Kubeflow Kale

Scaling Jupyter Notebooks to Complex Pipelines

Valerio Maggio

Stefano Fioravanzo

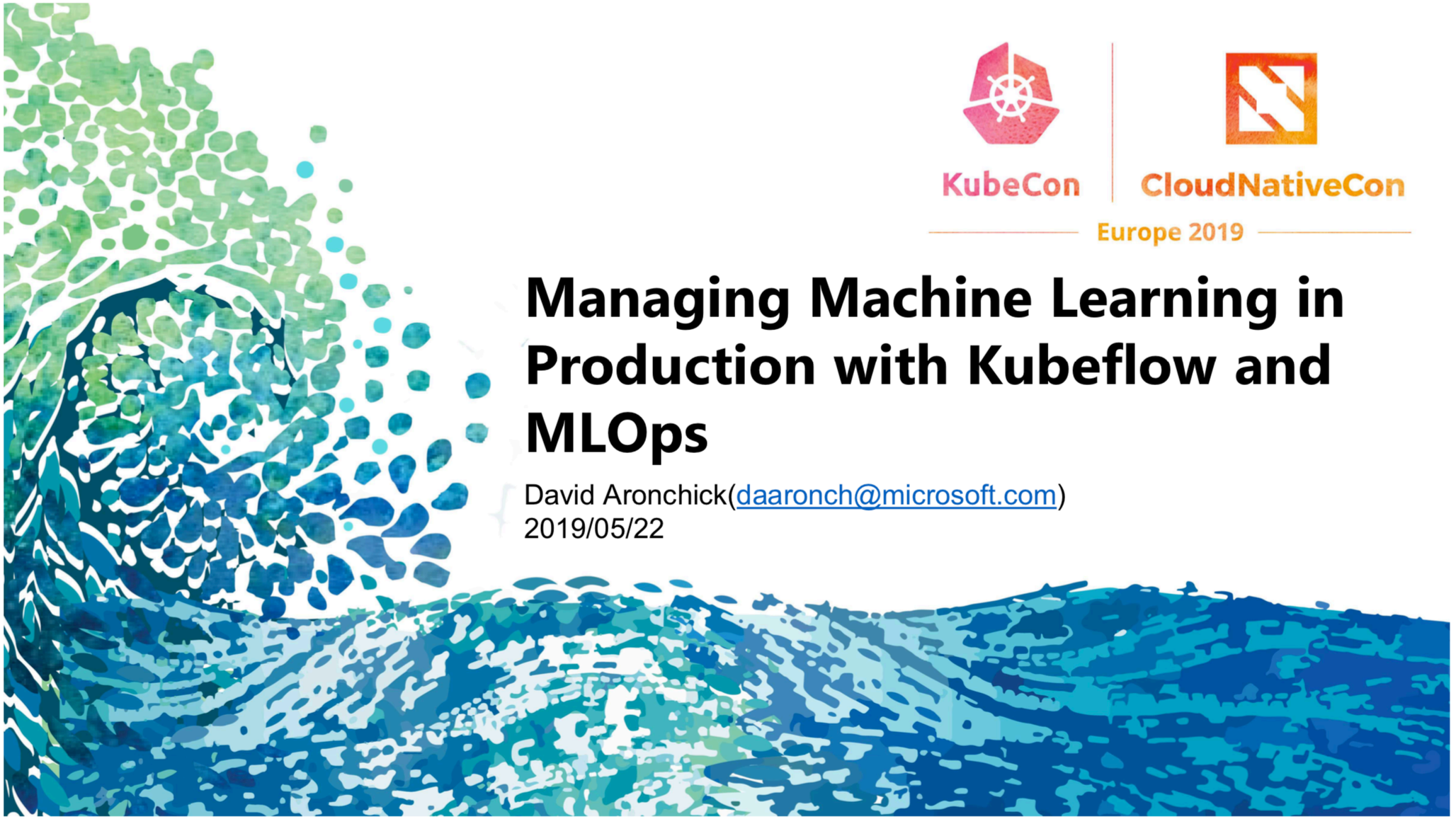
 @leriomaggio

 valeriomaggio@gmail.com

 @leriomaggio

MLOps!

ML + Dev + Ops

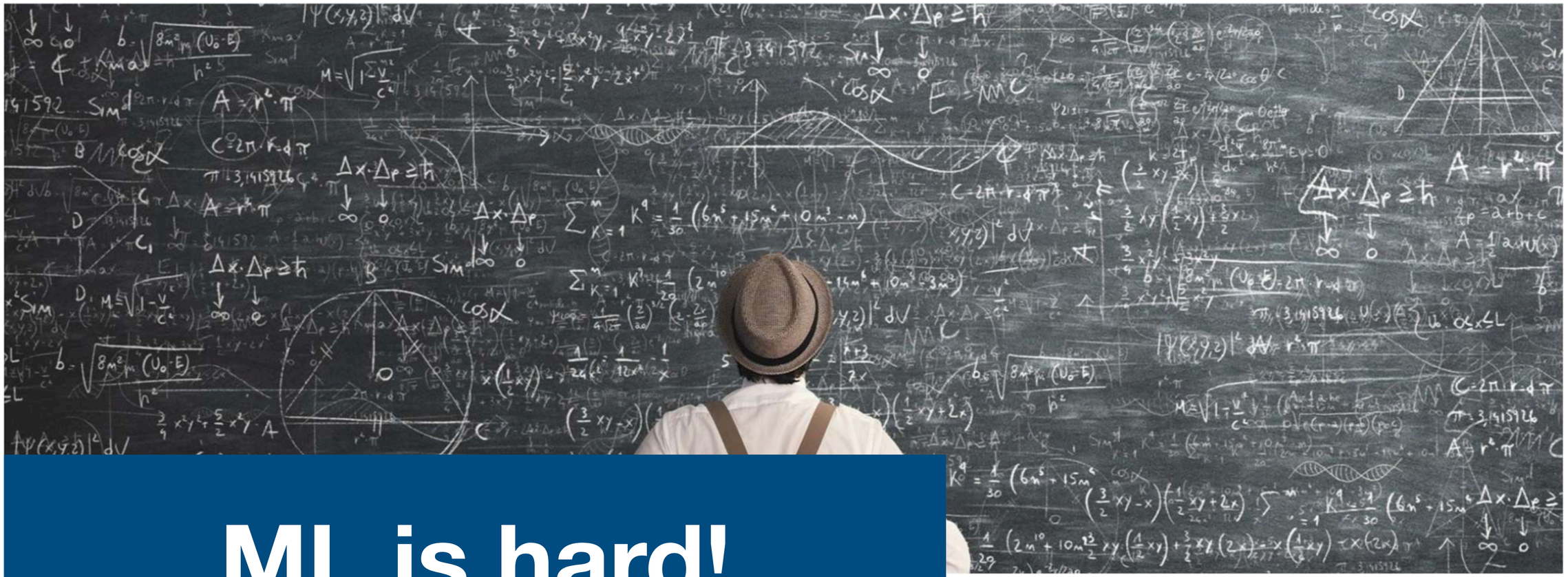


KubeCon | CloudNativeCon Europe 2019

Managing Machine Learning in Production with Kubeflow and MLOps

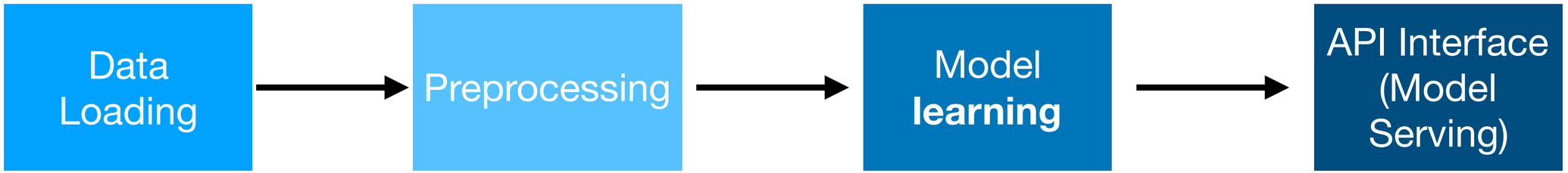
David Aronchick(daaronch@microsoft.com)
2019/05/22

Reference for the Introductory (MLOps) part

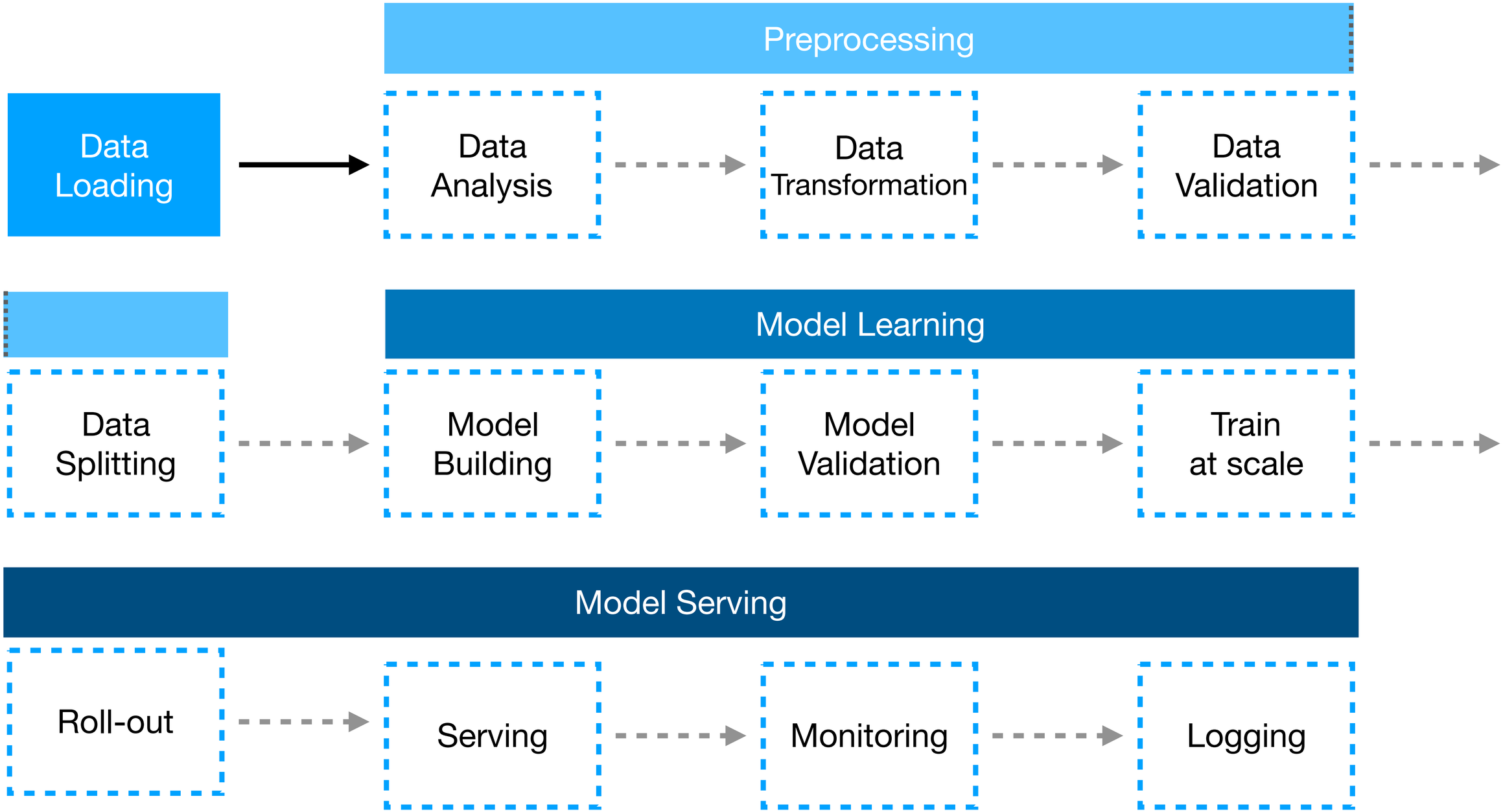


ML is hard!

“toy” ML Pipeline



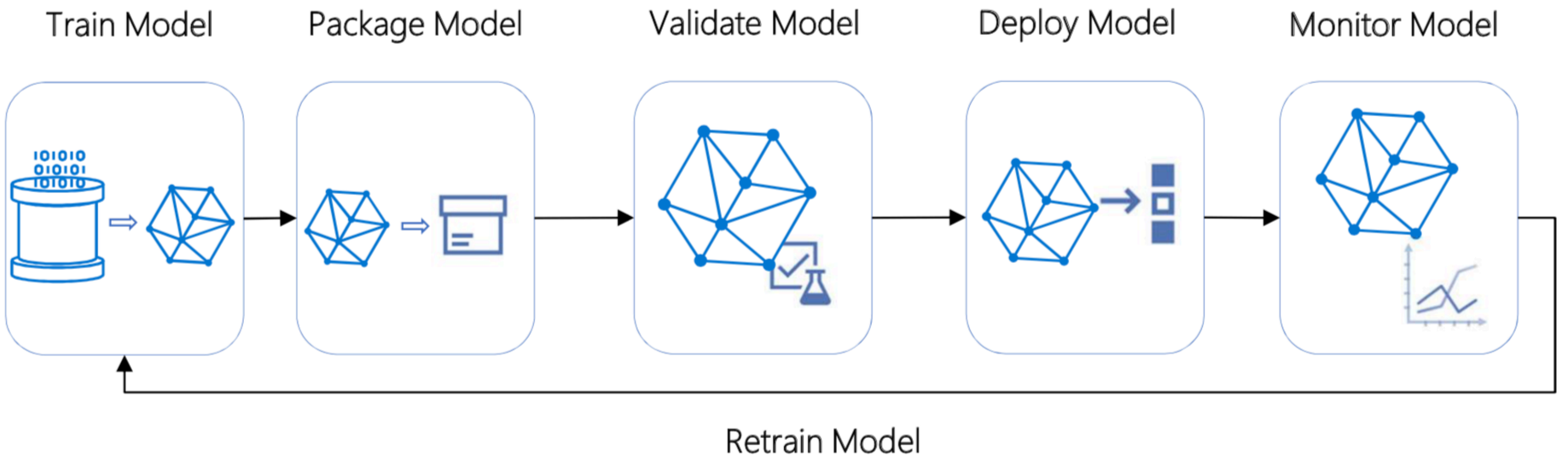
Adapted from “What about tests in Machine Learning projects?” Sarah Diot-Girard - EuroSciPy 2019



Adapted from "Managing Machine Learning in Production with Kubeflow and MLOps" David Aronchick - KubeCon EU 2019

What is the E2E ML lifecycle?

- **Develop & train model** with reusable ML pipelines
- **Package model** using containers to capture runtime dependencies for inference
- **Validate model behavior –** functionally, in terms of responsiveness, in terms of regulatory compliance
- **Deploy model -** to cloud & edge, for use in real-time / streaming / batch processing
- **Monitor model** behavior & business value, know **when to replace / deprecate a stale model**



Adapted from "Managing Machine Learning in Production with Kubeflow and MLOps" David Aronchick - KubeCon EU 2019

Ok, but, like, I'm
a data scientist. ~~IDGAF~~
I don't care
about all that.

Graham Lea @evolvable · May 2

Data science code doesn't need to follow the rules of good software engineering, because data science is not about creating software but about experimenting with building prototypes of models. 👈 Great tip from @jeremyphoward

15 4 26

Joel Grus
@joelgrus

Following

Replying to @evolvable @jeremyphoward

This is the most wrong tweet ever tweeted.

@jeremyphoward let's have a debate on this at some conference sometime

7:33 AM - 2 May 2018

ginablaber
@ginablaber

Follow

The story of enterprise Machine Learning: "It took me 3 weeks to develop the model. It's been >11 months, and it's still not deployed." @DineshNirmalIBM #StrataData #strataconf

10:19 AM - 7 Mar 2018

François Chollet
@fchollet

Following

Buggy code is bad science. Poorly tuned benchmarks are bad science. Poorly factored code is bad science (hinders reproducibility, increases chances of a mistake). If your field is all about empirical validation, then your code *is* a large part of your scientific output.

12:26 AM - 15 Jul 2018

47 Retweets **188** Likes

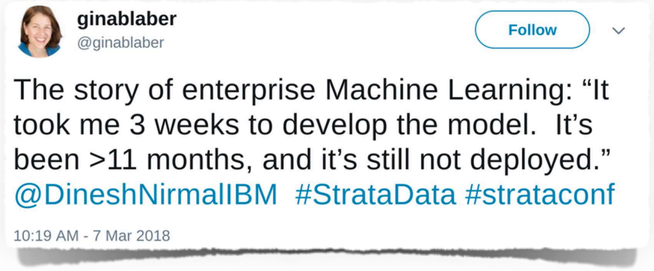
OMOJU
@omojumiller

Your neighborhood friendly reminder that 99% of #ML is actually infrastructure.

Thank you, you are welcome.

01:00 · 01/04/2019 · Twitter Web Client

53 Retweets **278** Likes



Cowboys and Rangers can be friends

Data Scientist

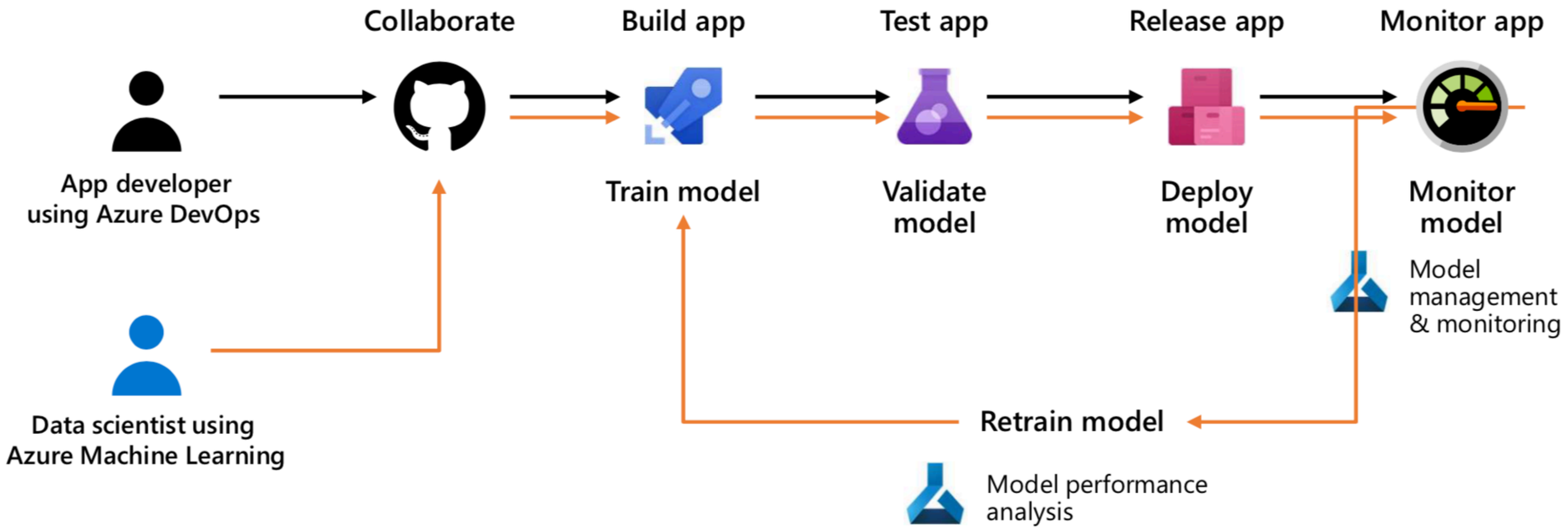
- Quick iteration
- Frameworks they understand
- Best of breed tools
- No management headaches
- Unlimited scale

SRE/ML Engineers

- Reuse of tooling and platforms
- Corporate compliance
- Observability
- Uptime

Adapted from “Managing Machine Learning in Production with Kubeflow and MLOps” David Aronchick - KubeCon EU 2019

MLOps Workflow

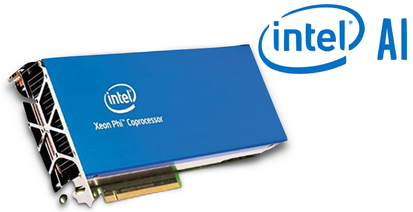


Model reproducibility | Model validation | Model deployment | Model retraining

Adapted from "Managing Machine Learning in Production with Kubeflow and MLOps" David Aronchick - KubeCon EU 2019

ML/DL Specialised Infrastructure

1. Specialised Chipset



2. Specialised Software

mxnet

PyTorch

TensorFlow

scikit learn

3. *(Specialised?)* Computing Platform

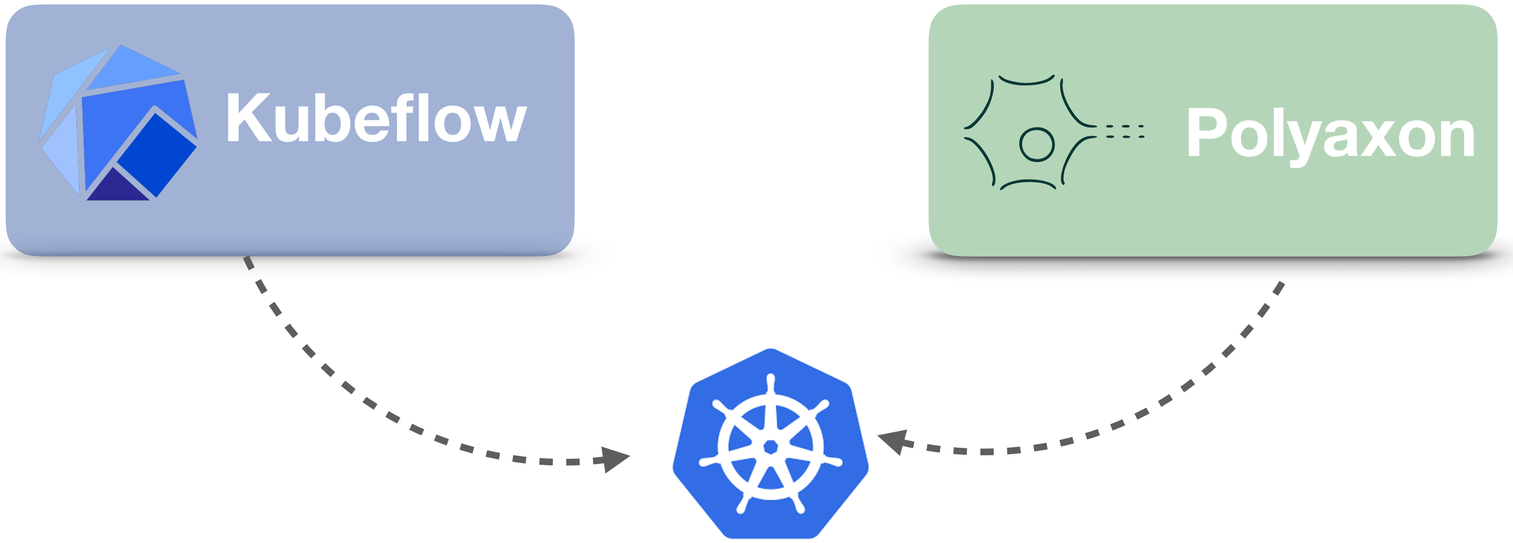
Azure

Google Cloud Platform

aws

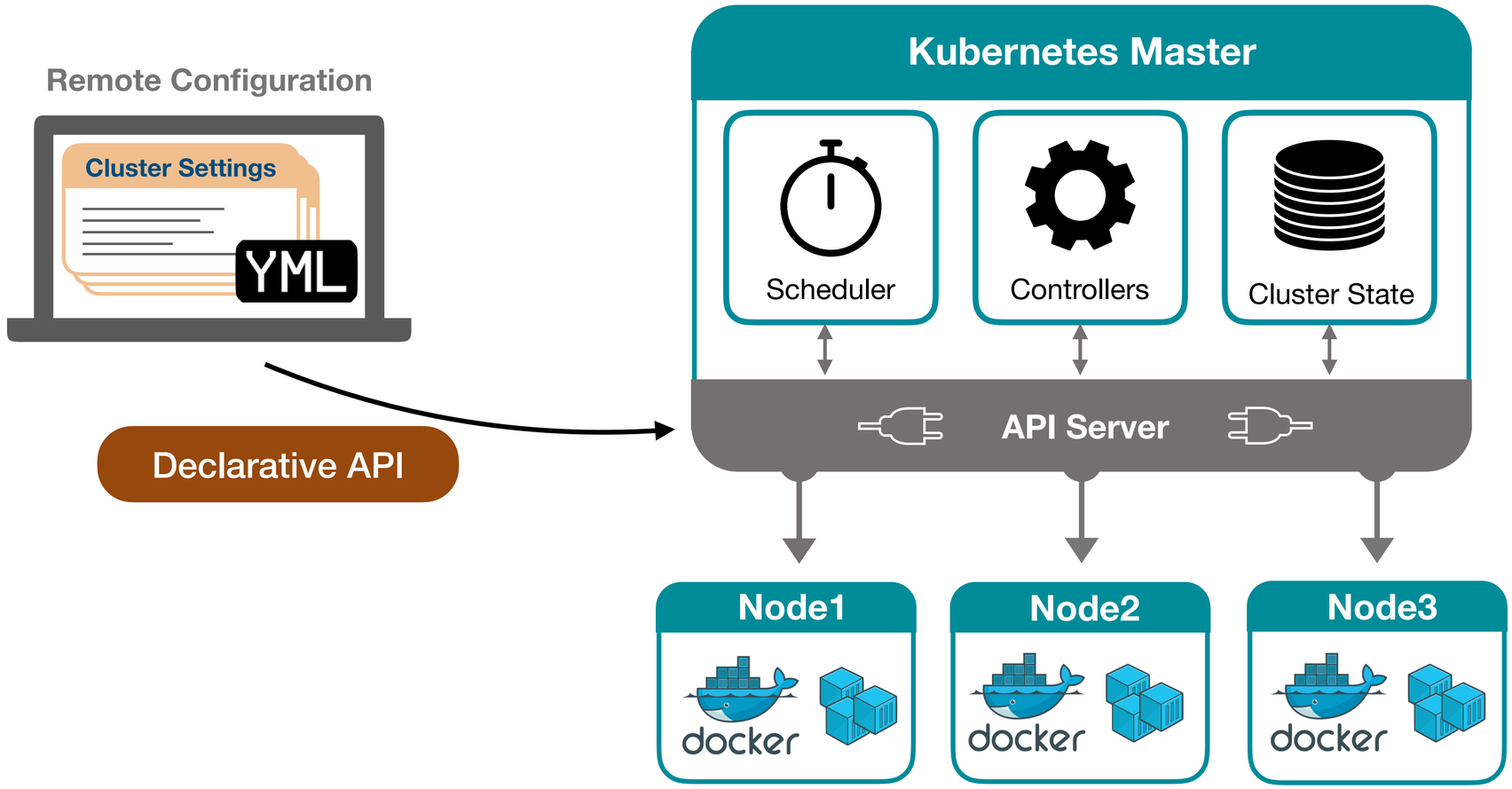
State of the Art Technologies

Support Machine Learning workloads in the Cloud



Cloud Agnostic

Kubernetes

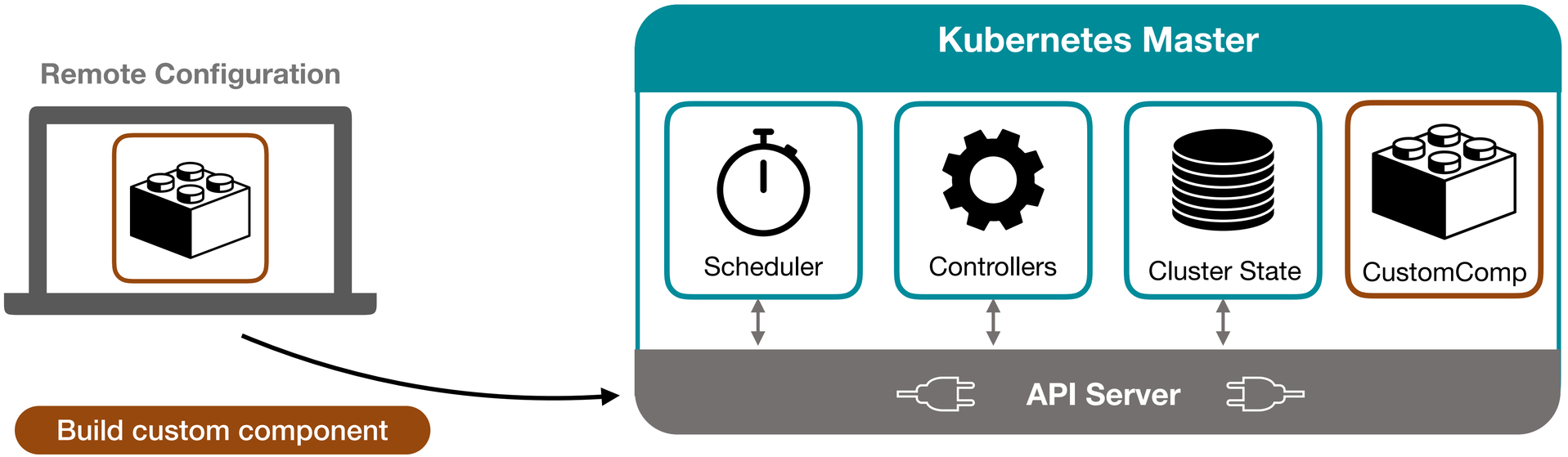


Kubernetes: Container Orchestration Engine

Developed internally at Google and open sourced in 2014

Extending Core Components

Kubernetes



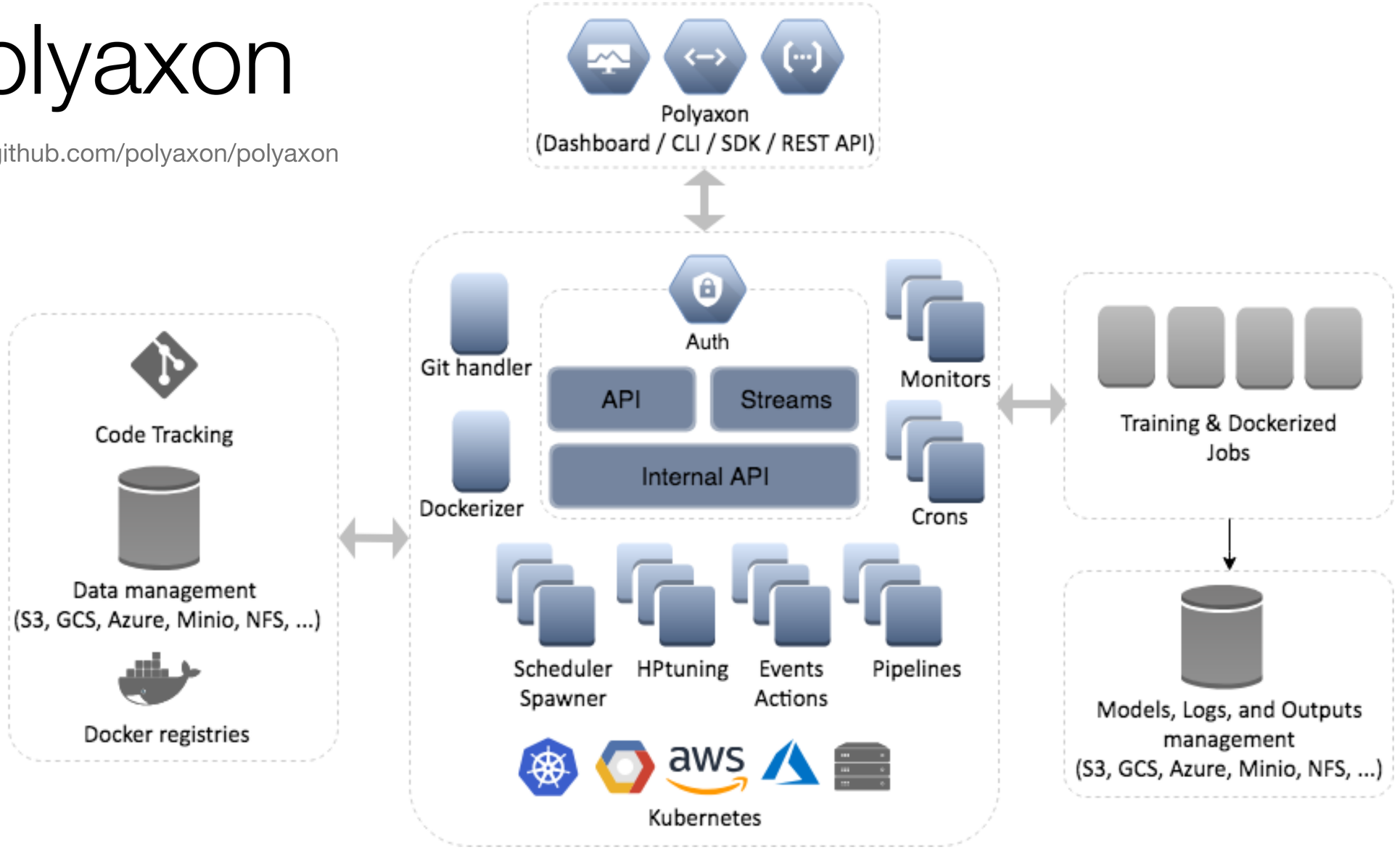
Supported Languages:

- GoLang
- Python
- Javascript
- Java
- Rust
- …

Automate Application specific services and tasks with custom components

Polyaxon

https://github.com/polyaxon/polyaxon

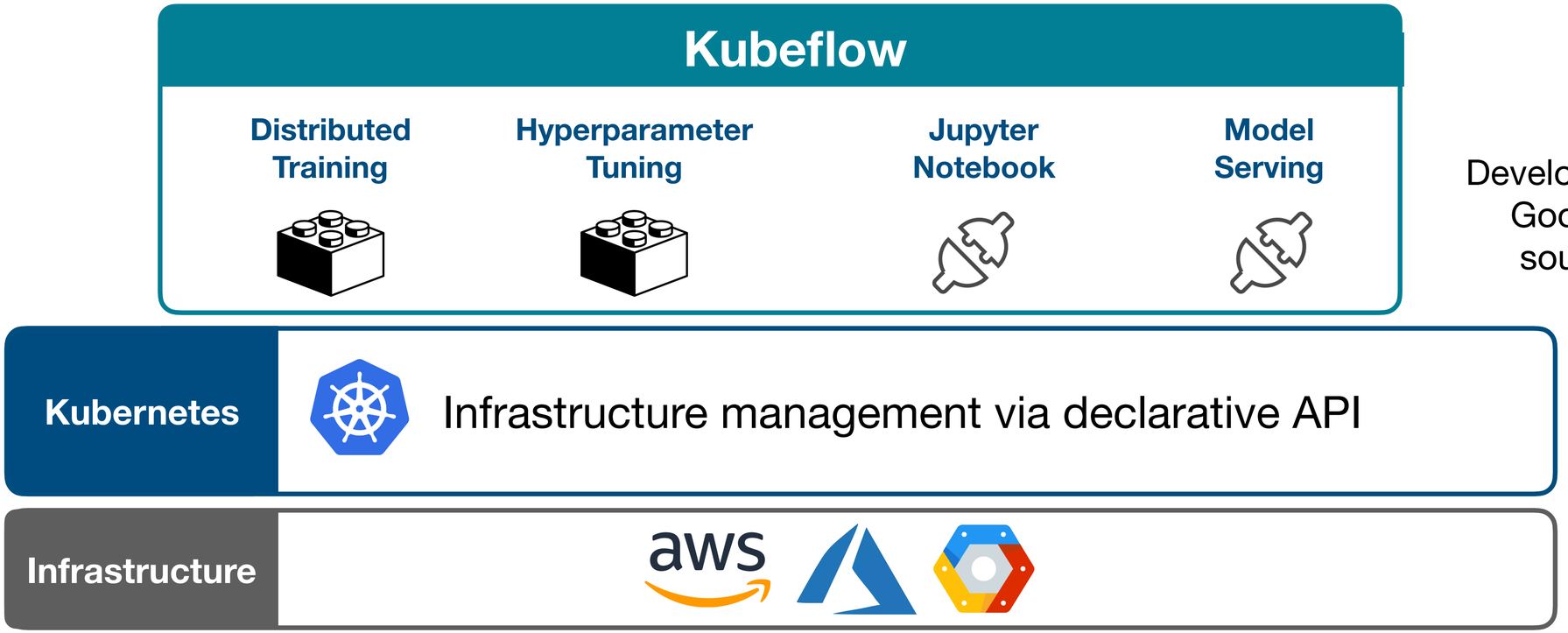


Platform for managing the whole lifecycle of large scale deep learning and machine learning applications.

Kubeflow

https://github.com/kubeflow/kubeflow

Kubernetes Components for Machine Learning

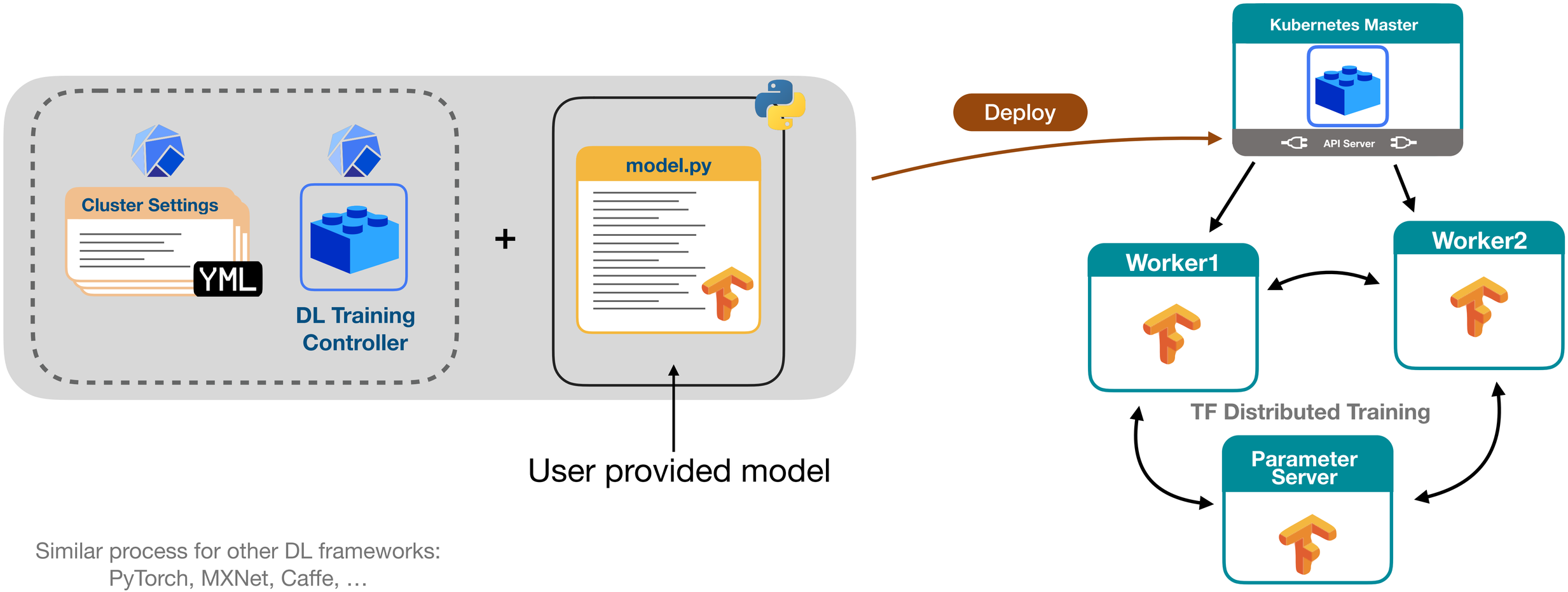


Kubeflow:
Developed internally at Google and open sourced in 2016

Distributed DL Training

Kubeflow

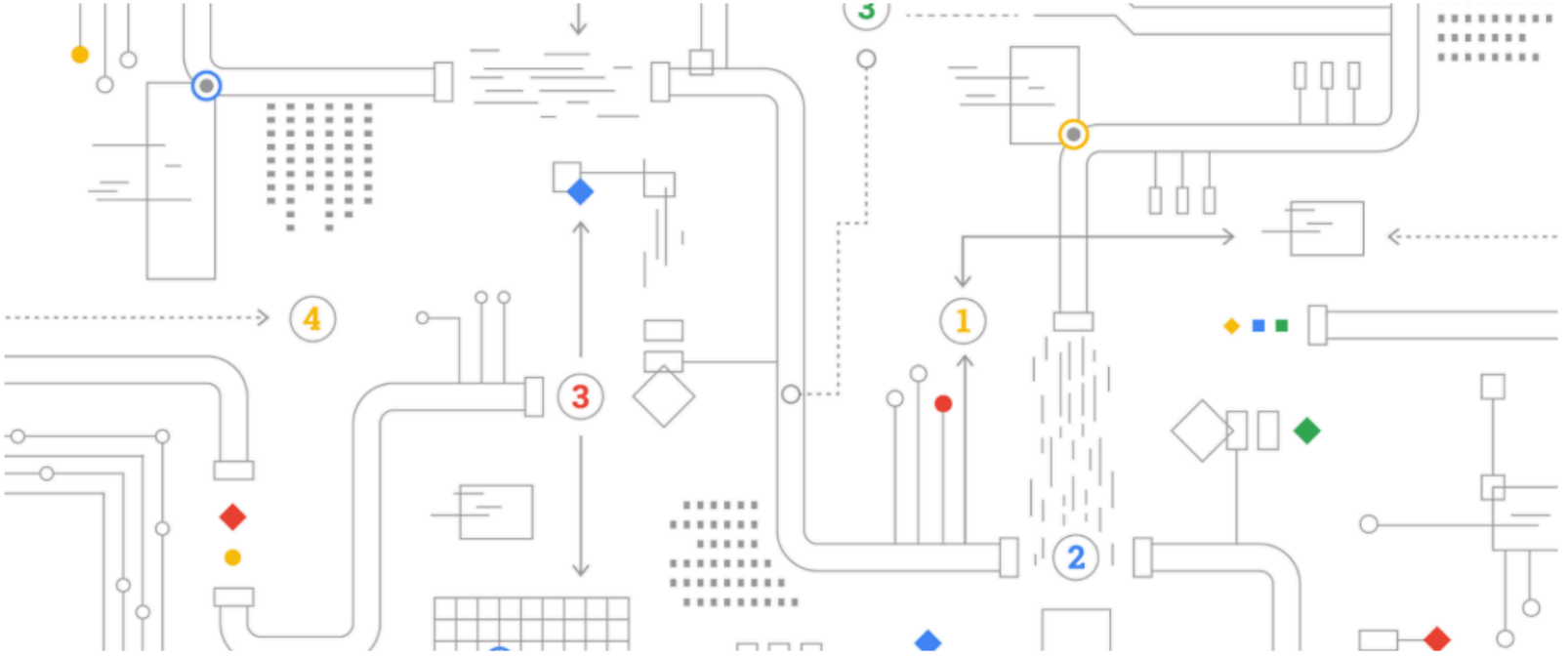
Example of distributed deep learning training on Kubeflow



Similar process for other DL frameworks:
PyTorch, MXNet, Caffe, …

AI & MACHINE LEARNING

Introducing AI Hub and Kubeflow Pipelines: Making AI simpler, faster, and more useful for businesses



Hussein Mehanna
Engineering Director, Cloud ML Platform

November 8, 2018

Whether they're revolutionizing the clothing manufacturing supply chain or accelerating e-commerce, businesses from every industry are increasingly turning to AI to advance what's possible. Yet for many businesses, the complexities of fully embracing AI can seem daunting.

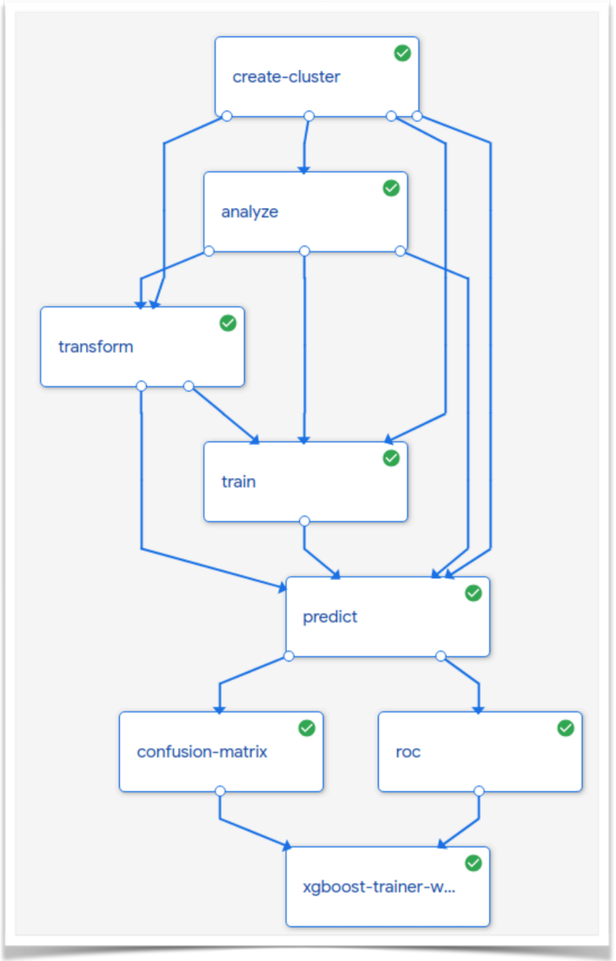
Introducing

Kubeflow
pipelines

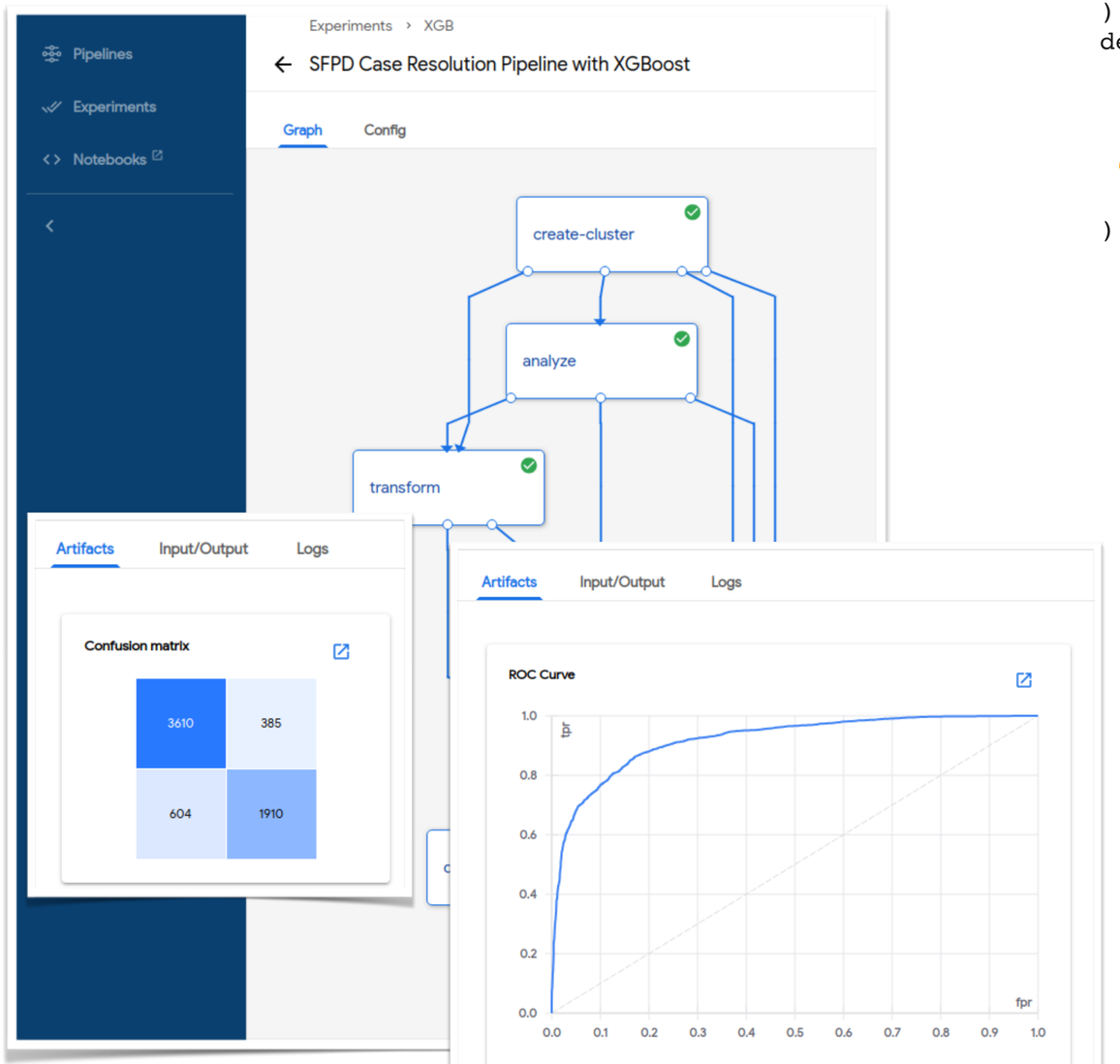
Kubeflow Pipelines: announced by **Google Cloud**.
A platform for building and deploying portable and scalable end-to-end ML workflows, based on *containers*.

The Kubeflow Pipelines platform consists of:

- **User interface** for managing and tracking experiments, jobs, and runs
- Engine for scheduling **multi-step ML workflows**
- **SDK** for defining and manipulating pipelines and components
- **Jupyter Notebooks** for interacting with the system using the SDK
- Integration with the other tools in the Kubeflow Toolkit
  (es: tf-operator for **distributed training**)



https://github.com/kubeflow/pipelines/



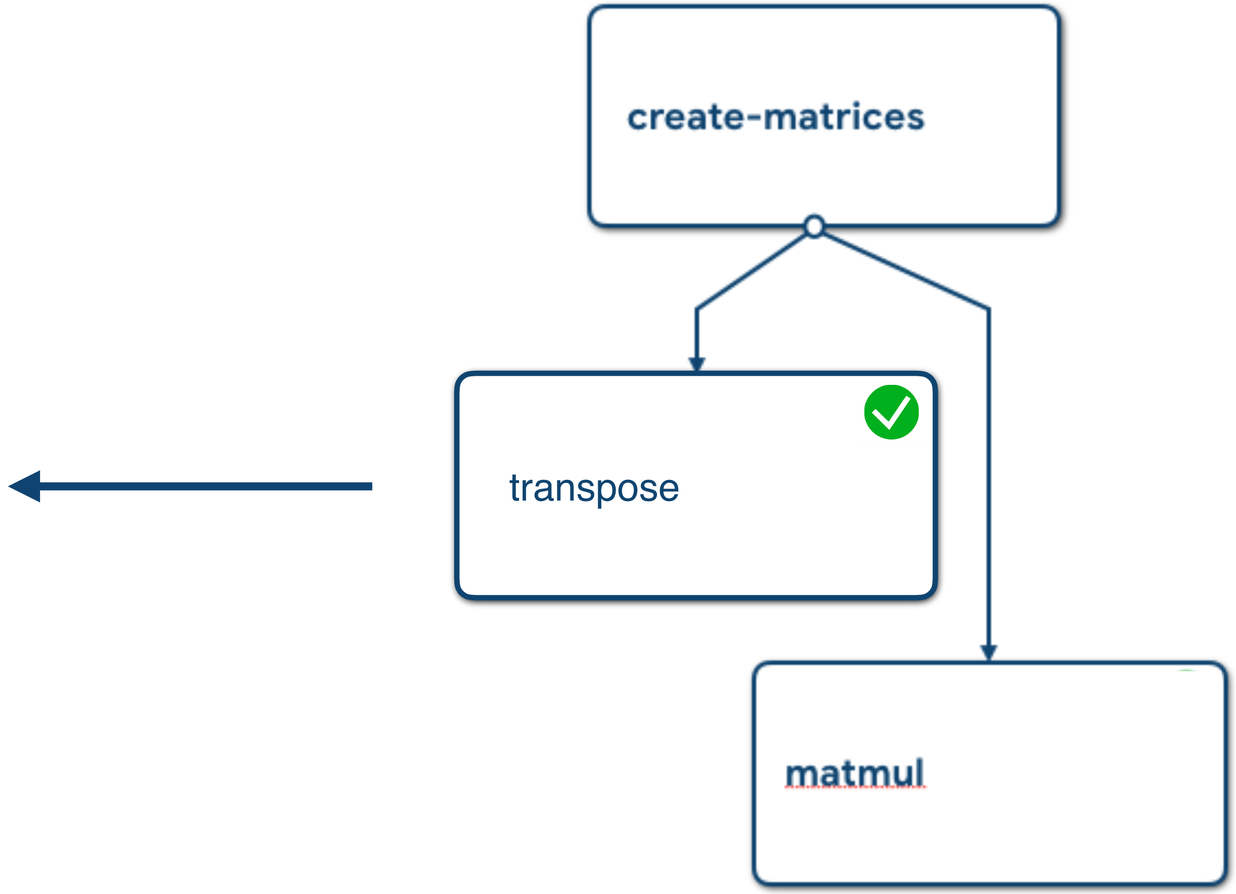

```
@dsl.pipeline(
  name='ML Workflow',
)
def xgb_train_pipeline(
    output,
    project,
    region='us-central1',
    train_data='gs://ml-pipeline-playground/sfpd/train.csv',
    …,
):
  with dsl.ExitHandler(exit_op=delete_cluster_op):
    create_cluster_op = CreateClusterOp('create-cluster', project, region, output)

    analyze_op = AnalyzeOp('analyze', project, region, create_cluster_op.output,
                           schema, train_data,
                           '%s/{{workflow.name}}/analysis' % output)

    transform_op = TransformOp('transform', project, region,
                               create_cluster_op.output, train_data, eval_data,
                               target, analyze_op.output,
                               '%s/{{workflow.name}}/transform' % output)

    train_op = TrainerOp('train', project, region, create_cluster_op.output,
                         transform_op.outputs['train'],transform_op.outputs['eval'],
                         target, analyze_op.output, workers,
                         rounds, '%s/{{workflow.name}}/model' % output)

    predict_op = PredictOp('predict', project, region, create_cluster_op.output,
                           transform_op.outputs['eval'], train_op.output, target,
                           analyze_op.output,
                           '%s/{{workflow.name}}/predict' % output)

    cm_op = ConfusionMatrixOp('confusion-matrix',
                              predict_op.output,
                              '%s/{{workflow.name}}/confusionmatrix' % output)

    roc_op = RocOp('roc', predict_op.output, true_label,
                   '%s/{{workflow.name}}/roc' % output)
```

Kubeflow Python SDK

Create Lightweight Components as Python Standalone functions

Standalone Python function

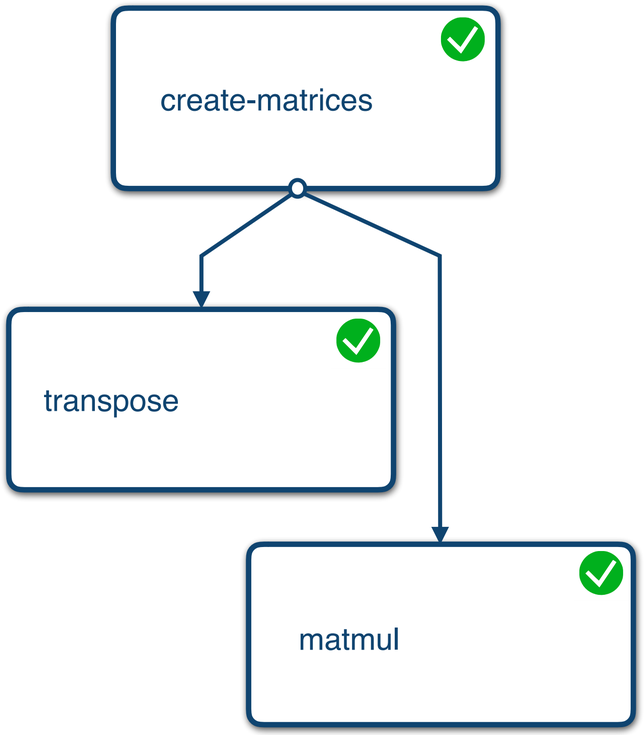
```python
def matrix_tranpose(A):
    '''
        Calculates the transpose
        of a matrix
    '''
    import numpy as np

    def compute_tranpose(m):
        return np.transpose(m)

    T = compute_transpose(A)
    return T
```



SDK

Kubeflow

Authoring Pipelines




```python
import kfp.components as comp

def create_matrices():
    import numpy as np
    a = np.random.random((10, 10))
    b = np.random.random((10, 10))
    return a, b

def transpose(a):
    import numpy as np
    return np.transpose(a)

def matmul(a, b):
    import numpy as np
    return np.matmul(a, b)

create_matrices_op = comp.func_to_container_op(create_matrices)
tranpose_op = comp.func_to_container_op(transpose)
matmul_op = comp.func_to_container_op(matmul)

import kfp.dsl as dsl
@dsl.pipeline(
   name='Matrix Calculation Pipeline',
   description='Test pipeline for matrix operations.'
)
def matrix_pipeline():
    create_task = creat_matrix_task(a, 4)
    transpose_task = divmod_op(create_task.outputs['a'])
    matmul_task = add_op(create_task.output)

pipeline_func = matrix_pipeline
pipeline_filename = pipeline_func.__name__ + '.pipeline.tar.gz'
import kfp.compiler as compiler
compiler.Compiler().compile(pipeline_func, pipeline_filename)

#Get or create an experiment and submit a pipeline run
import kfp
client = kfp.Client()
experiment = client.create_experiment("Matrix Test Experiment")

#Submit a pipeline run
run_name = pipeline_func.__name__ + ' run'
run_result = client.run_pipeline(experiment.id, run_name,
                                 pipeline_filename)
```

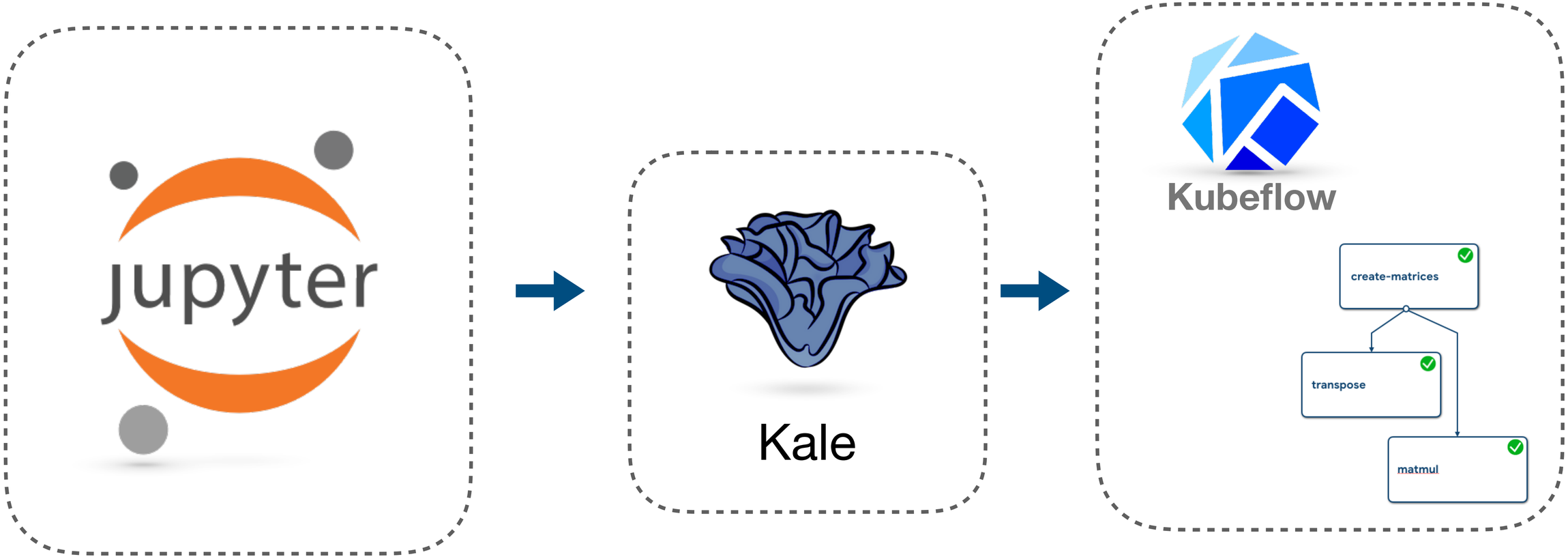
KALE (/ˈkeɪliː/)

Kubeflow Automated PipeLines Engine

https://kubeflow-kale.github.io

KALE

From (local) Jupyter Notebooks **to** (remote) Kubeflow Pipelines



Abstracting from KubeFlow Pipelines SDK

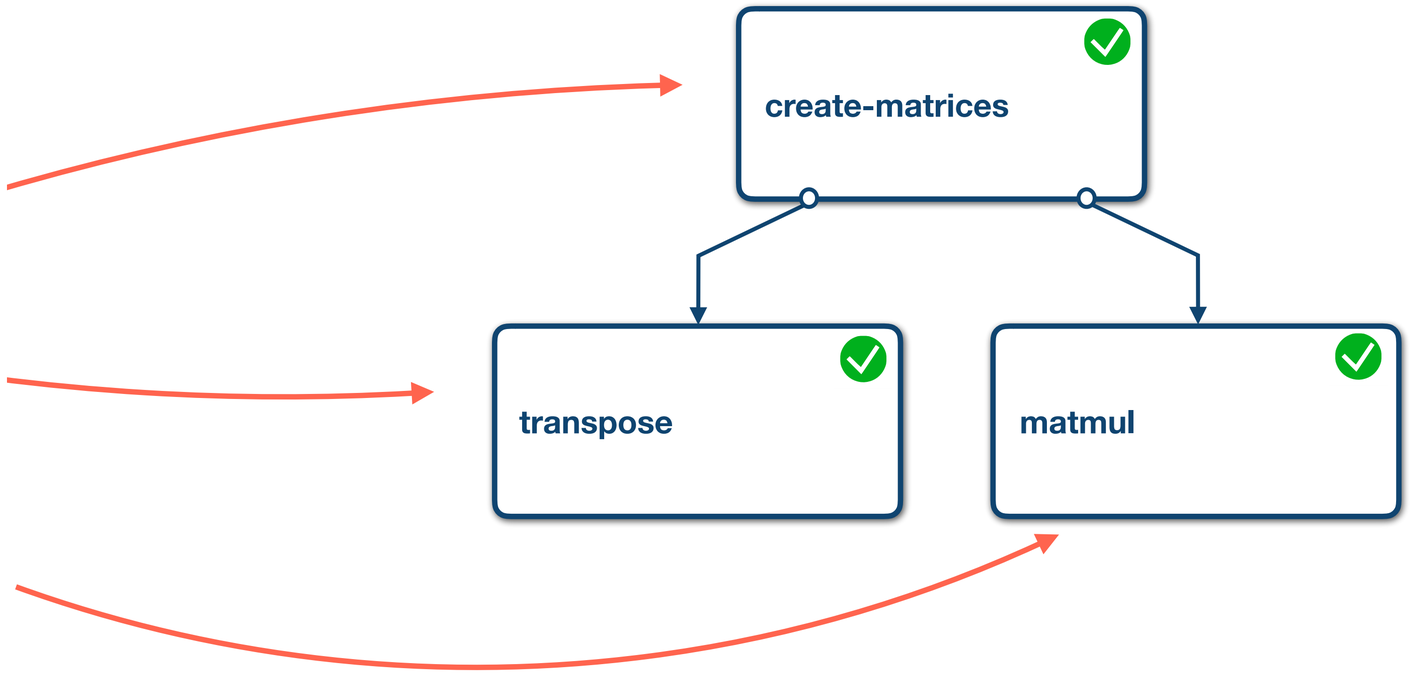
Jupyter Notebook to KFP

Kale

Local Jupyter Notebook

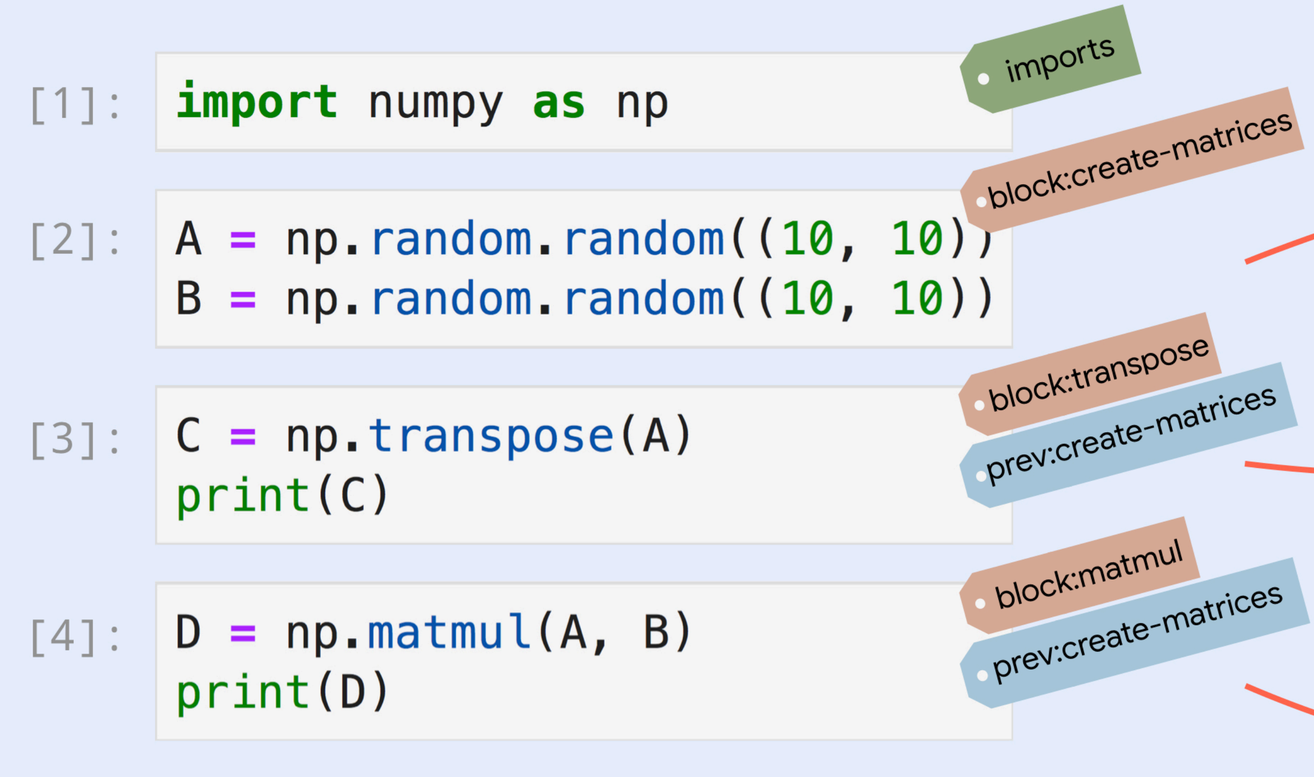
```
[1]: import numpy as np

[2]: A = np.random.random((10, 10))
     B = np.random.random((10, 10))

[3]: C = np.transpose(A)
     print(C)

[4]: D = np.matmul(A, B)
     print(D)
```

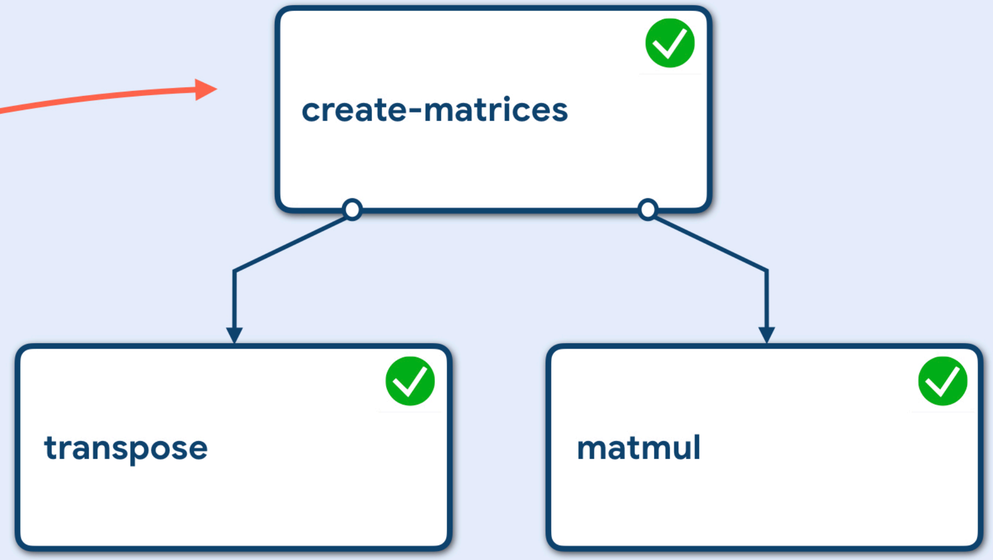
On cloud Kubeflow Pipeline



Local Jupyter Notebook

In Cloud Kubeflow Pipeline





block:<name>	Assign multiple code cells to the same KFP component	prev:<name>	Define a pipeline dependency	imports	Inject cell code at the beginning of every KFP component

Kale recognised list of tags

Tag	Description	Example
`^block:<block_name>(;<block_name>)*$`	Assign the current cell to a (multiple) pipeline step	`block:train-model` `block:processing-A;processing-B`
`^prev:<block_name>(;<block_name>)*$`	Define an execution dependency of the current cell to `n` other pipeline steps	`prev:load-dataset
`imports|functions`	Tell Kale to add this code block at the beginning of every pipeline code block. Useful to add imports/function to every pipeline step	-
`skip`	'Hide' the current cell from Kale.	-

Where `<block_name>` is matched against the regex: `[a-z0-9]`. So any string containing only digits and lowercase characters.

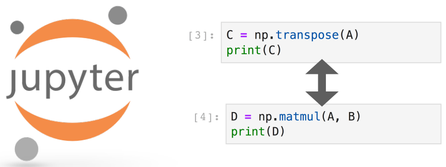
Jupyter-lab extension

https://github.com/kubeflow-kale/jupyterlab-kubeflow-kale

Kale main components

Kale

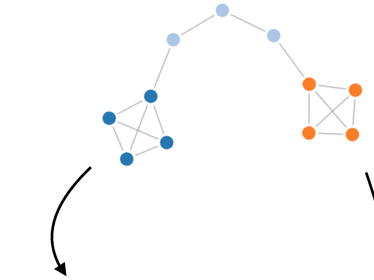
nbparser

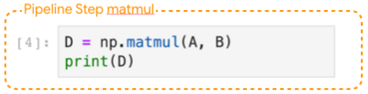


networkx

Derive pipeline structure

static_analyzer

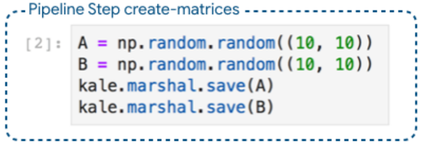


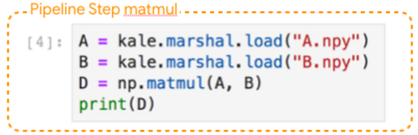


Pyflakes

Identify dependencies

marshal

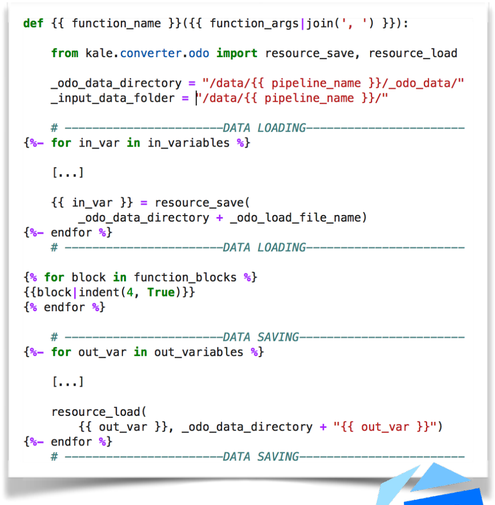




Odo
dill

Inject data objects

codegen



Jinja2

Generate & Deploy Pipeline

papermill support for **HyperParam** Tuning

static_analysis

Pipeline Step create-matrices

```
[2]: A = np.random.random((10, 10))
     B = np.random.random((10, 10))
```

Pipeline Step transpose

```
[3]: C = np.transpose(A)
     print(C)
```

Pipeline Step matmul

```
[4]: D = np.matmul(A, B)
     print(D)
```

marshal

Kale

Pipeline Step create-matrices

```
[2]: A = np.random.random((10, 10))
     B = np.random.random((10, 10))
     kale.marshal.save(A)
     kale.marshal.save(B)
```

Pipeline Step transpose

```
[3]: A = kale.marshal.load("A.npy")
     C = np.transpose(A)
     print(C)
```

Pipeline Step matmul

```
[4]: A = kale.marshal.load("A.npy")
     B = kale.marshal.load("B.npy")
     D = np.matmul(A, B)
     print(D)
```

Marshalling

```python
from kale.marshal import resource_save, resource_load

# register functions to load and save numpy objects
@resource_load.register('.*\.npy')  # match anything ending in '.npy'
def resource_numpy_load(uri, **kwargs):
    return np.load(uri)

@resource_save.register('numpy\..*')  # match anything starting with 'numpy'
def resource_numpy_save(obj, path, **kwargs):
    return np.save(path+".npy", obj)

a = np.random.random((10,10))
resource_save(a, 'marshal_dir/')
b = resource_load('marshal_dir/a.npy')  # a == b
```

PatternDispatcher

TypeDispatcher

Kale

codegen

Code generation via templates

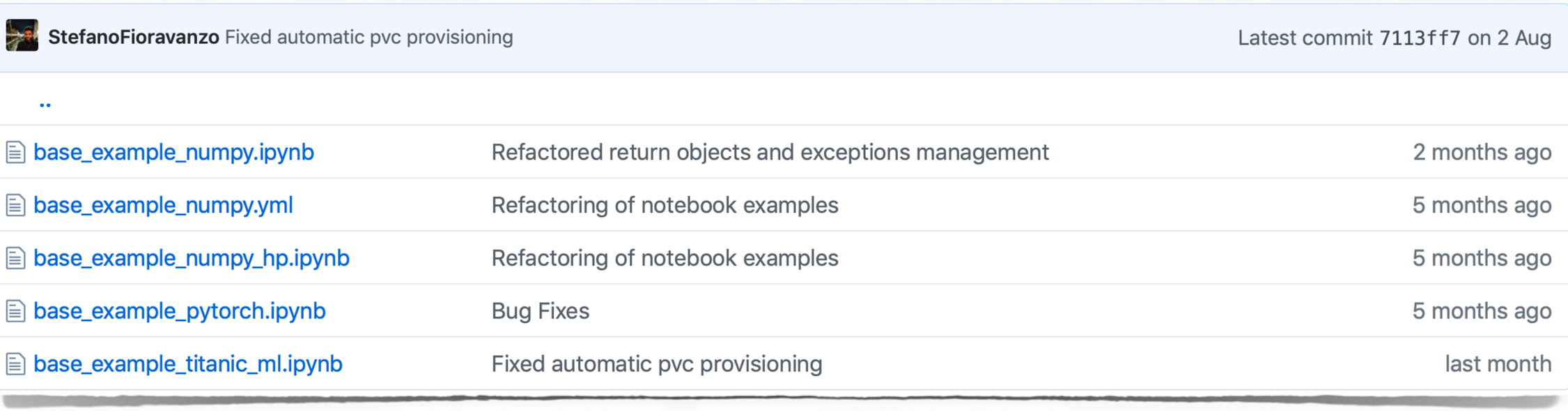

```
def {{ function_name }}({{ function_args|join(', ') }}):

    from kale.converter.odo import resource_save, resource_load

    _odo_data_directory = "/data/{{ pipeline_name }}/_odo_data/"
    _input_data_folder = "/data/{{ pipeline_name }}/"

    # ----------------------DATA LOADING----------------------
{%- for in_var in in_variables %}

    [...]

    {{ in_var }} = resource_save(
        _odo_data_directory + _odo_load_file_name)
{%- endfor %}
    # ----------------------DATA LOADING----------------------

{% for block in function_blocks %}
{{block|indent(4, True)}}
{% endfor %}

    # ----------------------DATA SAVING----------------------
{%- for out_var in out_variables %}

    [...]

    resource_load(
        {{ out_var }}, _odo_data_directory + "{{ out_var }}")
{%- endfor %}
    # ----------------------DATA SAVING----------------------
```

Sample template used to generate a standalone function

Examples

https://github.com/kubeflow-kale/kale/tree/master/examples

https://github.com/kubeflow-kale/examples



StefanoFioravanzo Fixed automatic pvc provisioning — Latest commit `7113ff7` on 2 Aug

..		
base_example_numpy.ipynb	Refactored return objects and exceptions management	2 months ago
base_example_numpy.yml	Refactoring of notebook examples	5 months ago
base_example_numpy_hp.ipynb	Refactoring of notebook examples	5 months ago
base_example_pytorch.ipynb	Bug Fixes	5 months ago
base_example_titanic_ml.ipynb	Fixed automatic pvc provisioning	last month

Thank you very much for your kind attention

https://kubeflow-kale.github.io

@leriomaggio valeriomaggio@gmail.com @leriomaggio