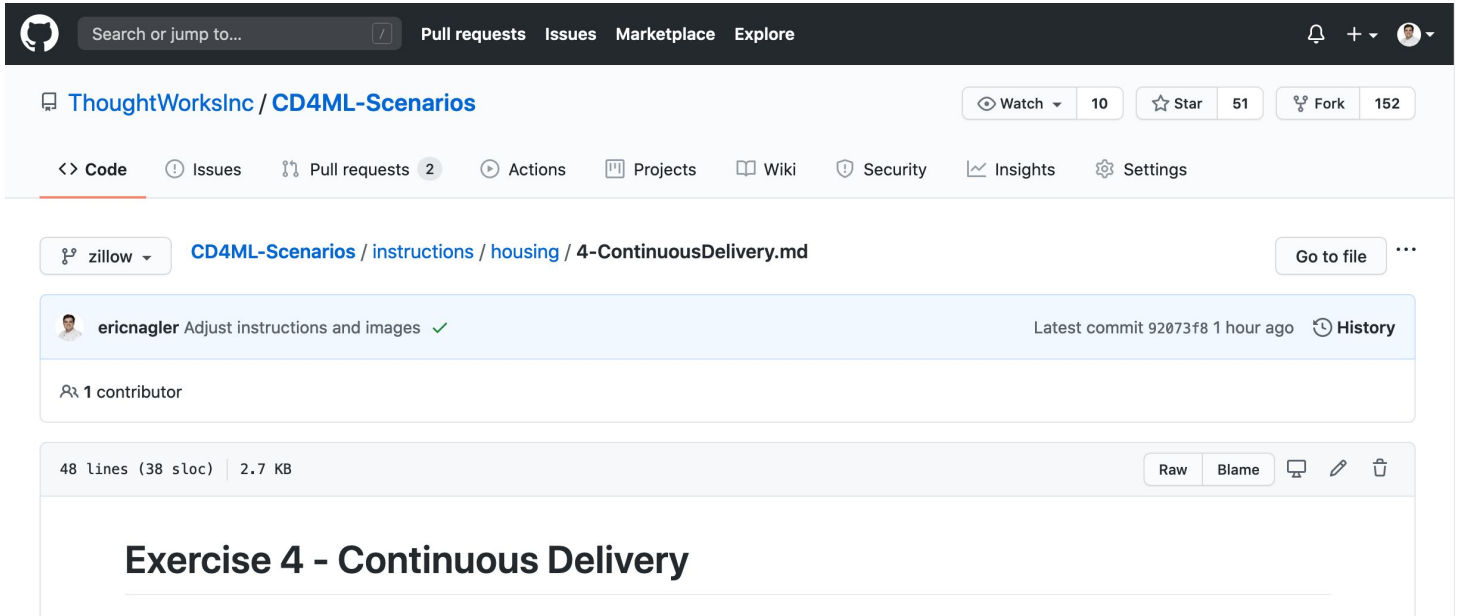
2. **Data Science:** Develop the model and test it (with TDD)
    a. Run the ML pipeline
    b. Run experiments
    c. Change parameters, improve score and monitor models in ML Flow
    d. Overview of codebase and design
    e. Testing (TDD)

3. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the quality threshold

4. **Monitor** the model “in the wild”

CONTINUOUS DEPLOYMENT

- Navigate to instructions/4-ContinuousDeployment.md



CONTINUOUS DEPLOYMENT

"Continuous Deployment is the ability to get changes of all types — including new features, configuration changes, bug fixes and experiments — into production, or into the hands of users, safely and quickly in a sustainable way."

- Ensure that the experiment that we just ran with lasso can be promoted to production using the "default" algorithm.
- Change the minimum acceptance criteria to simulate an unexpected model performance drop and watch the jenkins pipeline capture the error
- Transfer the contents from lasso big_alpha.json to default.json
- We will also observe that this will not impact the model running in production

Continuous Deployment

How are we checking for performance (accept_model.py)

```
def is_model_accepted(model_id):
    model_files = get_model_files(model_id)
    params = get_json(model_files['ml_pipeline_params'])

    metric_name = params['acceptance_metric'].replace("'", "")
    threshold_min = float(params['acceptance_threshold_min'])
    threshold_max = float(params['acceptance_threshold_max'])

    metrics = get_json(model_files['model_metrics'])
    metric_value = metrics[metric_name]

    accepted = threshold_min <= metric_value <= threshold_max

    message = get_message(model_id, metric_name, metric_value,
                          threshold_min, threshold_max, accepted)

    return accepted, message
```

Retrieve the machine learning pipeline parameters

Retrieve the r2 score of the model

Check the that the threshold score is within range

STEP BY STEP

1. **Doing the plumbing:** Set up the pipeline and see if it is working

2. **Data Science:** Develop the model and test it (with TDD)
   a. Run the ML pipeline
   b. Run experiments
   c. Change parameters, improve score and monitor models in ML Flow
   d. Overview of codebase and design
   e. Testing (TDD)

3. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the quality threshold

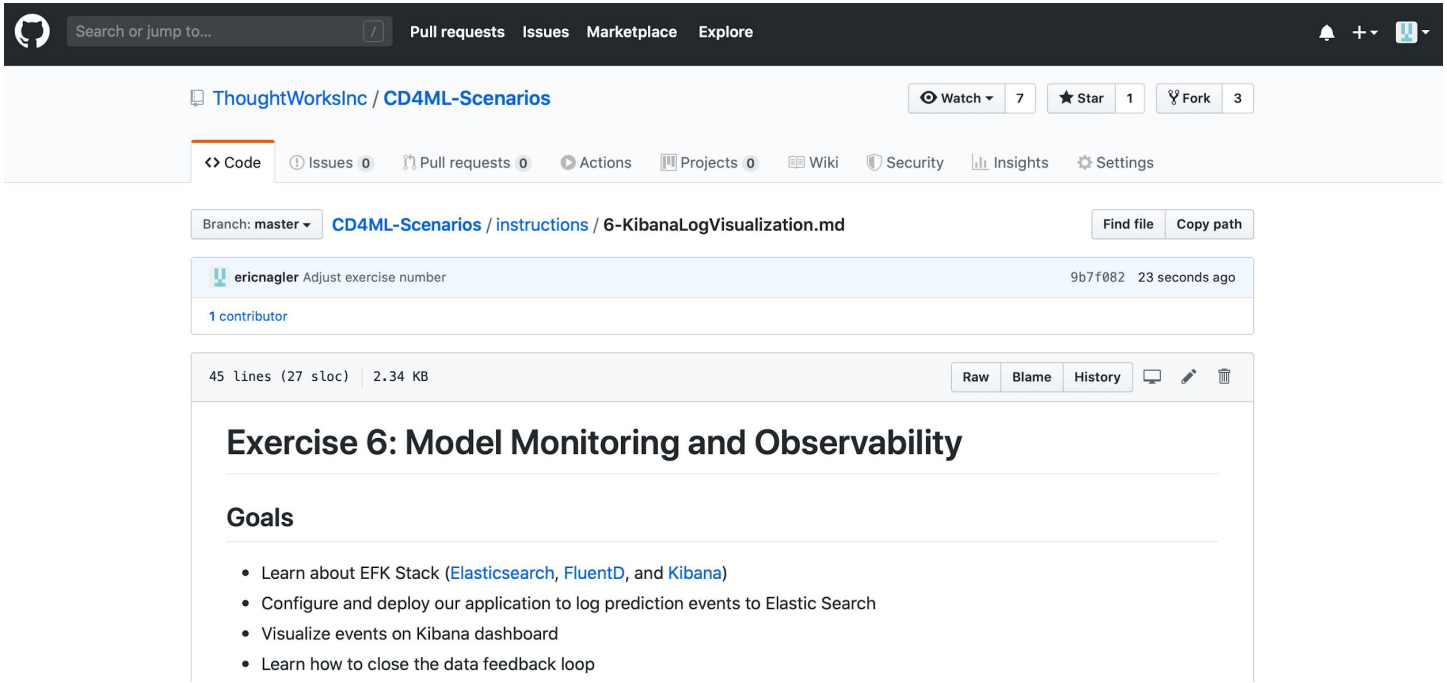
4. **Monitor** the model “in the wild”

Technical Details of the setup of fluentd, elasticsearch and kibana (EFK Stack)

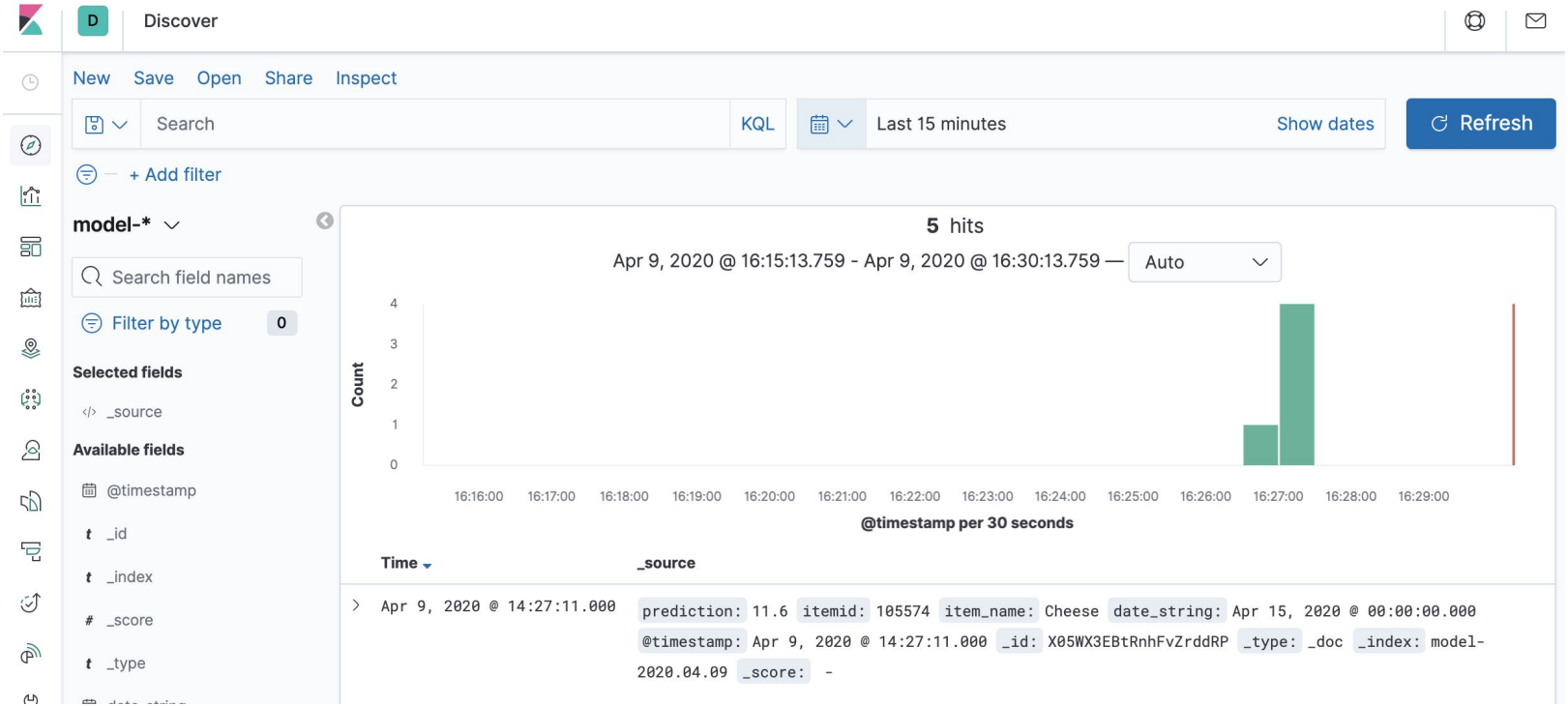
- **FluentD** is a open source data collector that allows for unified logging. FluentD allows for collection and forwarding of logs from any application into whatever service you like. In our case, we collect the logs from the model and forward them to ElasticSearch.
- **ElasticSearch** is a distributed JSON search and analytics engine allowing rapid search and log ingest. This allows us to index our models for monitoring and analytics. These records are then visualized in Kibana.
- **Kibana** is a visualization tool for elasticsearch allowing for data discovery, exploration, visualization and dashboarding.
- **These tools enable you to understand:**
    - What your customers are searching for and asking your model for
    - What your model is inferencing so you can track it's performance

SETUP fluentd, elasticsearch and kibana

- Navigate to instructions/housing/5-KibanaLogVisualization.md page on github to setup fluentd, elasticsearch and kibana



See prediction of your model in kibana



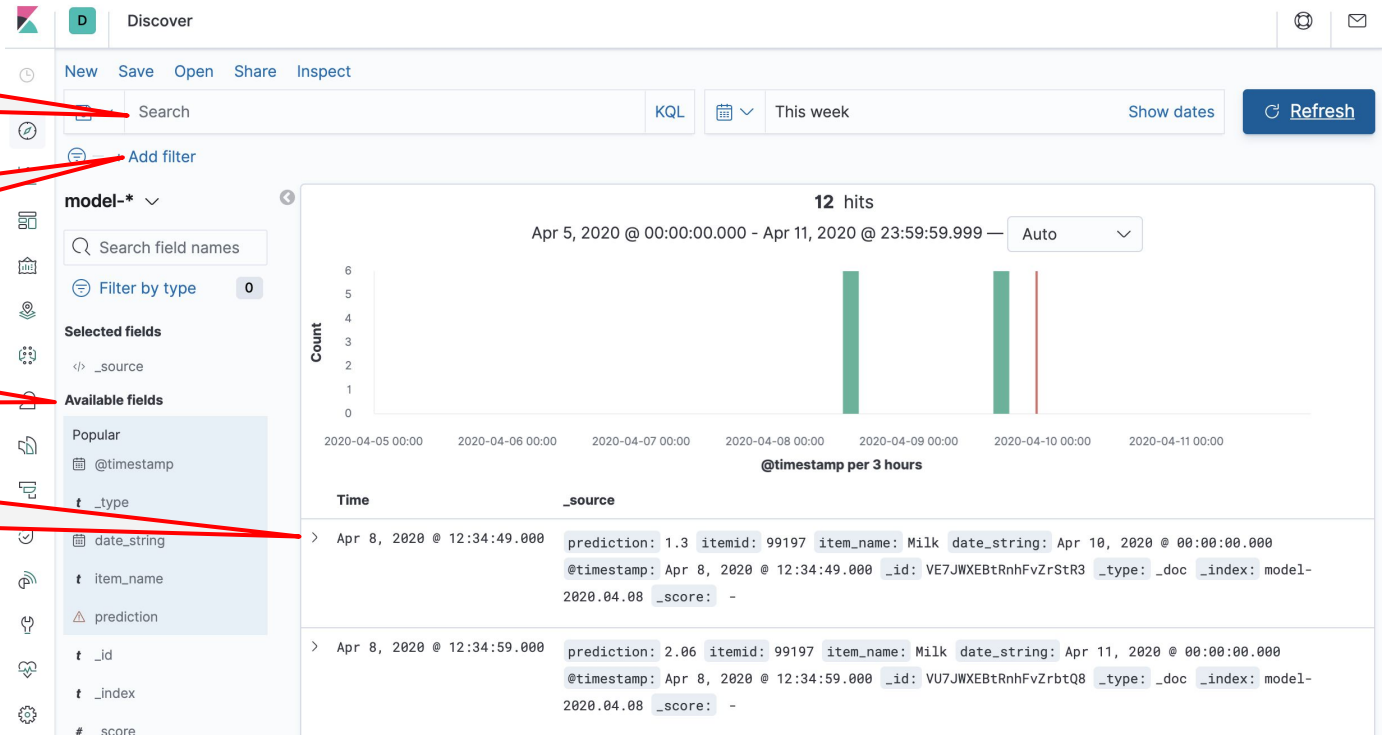
Exercise

Search predictions

Filter predictions

Select fields

Inspect table and json file



WRAP UP

WRAP UP

- You learned:
  - The Principles of CD4ML
  - The components of a tech stack you need to run CD4ML
  - Data Science best practice in the industry
  - Bringing ML model into production
  - Run CD4ML on your laptop!

- Continue the conversation in our Google group!
  - https://groups.google.com/a/thoughtworks.com/g/cd4ml-webinar

- Reach out and ask questions on slack channel or with a direct message, we would love to hear your feedback!

REFERENCES / LINKS

- **Github**

  https://github.com/ThoughtWorksInc/CD4ML-Scenarios

- **Google Group**

  https://groups.google.com/a/thoughtworks.com/g/cd4ml-webinar

- **Articles**

  https://martinfowler.com/articles/cd4ml.html

  https://www.thoughtworks.com/insights/articles/intelligent-enterprise-series-cd4ml

  https://www.thoughtworks.com/continuous-delivery-for-machine-learning

  https://opendatascience.com/continuous-delivery-for-machine-learning/

QUESTIONS

THANK YOU