Continuous Delivery for Machine Learning - ODSC East 2020

Christoph Windheuser - cwindheu@thoughtworks.com
David Johnson - 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

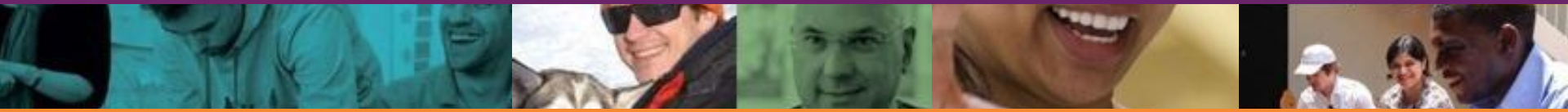
eric.nagler@thoughtworks.com

www.thoughtworks.com

ThoughtWorks®

7000 technologists with 40 offices in 14 countries

Partner for technology driven business transformation



Atlanta - Chicago - Dallas - Denver - New York - San Francisco - Toronto

100+

books written

#1

in Agile and Continuous Delivery

ODSC East, April 15th, 2020

ODSC East, April 15th, 2020

TECHNIQUES

Continuous delivery for machine learning (CD4ML) models #8

TRIAL

Agenda

1. Setting up the local environment on your laptop (20 min)
2. Continuous Delivery for Machine Learning (CD4ML) (40 min)
3. Let's start: Running through the different steps of the process (2h)

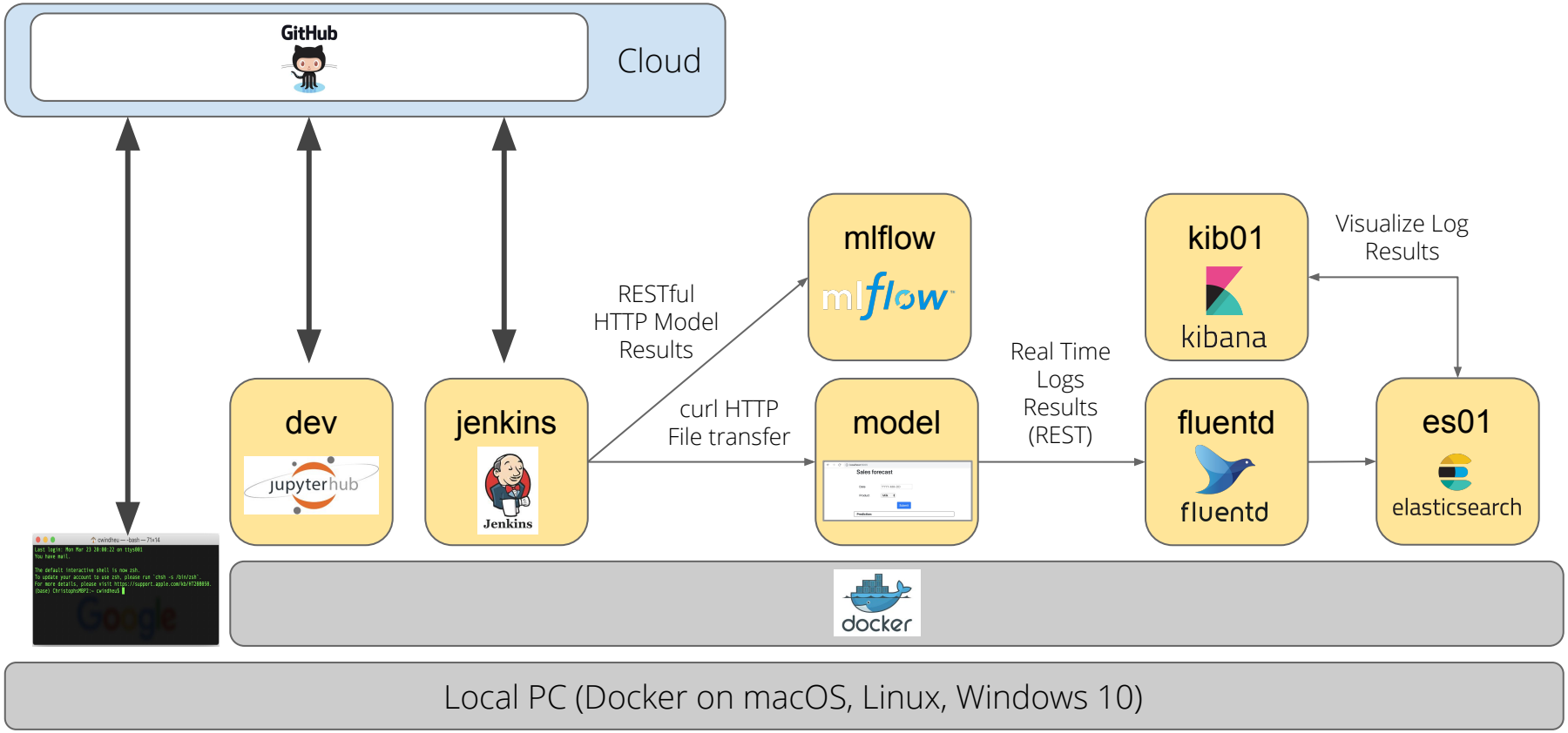
Slack: #wed_christoph-windheuser-david-johnston-eric-nagler_data-science-best-practices

Setting up your local environment

Prerequisites

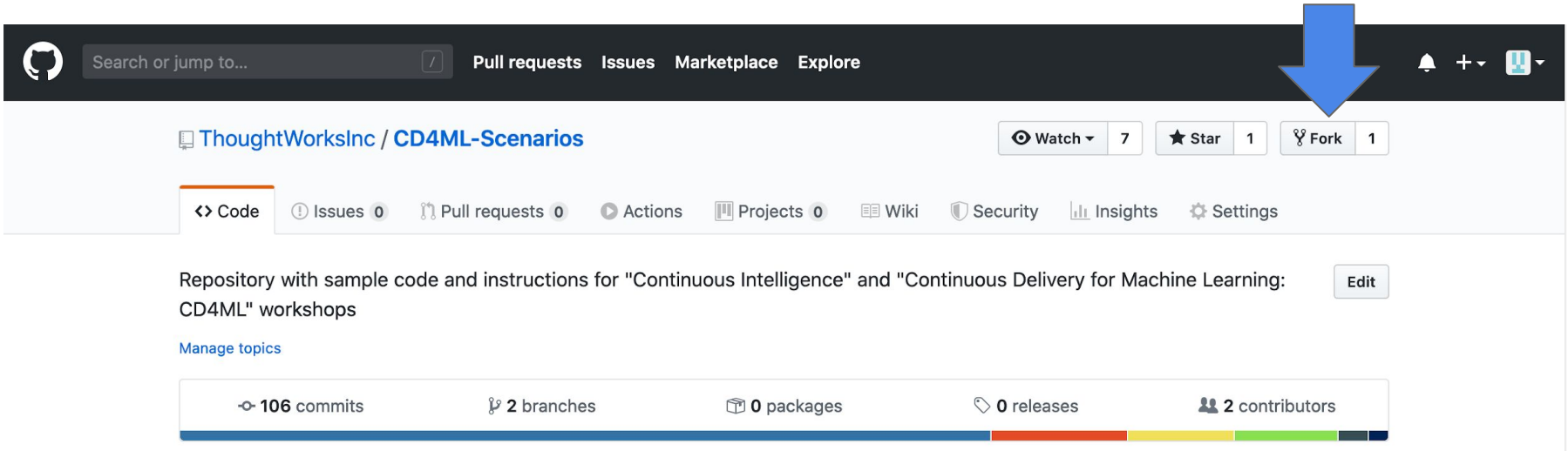
- PC with Mac OS, Windows 10 or Linux
- Docker installed on your PC
- Min 20 GB free space on your local hard drive
- Github account
- Git client installed on your machine
- Enable your slack account:
  #wed_christoph-windheuser-david-johnston-eric-nagler_data-science-best-practices

The Workshop Infrastructure



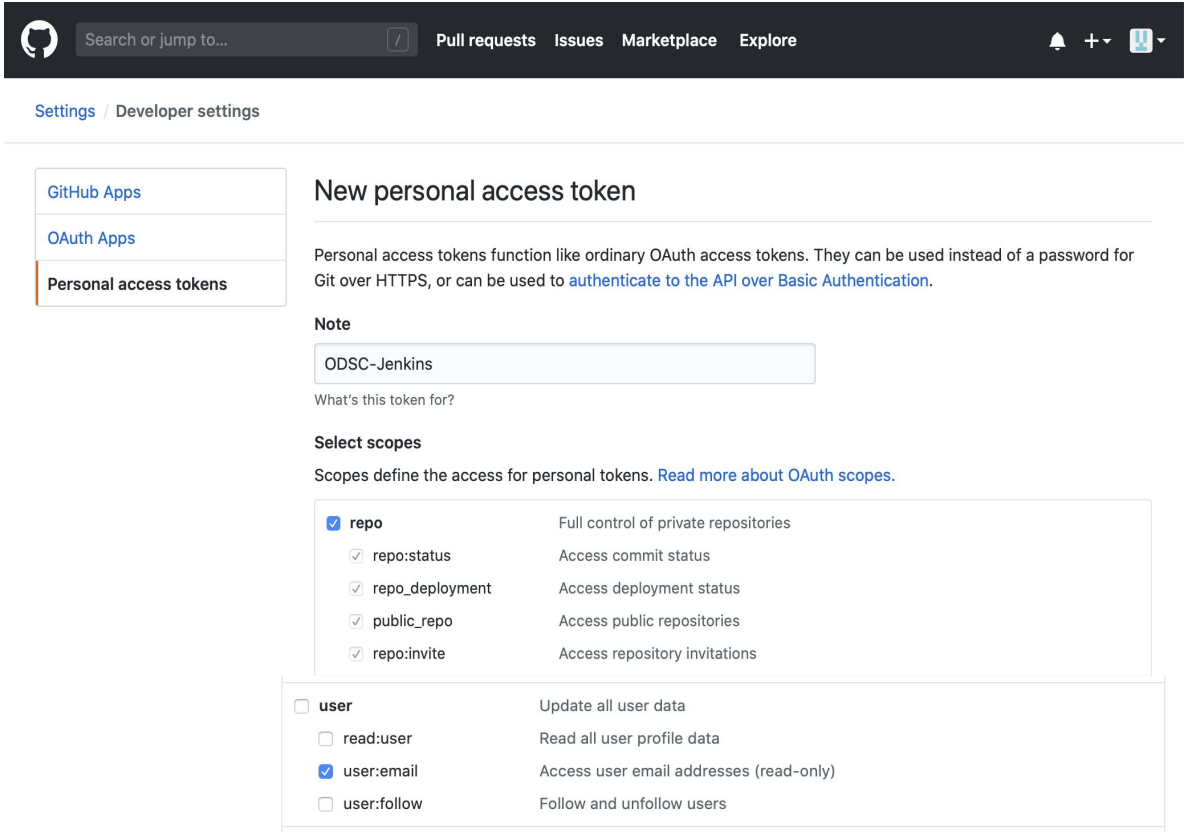
Installation (1)

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



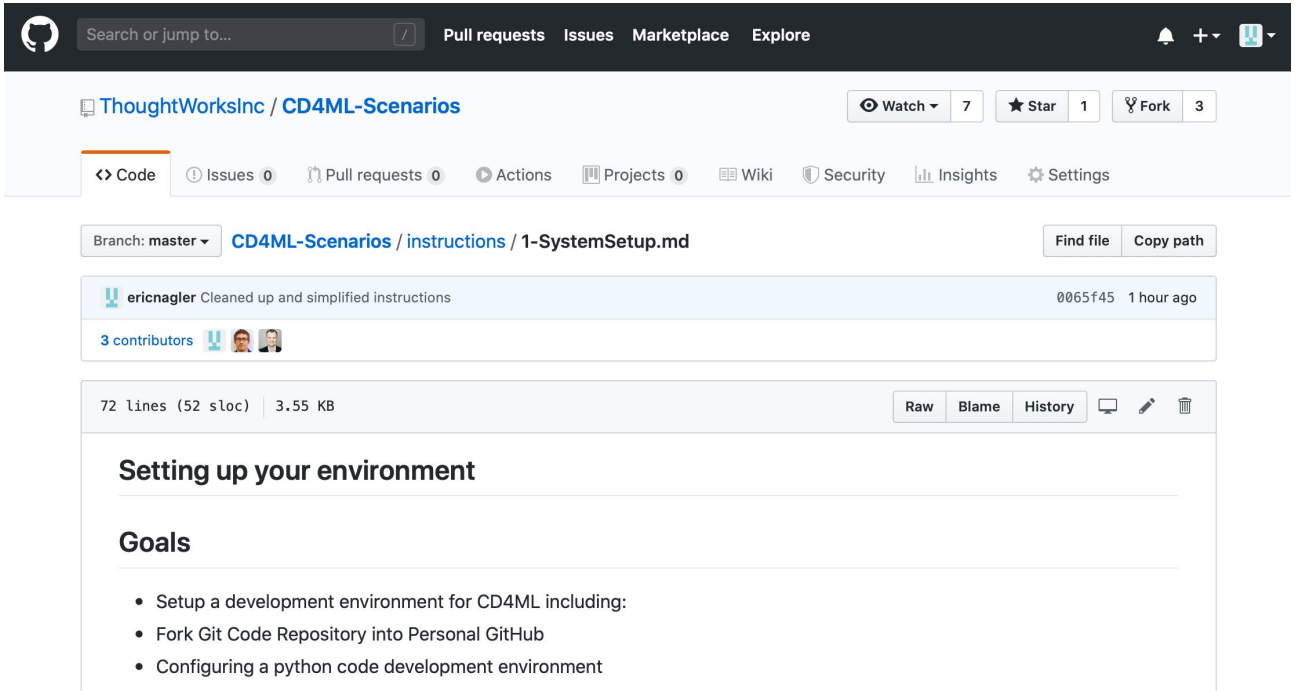
Installation (2)

- 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



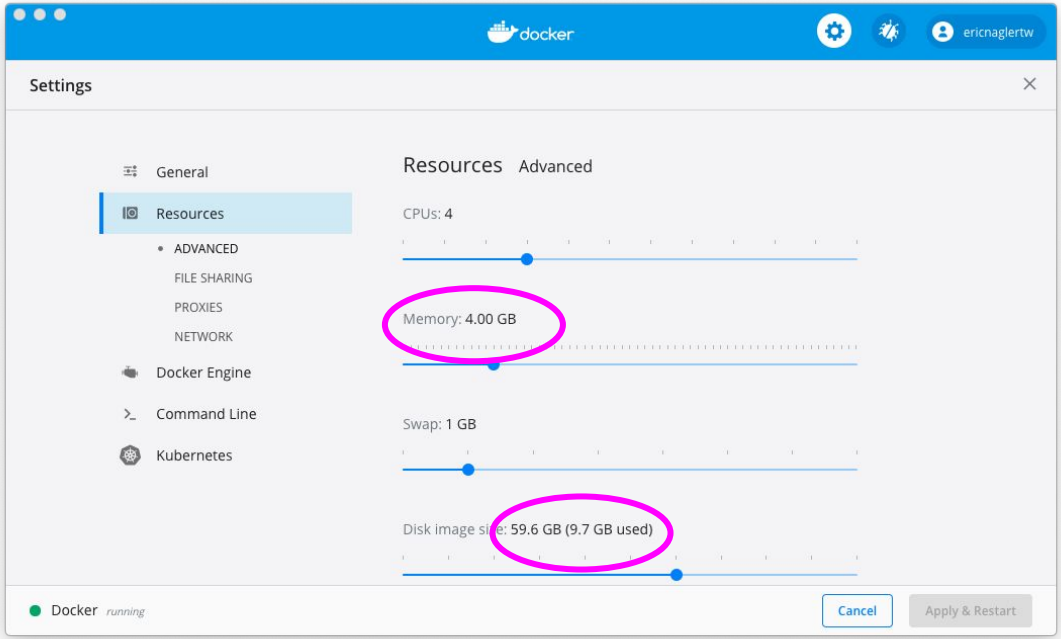
Installation (3)

- 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)

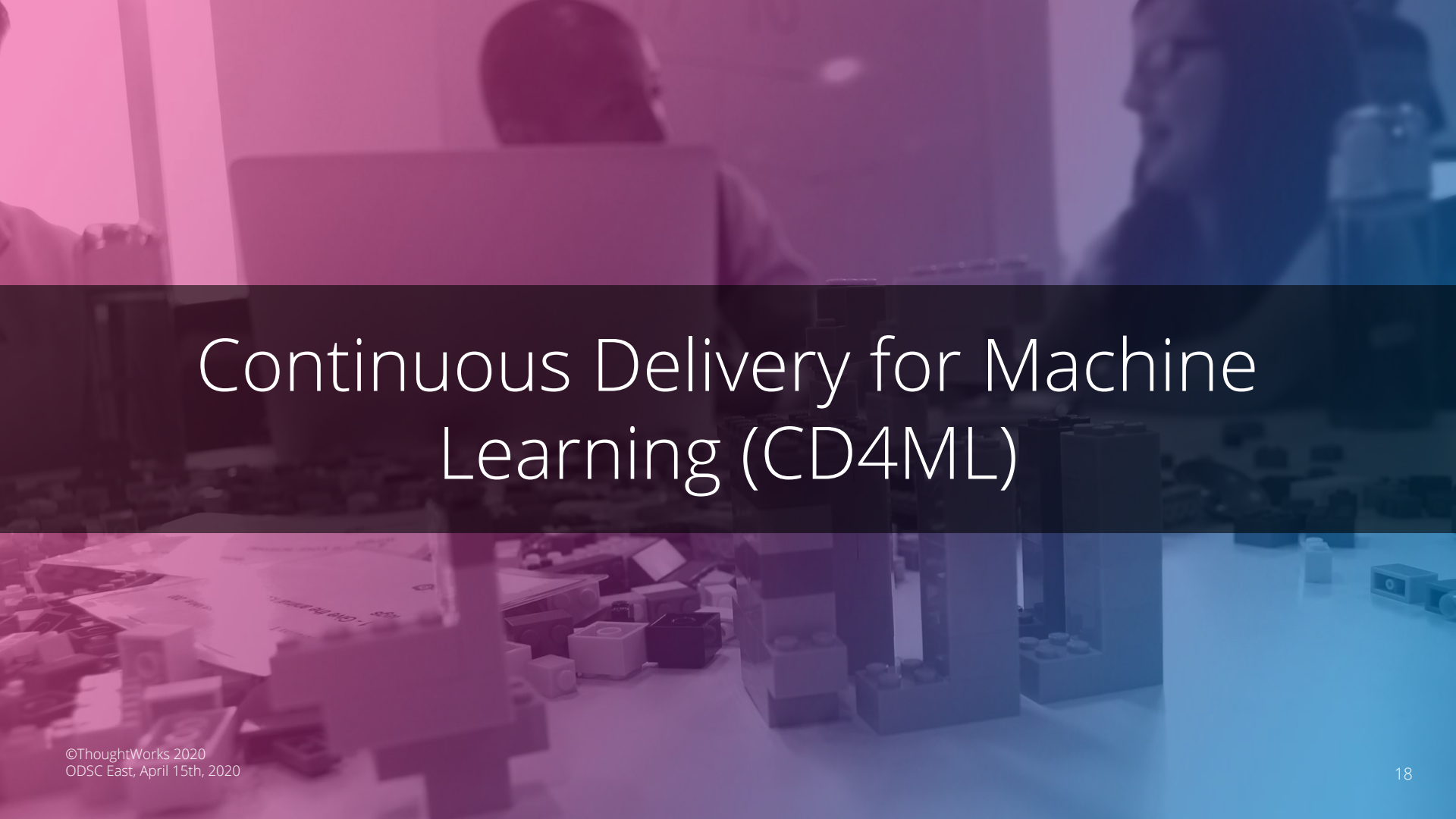
Configuring your jenkins password

- 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.

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
```



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

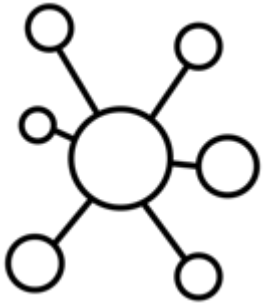
CD4ML isn't a technology or a tool; it is a practice and a set of principles. Quality is built into software and improvement is always possible.

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

+



Model

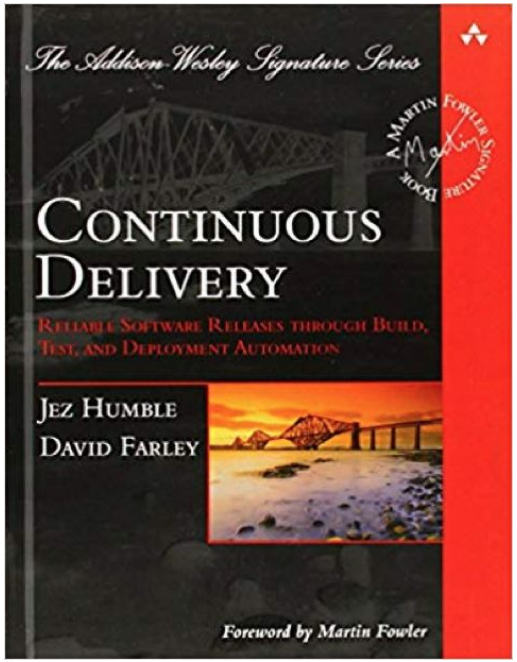
+

Code

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

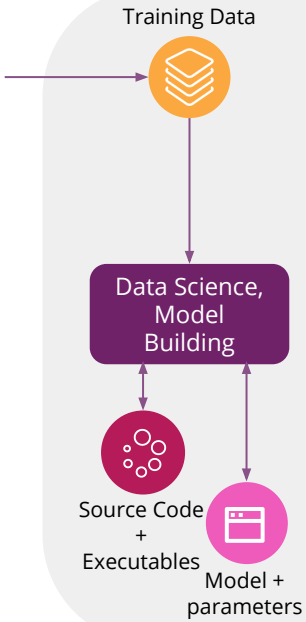
*"Continuous Delivery is the ability to get **changes of all types** — including new features, configuration changes, bug fixes and experiments — into production, or into the hands of users, **safely and quickly** in a **sustainable way.**"*

- 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

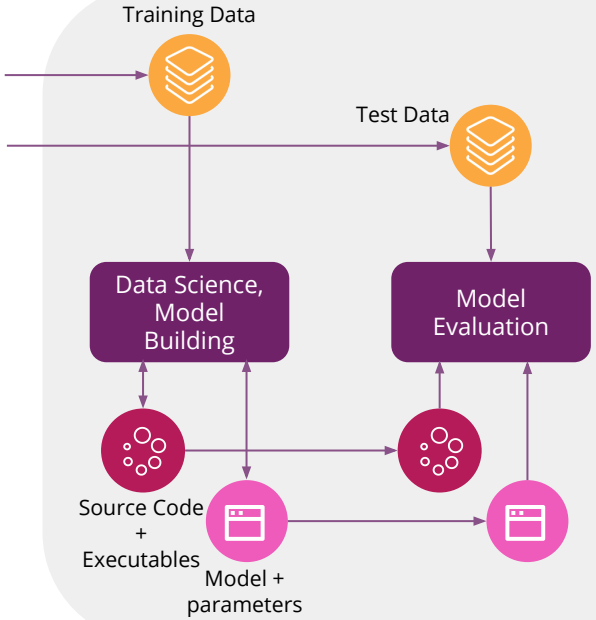
Discoverable and Accessible Data



CD Tools and Repositories

HOW DOES IT WORK

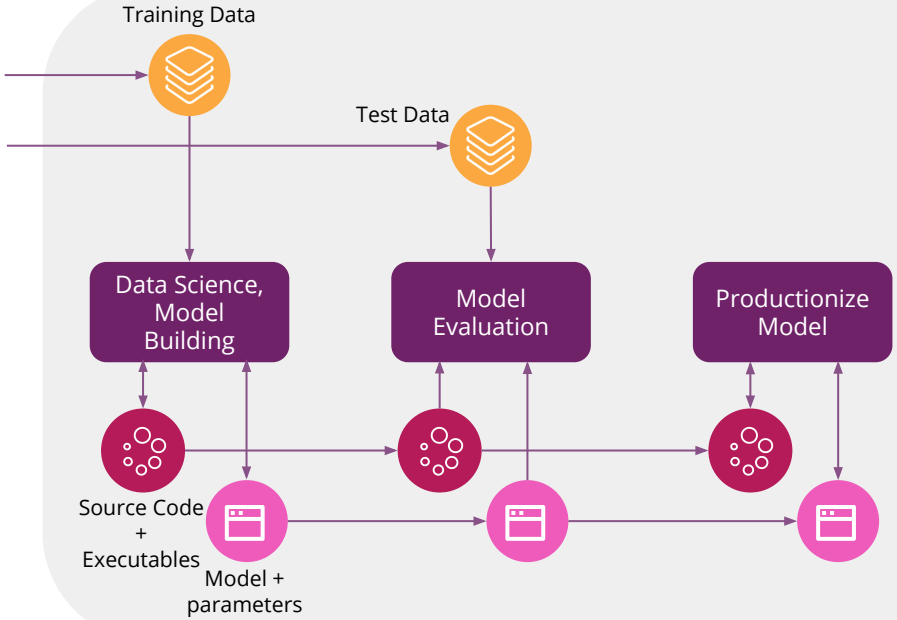
Discoverable and Accessible Data



CD Tools and Repositories

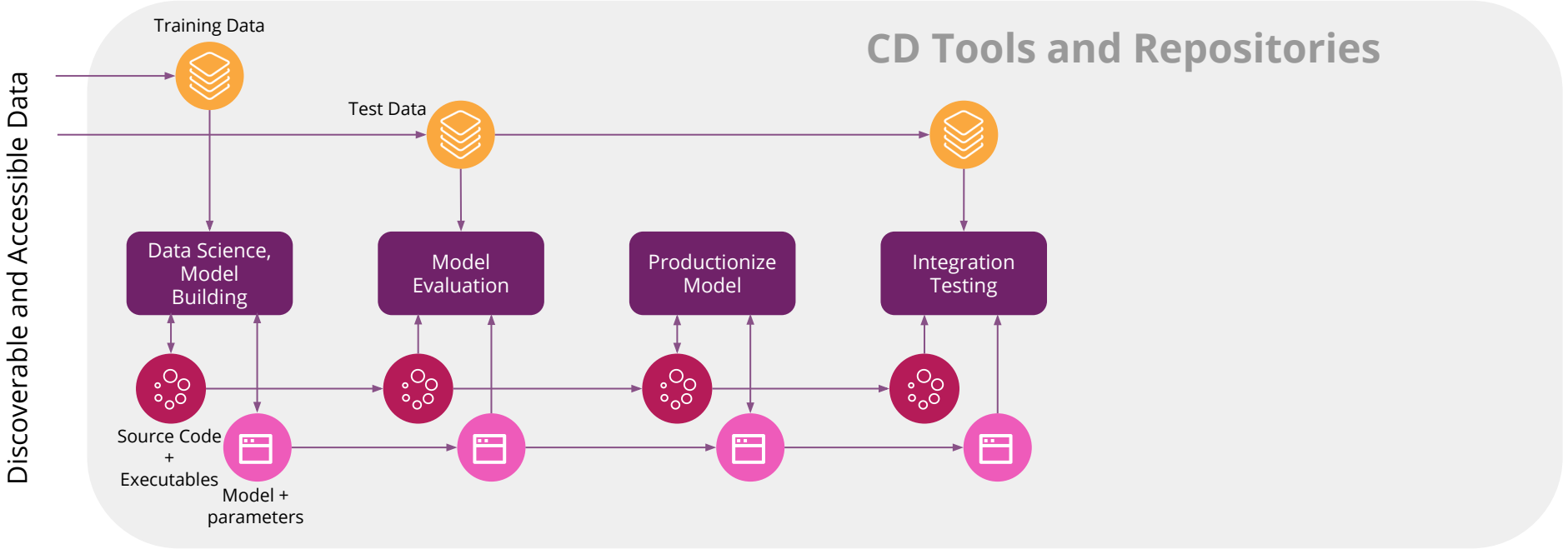
HOW DOES IT WORK

Discoverable and Accessible Data

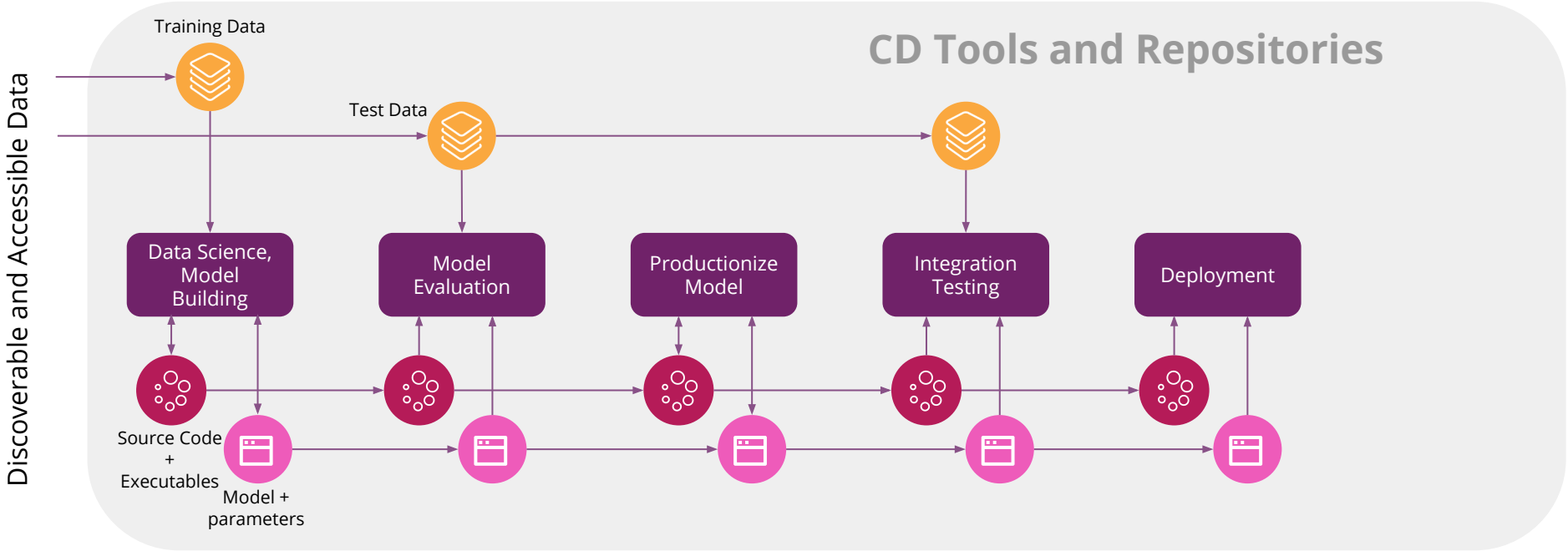


CD Tools and Repositories

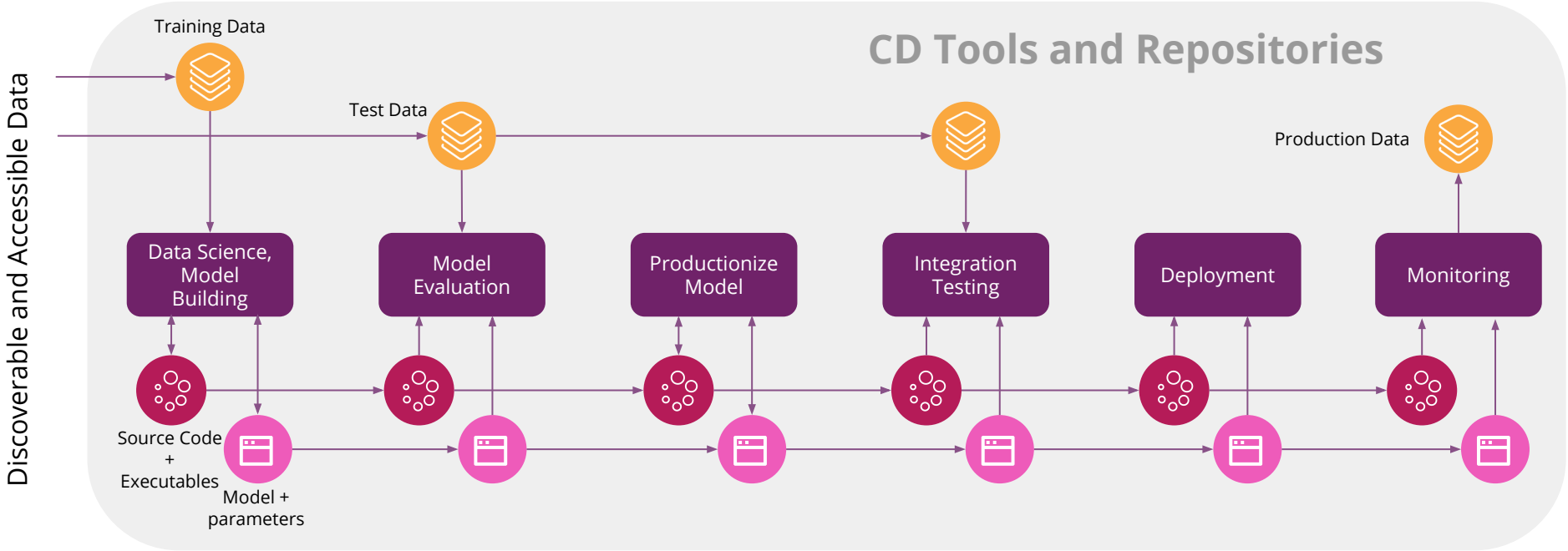
HOW DOES IT WORK



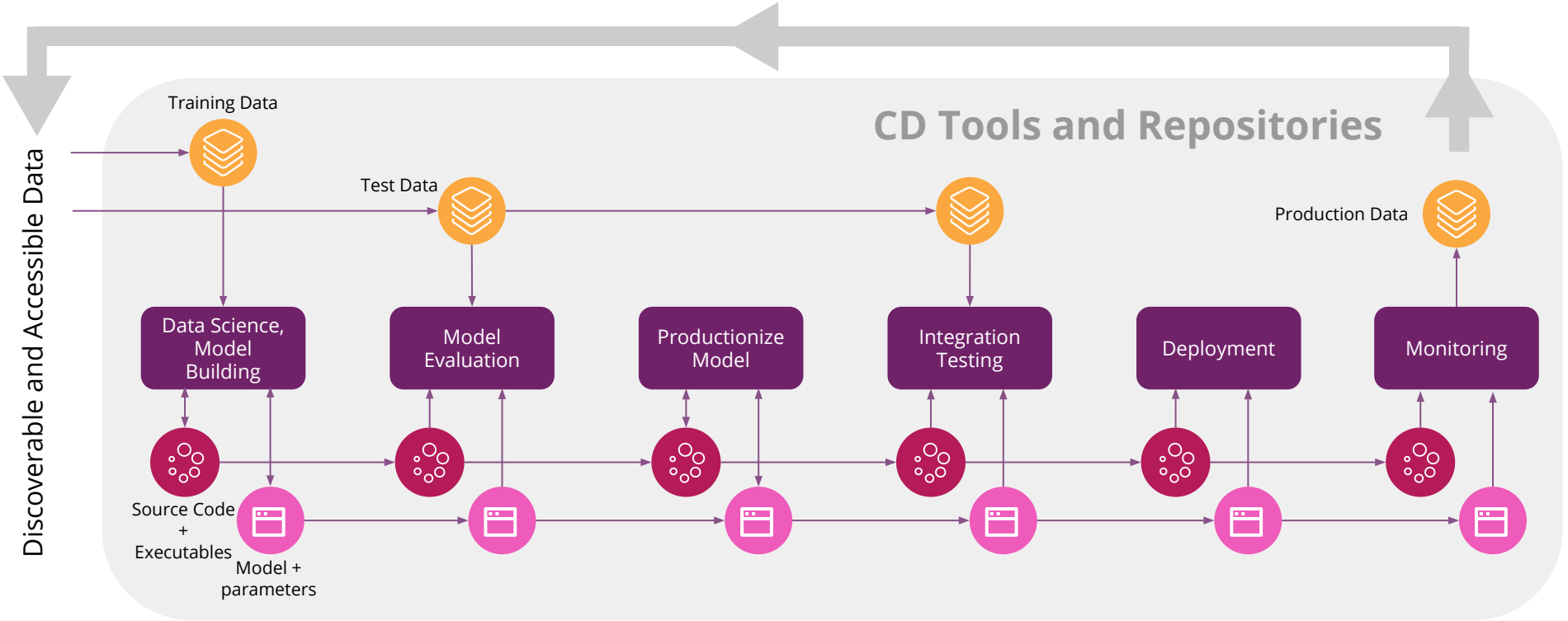
HOW DOES IT WORK



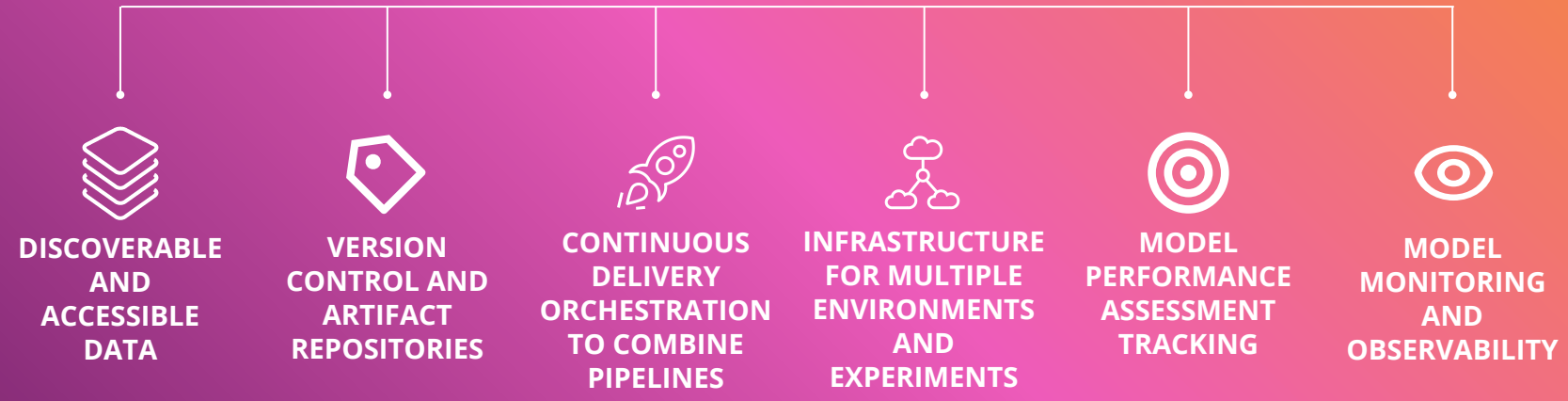
HOW DOES IT WORK



HOW DOES IT WORK



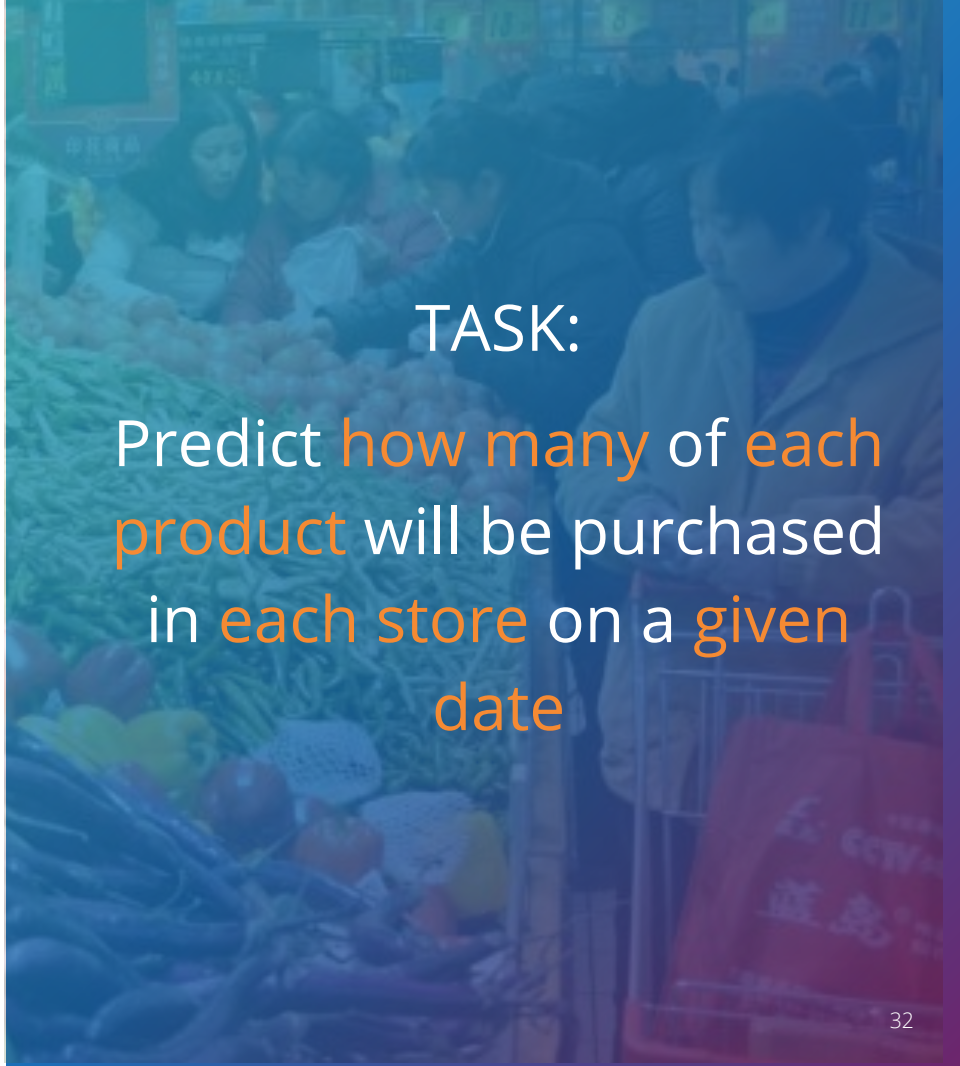
WHAT DO WE NEED IN OUR CD4ML STACK?



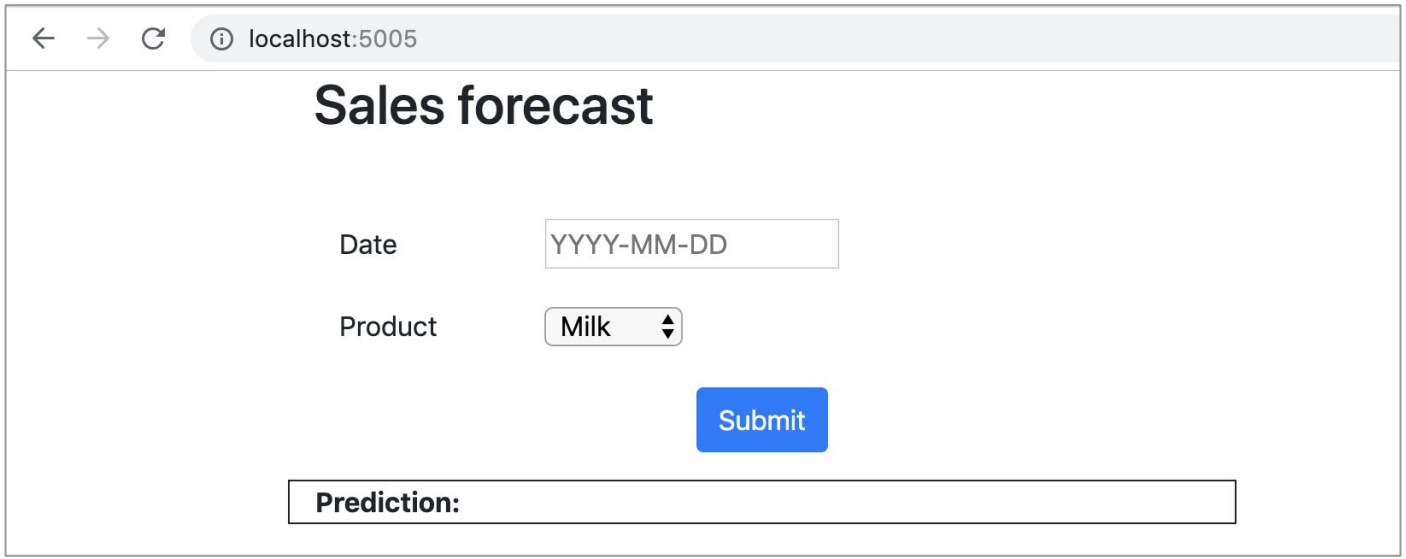
THE BUSINESS PROBLEM: SALES FORECASTING FOR GROCERY RETAILER

Make predictions based on data from:

- 4,000 items
- 50 stores
- 125,000,000 sales transactions
- 4.5 years of data



WE WILL BUILD A SIMPLIFIED WEB APPLICATION



STEP BY STEP

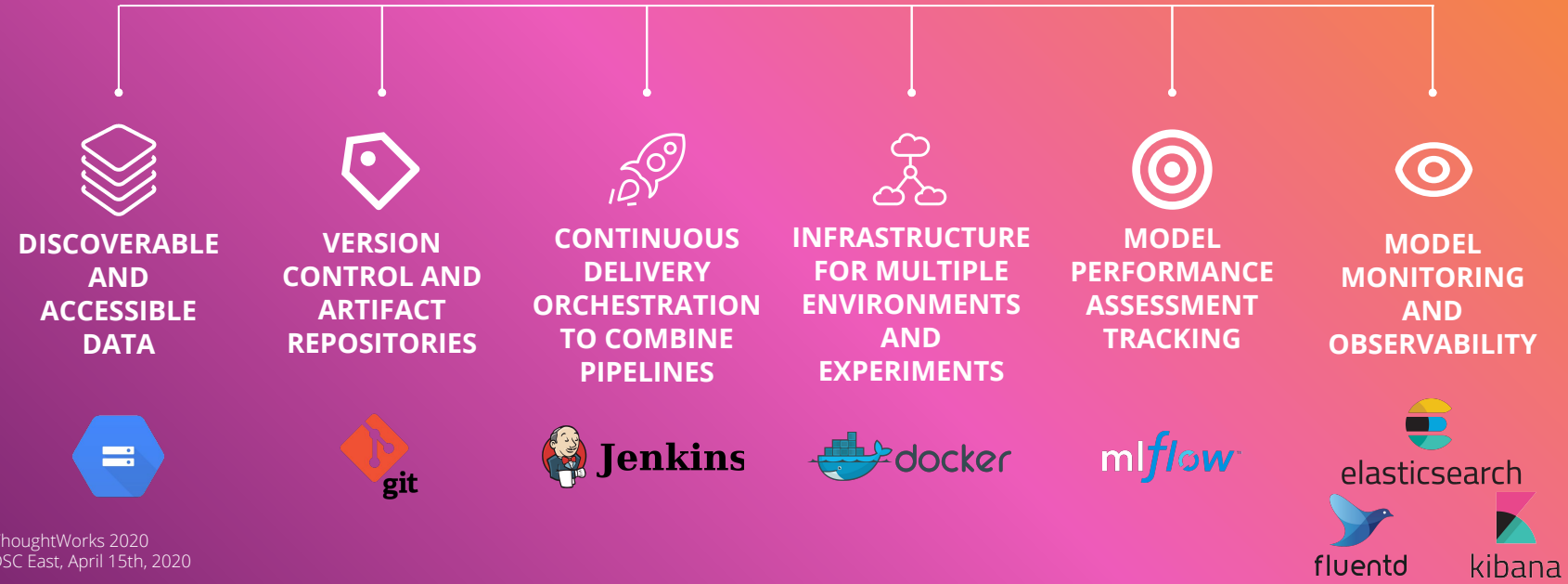
1. **Doing the plumbing:** Set up the pipeline and see if it is working
2. **Data Science:** Develop the model and test it (with Test-Driven Development - TDD)
3. **Machine Learning Engineering:** Improve the model in several steps and monitor the results
4. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the test
5. **Undo changes:** Roll back in time consistently with all artifacts
6. **Our app in the wild:** Monitor it in production with fluentd, elasticsearch and kibana

STEP BY STEP

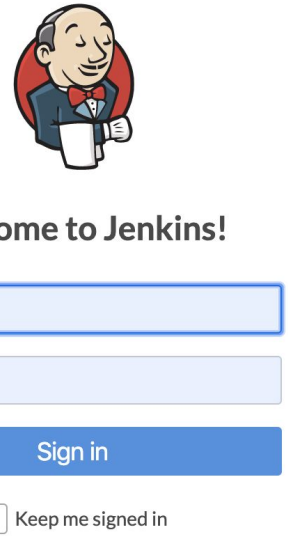
1. **Doing the plumbing:** Set up the pipeline and see if it is working (CW, EN)
2. **Data Science:** Develop the model and test it (with TDD) (DJ)
3. **Machine Learning Engineering:** Improve the model in several steps and monitor the results (DJ)
4. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the test (EN)
5. **Undo changes:** Roll back in time consistently with all artifacts (EN)
6. **Our app in the wild:** Monitor it in production with fluentd, elasticsearch and kibana (CW, EN)

WHAT WE WILL USE IN THIS WORKSHOP

There are many options for tools and technologies to implement CD4ML

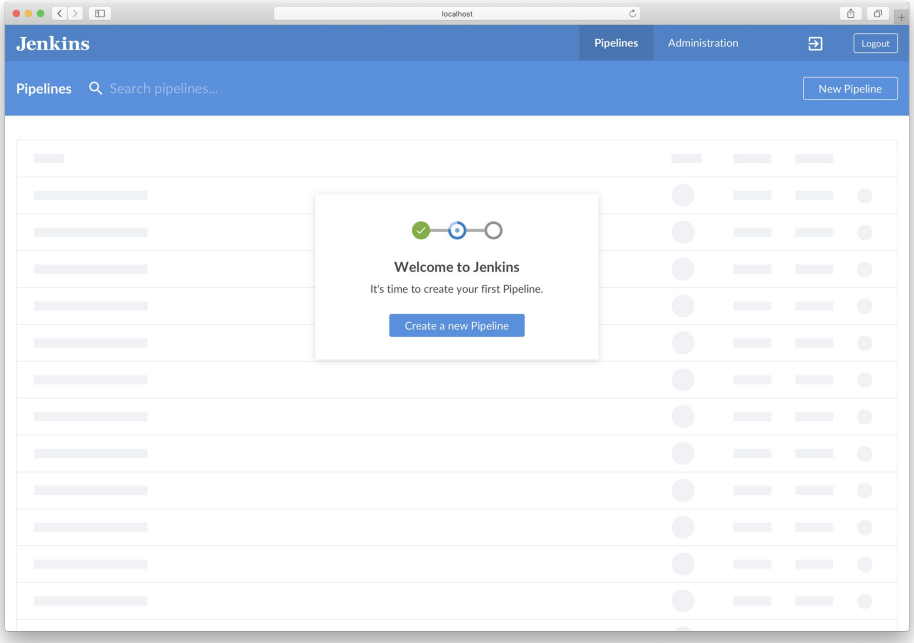


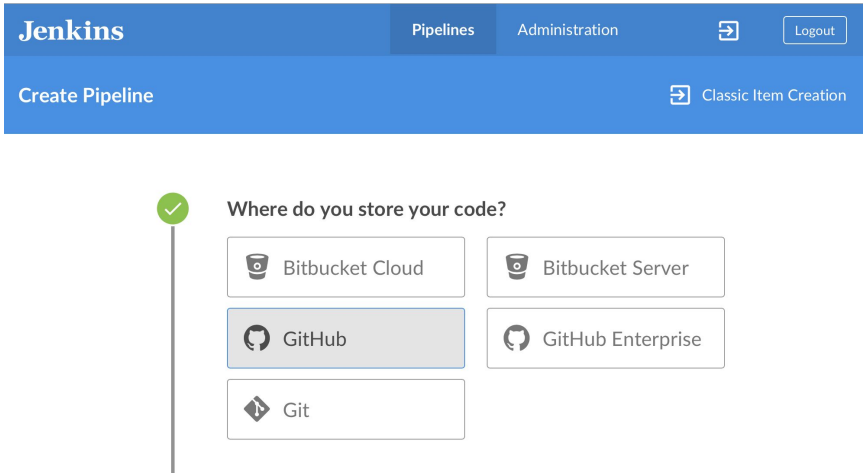
LOG IN TO JENKINS



http://localhost:10000/blue

SETUP THE PIPELINE





OUR END GOAL JENKINS PIPELINE


```
sh 'pip3 install
-r requirements.txt'
```

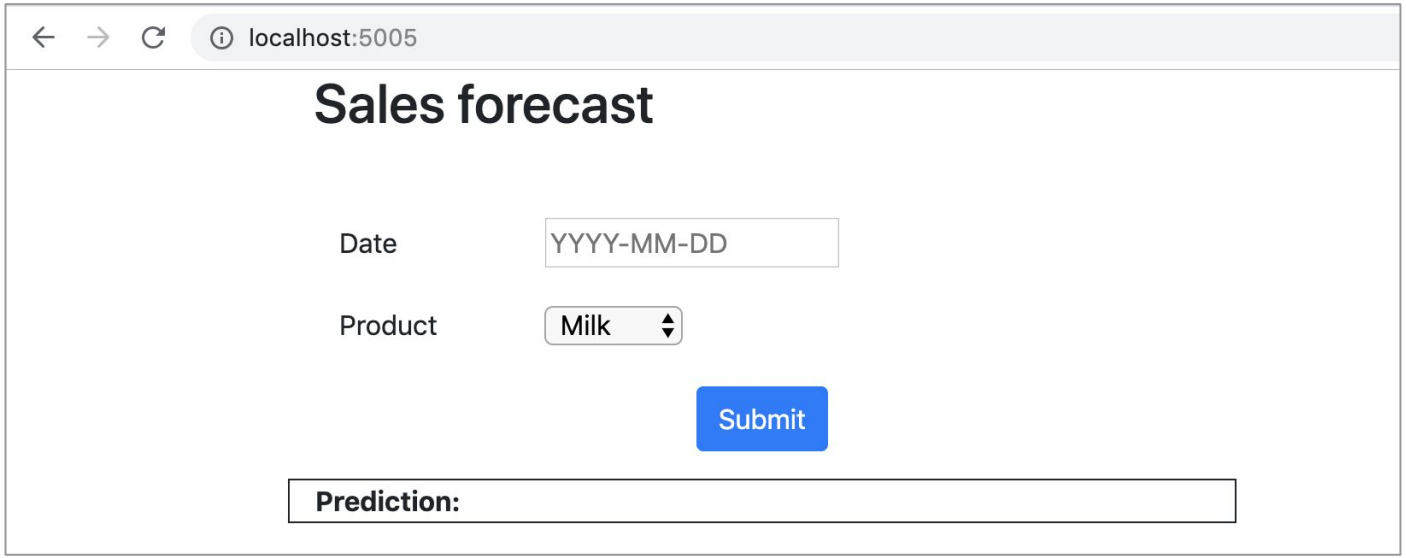
```
sh './run_tests.sh'
```

```
sh 'python3
run_python_script.py pipeline'
```

```
sh 'python3 run_python_script.py
acceptance'
```

```
sh 'curl --request POST --data-binary "@data/models/model.pkl" http://model:5005/replace_model'
sh 'curl --request POST --data-binary "@data/models/encoder.json"
http://model:5005/replace_encoder'
```

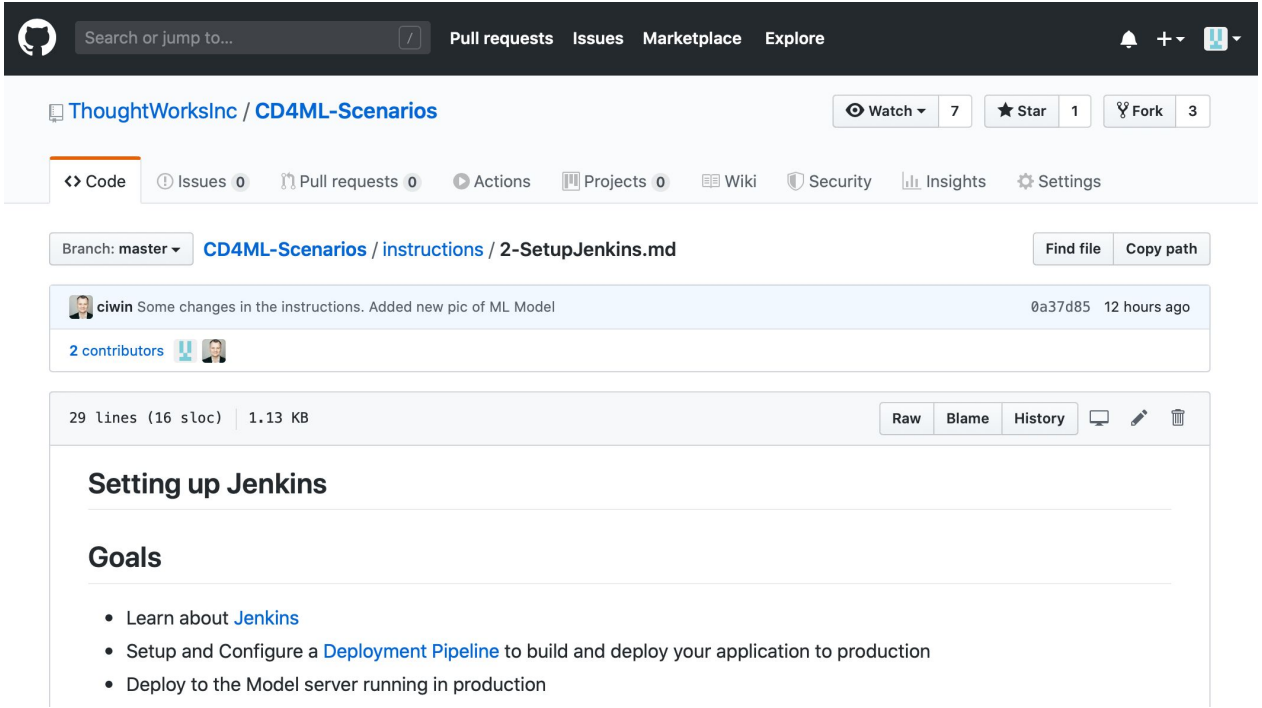
RUN THE WEB APPLICATION



http://localhost:11000/

Setup Jenkins

- Navigate to instructions/2-SetupJenkins.md



STEP BY STEP

1. **Doing the plumbing:** Set up the pipeline and see if it is working (CW)
2. **Data Science:** Develop the model and test it (with TDD) (DJ)
3. **Machine Learning Engineering:** Improve the model in several steps and monitor the results (DJ)
4. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the test (EN)
5. **Undo changes:** Roll back in time consistently with all artefacts (EN)
6. **Our app in the wild:** Monitor it in production with fluentd, elasticsearch and kibana (CW)

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

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
```

```
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.

Depth first design

Most common pattern is breadth-first design consisting of stages of batch processing
Run Stage 1 on all rows
Run Stage 2 on all rows
....
Construct final outputs

Debugging is hard. Wait 10 minutes for it to crash in stage 3 and try to add print statements. Slow.

Depth first design
Chain together transformation that can be run in streaming fashion. All stages first and then move to next row.
If it crashes on some line, just make that one line a test case and following TDD to fix it. Fast feedback

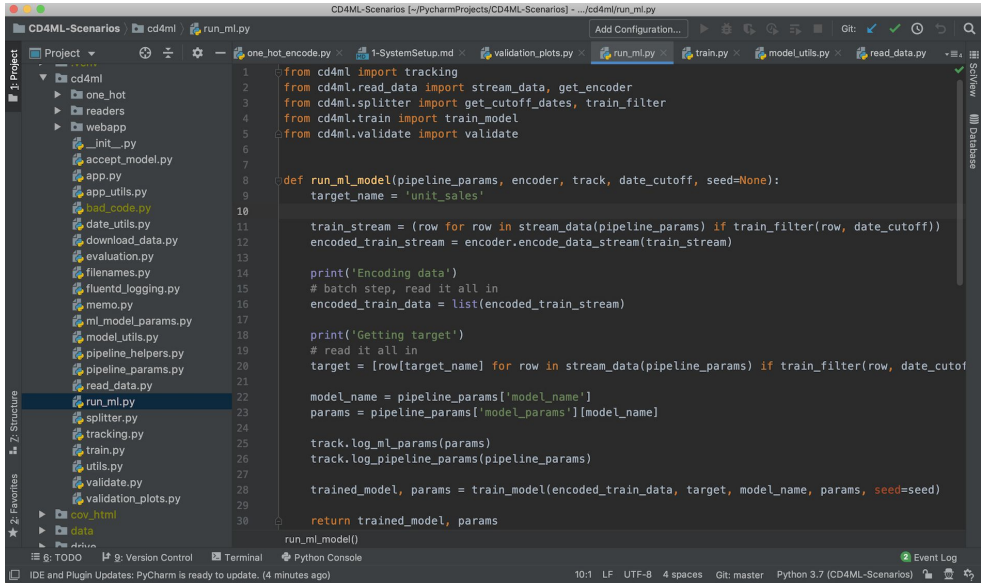
Don't mix

- Run all transformations as a single composite function to make new stream
- Apply filter function to create another stream
- Apply aggregation if needed to create another stream
- Collect the stream into a batch list only when needed. E.g. at Machine Learning Training
- More streams for example scoring and other processing

STEP BY STEP

1. **Doing the plumbing:** Set up the pipeline and see if it is working (CW)
2. **Data Science:** Develop the model and test it (with TDD) (DJ)
3. **Machine Learning Engineering:** Improve the model in several steps and monitor the results (DJ)
4. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the test (EN)
5. **Undo changes:** Roll back in time consistently with all artefacts (EN)
6. **Our app in the wild:** Monitor it in production with fluentd, elasticsearch and kibana (CW)

Overview of the Code Base and Scenarios



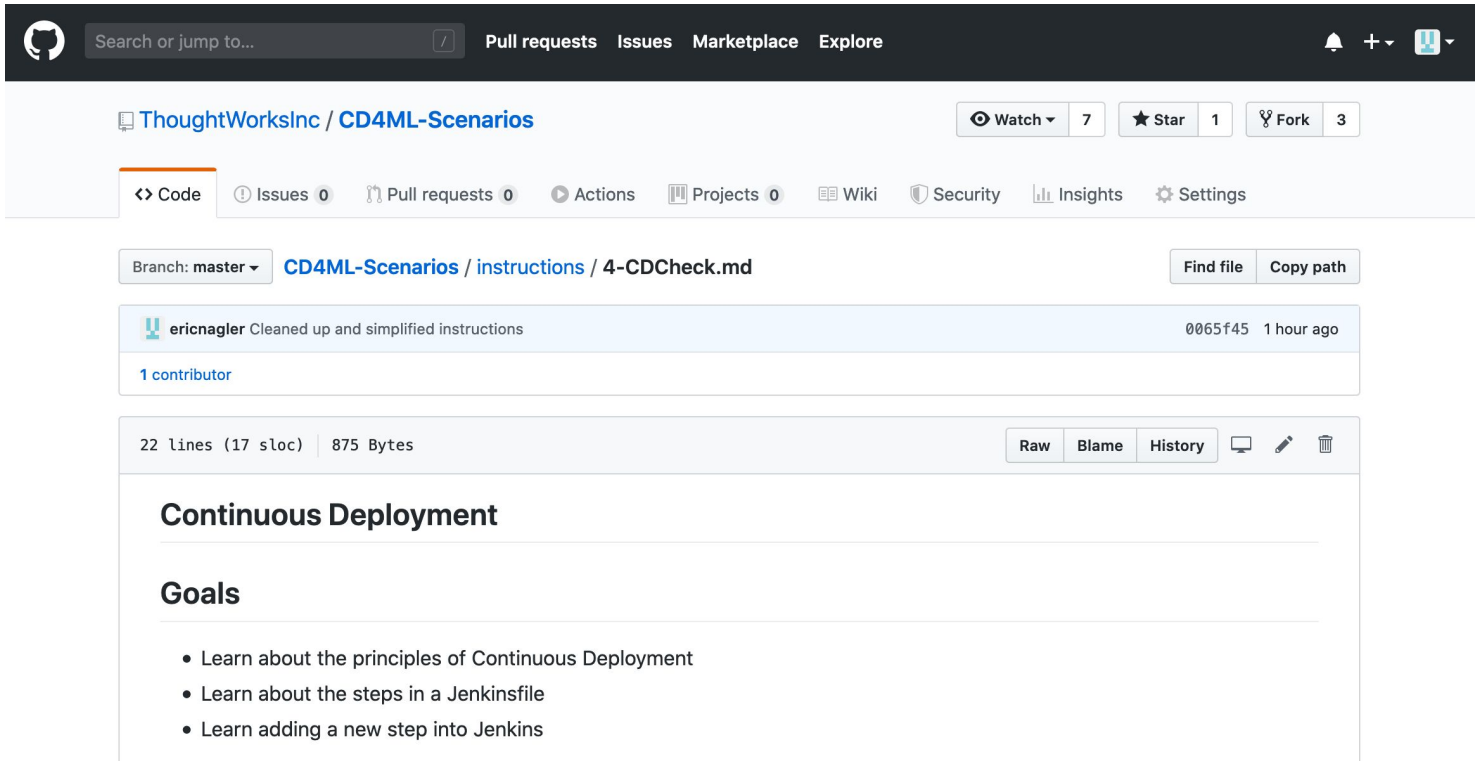
- Structure of code base
- How to run the pipeline After making changes
- How to view results of models and keep track of changes

STEP BY STEP

1. **Doing the plumbing:** Set up the pipeline and see if it is working (CW)
2. **Data Science:** Develop the model and test it (with TDD) (DJ)
3. **Machine Learning Engineering:** Improve the model in several steps and monitor the results (DJ)
4. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the test (EN)
5. **Undo changes:** Roll back in time consistently with all artefacts (EN)
6. **Our app in the wild:** Monitor it in production with fluentd, elasticsearch and kibana (CW)

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.”

- In the earlier scenarios, we have iterated on our model multiple times. Let’s add a step to our jenkins pipeline to add this continuous deployment check to ensure that our pipeline checks the model performance before deploying into our environments
- Uncomment the following block in your Jenkinsfile, commit and push your code

```
}
stage('Acceptance test') {
    steps {
        sh 'python3 run_python_script.py acceptance'
    }
}
```

Continuous Delivery

How are we checking for performance (accept_model.py)

```
def check_model_performance(metric_name, threshold_min, threshold_max):
    mlflow.set_tracking_uri(os.environ["MLFLOW_TRACKING_URL"])
    experiment = mlflow.get_experiment_by_name("local")
    runs = mlflow.search_runs(experiment_ids=experiment.experiment_id)
    last_run_record = get_latest_executed_run(runs)


    metric_value = get_metric(metric_name, last_run_record)
    run_name = last_run_record["tags.mlflow.runName"].head().values[0]
    template = "Metric: {metric_name} for Run: {run_name} was not accepted, " \
               "value: {metric_value}, " \
               "threshold_min: {threshold_min}, threshold_max: {threshold_max}"
    message = template.format(run_name=run_name,
                              metric_name=metric_name,
                              metric_value=metric_value,
                              threshold_min=threshold_min,
                              threshold_max=threshold_max)

    assert threshold_min <= metric_value <= threshold_max, message
```

Connect to MLFlow and retrieve the last run of the model

Retrieve the r2 score of the model

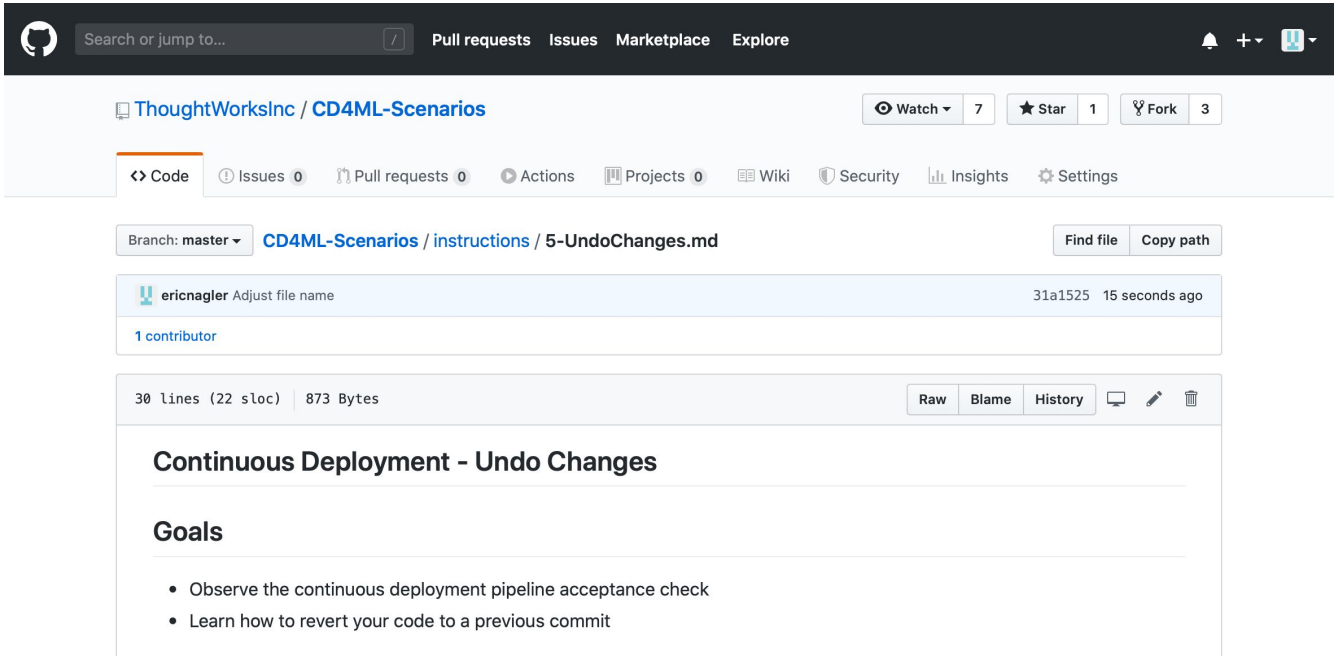
Ensure the model score is acceptable for deployment

STEP BY STEP

1. **Doing the plumbing:** Set up the pipeline and see if it is working (CW)
2. **Data Science:** Develop the model and test it (with TDD) (DJ)
3. **Machine Learning Engineering:** Improve the model in several steps and monitor the results (DJ)
4. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the test (EN)
5. **Undo changes:** Roll back code in time consistently with all artifacts (EN)
6. **Our app in the wild:** Monitor it in production with fluentd, elasticsearch and kibana (CW)

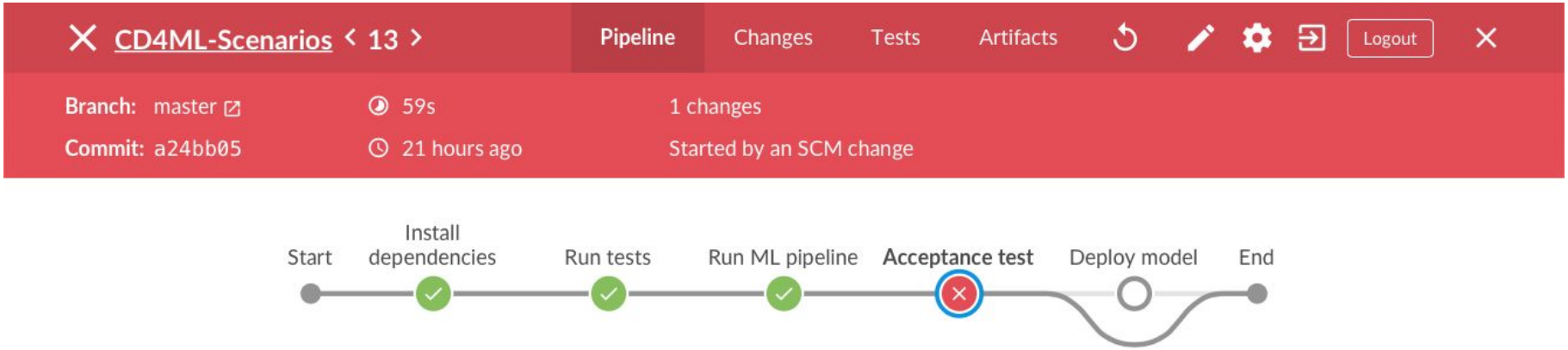
Undo Changes for Performance Regressions

- Navigate to [instructions/5-UndoChanges.md]



Undo Changes for Performance Regressions

- Verify that model adjustments that negatively impact model performance don't get released to production.
- We are going to purposely regress the model and watch our pipeline fail, then fix it!
- Change the model used to use less random forest trees. Change from 50 estimators to 5



Undo Changes for Performance Regressions

- But, because we performed our development in small incremental steps we can easily revert our changes back.
- `git revert HEAD`
- `git push`

STEP BY STEP

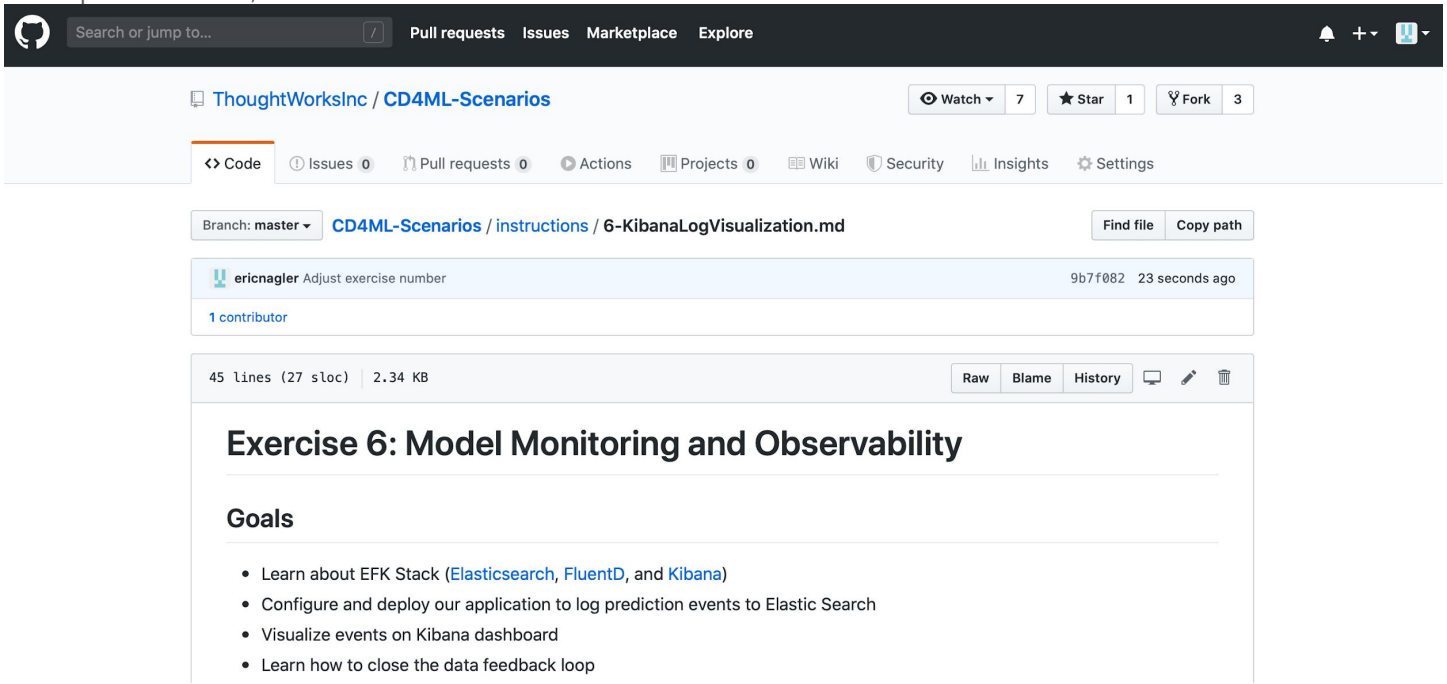
1. **Doing the plumbing:** Set up the pipeline and see if it is working (CW)
2. **Data Science:** Develop the model and test it (with TDD) (DJ)
3. **Machine Learning Engineering:** Improve the model in several steps and monitor the results (DJ)
4. **Continuous Deployment:** Setup a performance test of the model, which only allows automatic deployment if the model passes the test (EN)
5. **Undo changes:** Roll back in time consistently with all artefacts (EN)
6. **Our app in the wild:** Monitor it in production with fluentd, elasticsearch and kibana (CW)

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

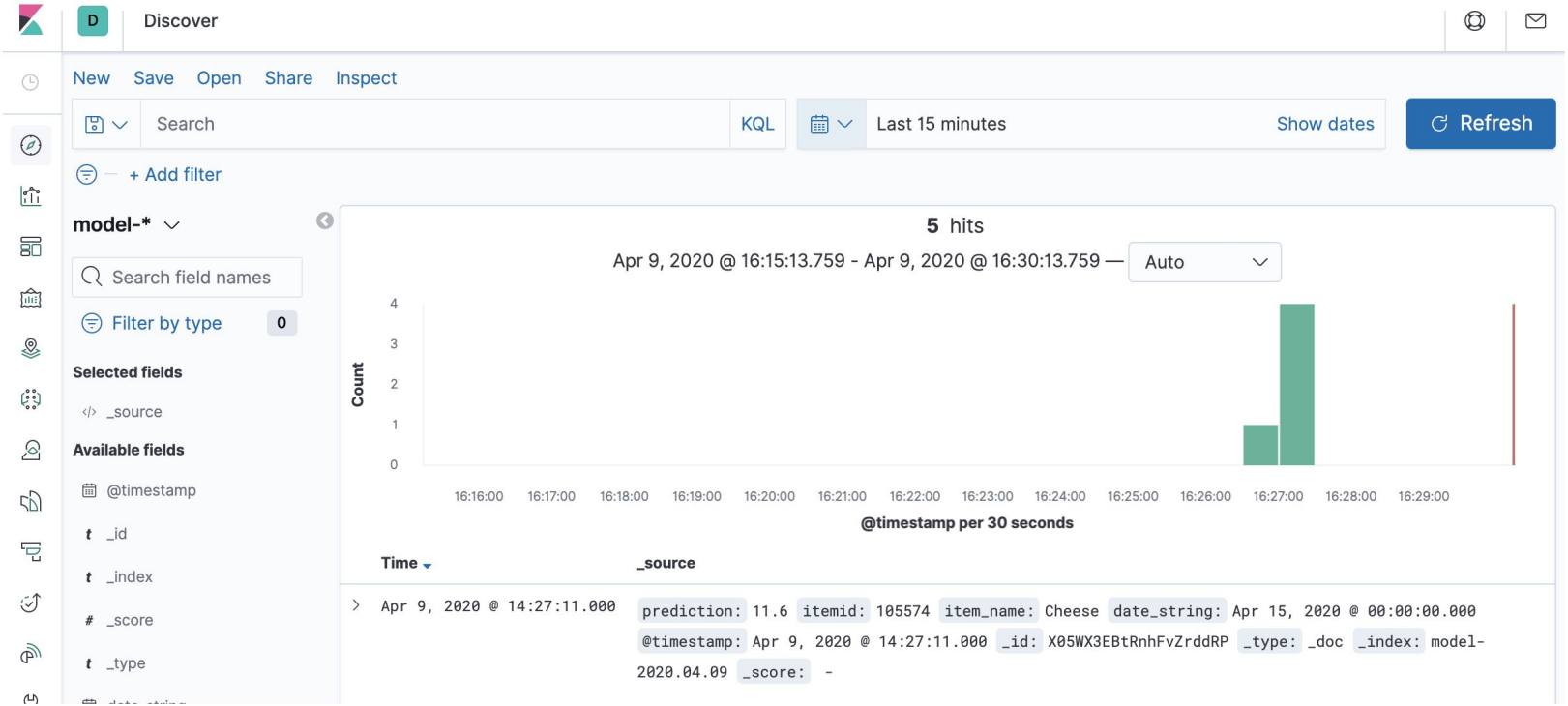
- **FluentD** is a open source data collector that allows for unified logging. FluentD allows for collection and forwarding of logs from any application into whatever service you like. In this case, we collect the logs from the model and forward them to...
- **ElasticSearch** is a distributed JSON, search and analytics engine allowing rapid search and log ingest. This allows us to index our models for monitoring and analytics. These records can be visualized in...
- **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/6-KibanaLogVisualization.md page on github to setup fluentd, elasticsearch and kibana



See prediction of your model in kibana



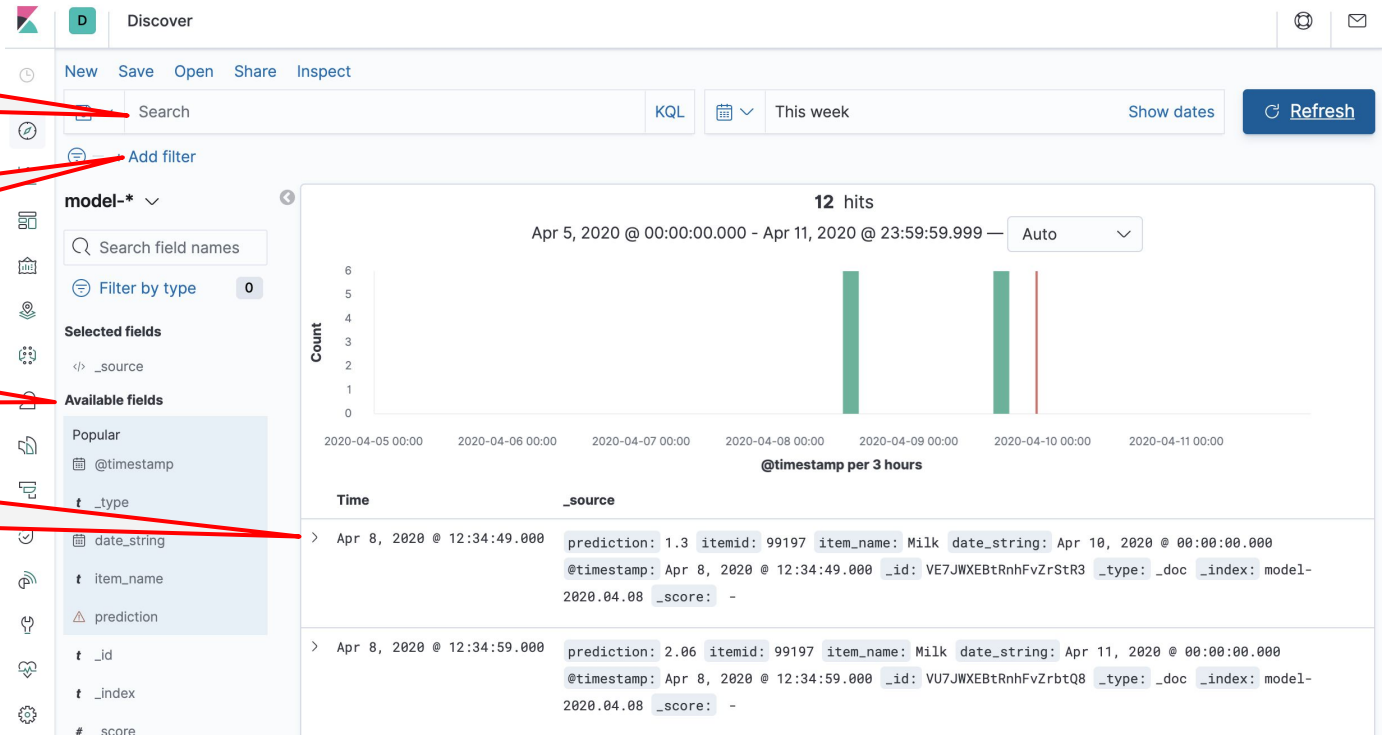
Exercise

Search predictions

Filter predictions

Select fields

Inspect table and json file



Wrap up

WHAT DID YOU LEARN?

- The Principles of CD4ML
- The components of a tech stack you need to run CD4ML
- Using Test Driven Development in a ML context
- Data Science best practice in the industry
- Run CD4ML on your laptop!

Links

- **Github**

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

- **Articles**

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

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