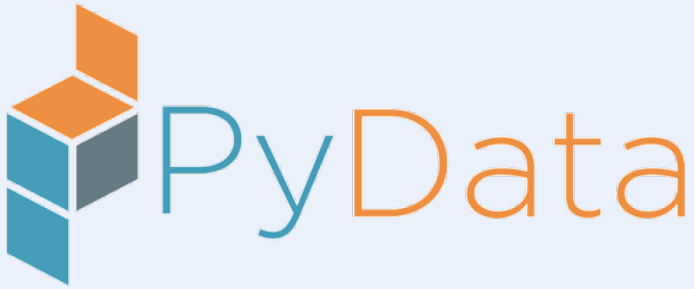




# RAPIDS.AI

Leveraging GPUs for accelerated data science & data analytics

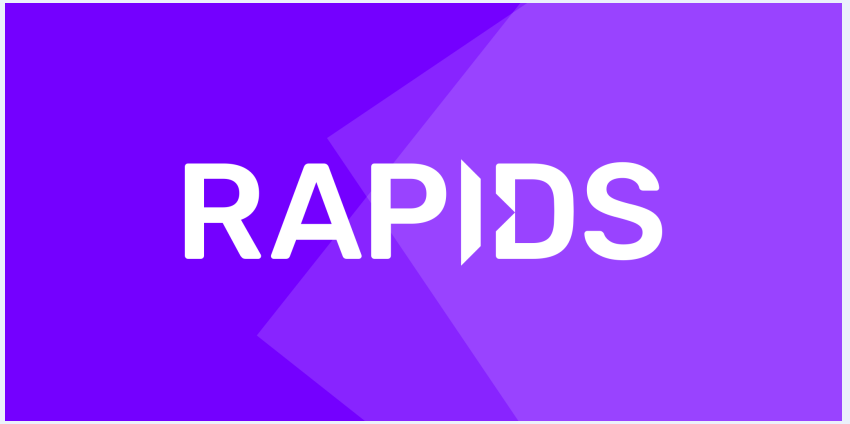
Arnaud Wald – Machine Learning Engineer – Scaleway

# The Data Science life

- Lots of **iterations**
  - New features
  - Transformations
  - Training Machine Learning
  - Bug/corrections
- Faster iterations → More efficiency / Less frustration
- All PyData tools help for this
  - On CPU

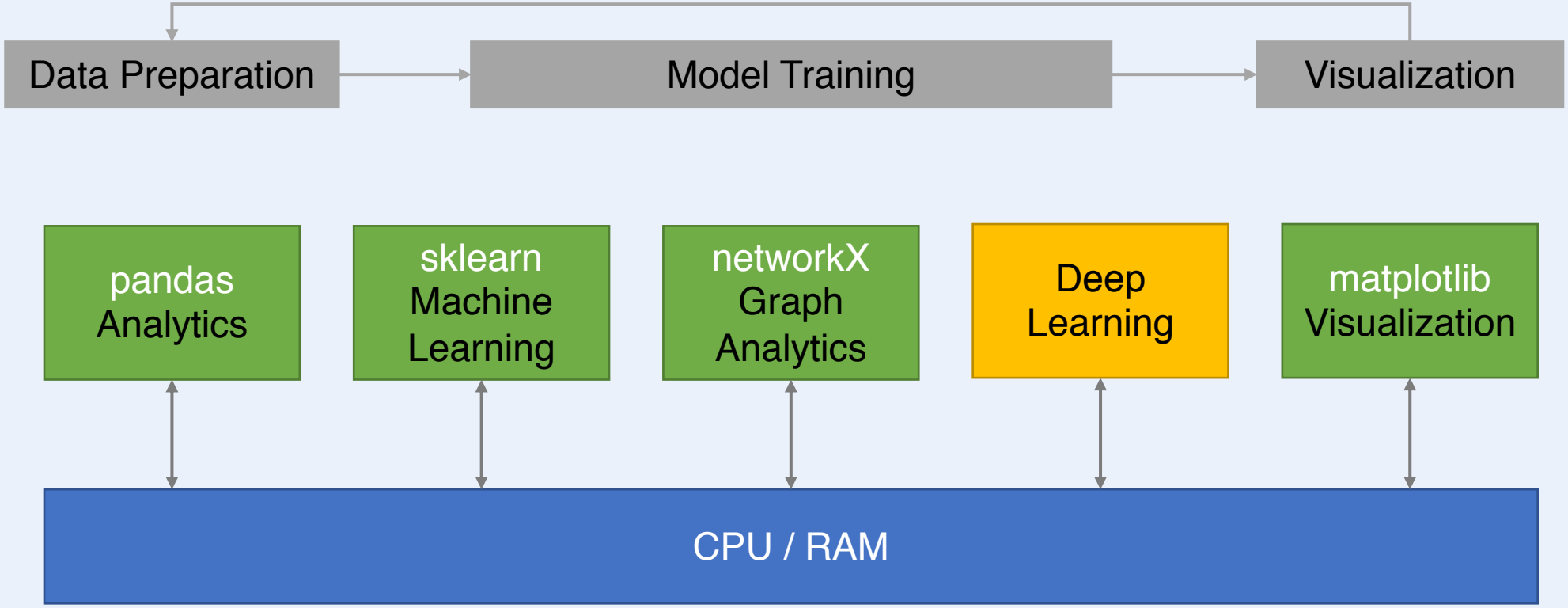
# INTRODUCING RAPIDS.AI

- Tool for **data science** tasks on GPU
- Backed by **NVIDIA**, but open-source
- Very ambitious **roadmap**
- **Active community**

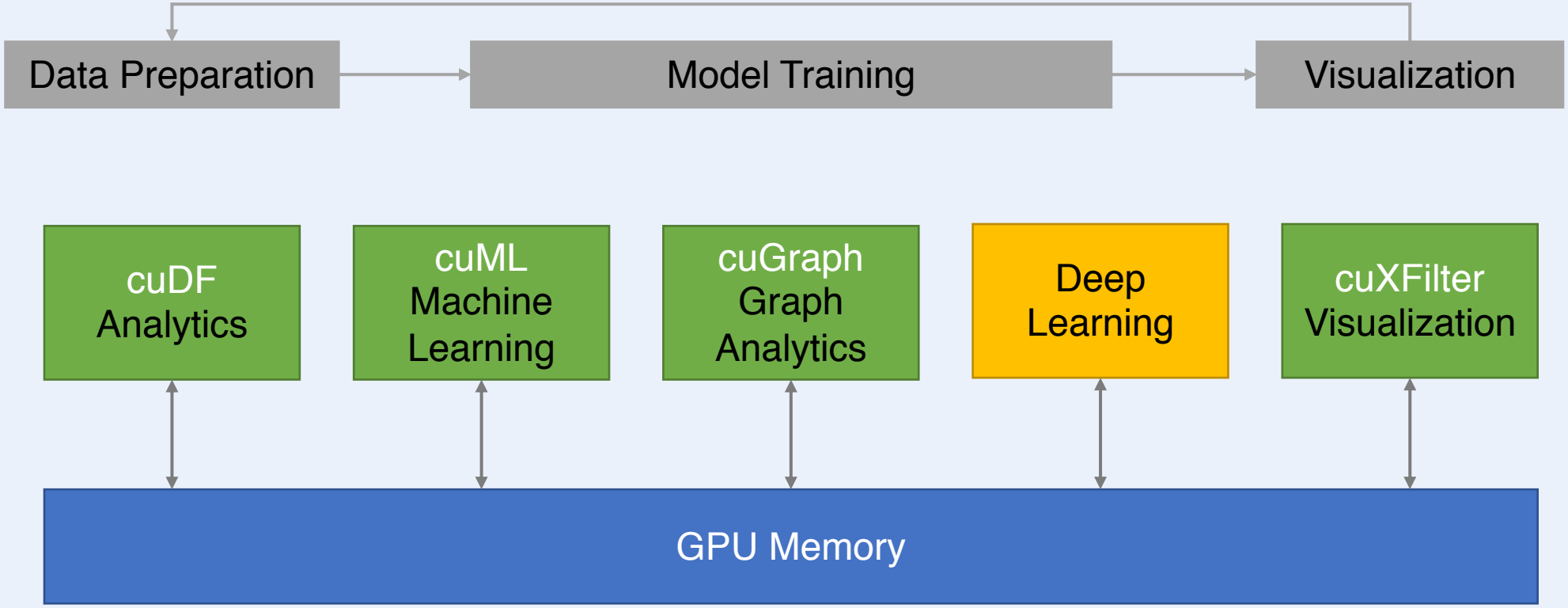



rapids.ai
github.com/rapidsai
medium.com/rapids-ai
@rapidsai on Twitter

# "TRADITIONAL" COMPONENTS



# RAPIDS COMPONENTS (1/2)



# COMPONENTS (2/2)

- Extract/Transform/Load with **cuDF**
  - Spatiotemporal data: **cuSpatial**
  - Strings: **nvStrings / cuStrings**

- Traditional Machine Learning, with **cuML**
  - Classification
  - Clustering
  - Dimensionality Reduction
  - Time Series
  - Gradient Boosting (Inference)

- Graph Analytics with **cuGraph**
- Visualization with **cuXFilter**

# TUTORIALS

- RAPIDS notebooks
  - From the RAPIDS team
    - http://www.github.com/rapidsai/notebooks


  - Community tutorials
    - http://www.github.com/rapidsai/notebooks-contrib

# Notebooks

- My own notebooks for this meetup:
  - https://github.com/arnaudwald/rapids-meetup-pydata


- Experiments done on NVIDIA Tesla P100 with CUDA 9.2

# SYNTAX (1/3)

```
import cudf

gdf = cudf.DataFrame()
gdf['some_column'] = [1, 2, 3]
```

Create an empty DataFrame, and add a column

```
import cudf

gdf = cudf.DataFrame({'a': [1, 2, 3],
                      'b': [4, 5, 6]})
```

Create a DataFrame from a dictionary

```
import cudf
path = './diabetes.csv'

gdf = cudf.read_csv(path, delimiter=',')
```

Load a CSV into a GPU DataFrame

```
import cudf
import pandas as pd

df = pd.read_csv(path, delimiter=',')
gdf = cudf.from_pandas(df)
```

Convert a pandas DataFrame into a GPU DataFrame

# Syntax (2/3)

```
gdf.head(5)
gdf.tail(4)
```

Get first/last rows of DataFrame

```
gdf.loc[2:5, ['Age', 'Glucose']]
gdf.query('Age > 65')
```

Filter data

```
age_mean = gdf['Age'].mean()
age_std = gdf['Age'].std()
```

Mean, standard deviation

```
age_counts = gdf['Age'].value_counts()
age_unique = gdf['Age'].nunique()
```

Number of occurrences, number of unique values

```
def double_age(age):
    return age * 2

gdf['Age'].applymap(double_age)
```

Transform values with a function

```
import numpy as np

gdf['Age'] = gdf['Age'].astype(np.float64)
```

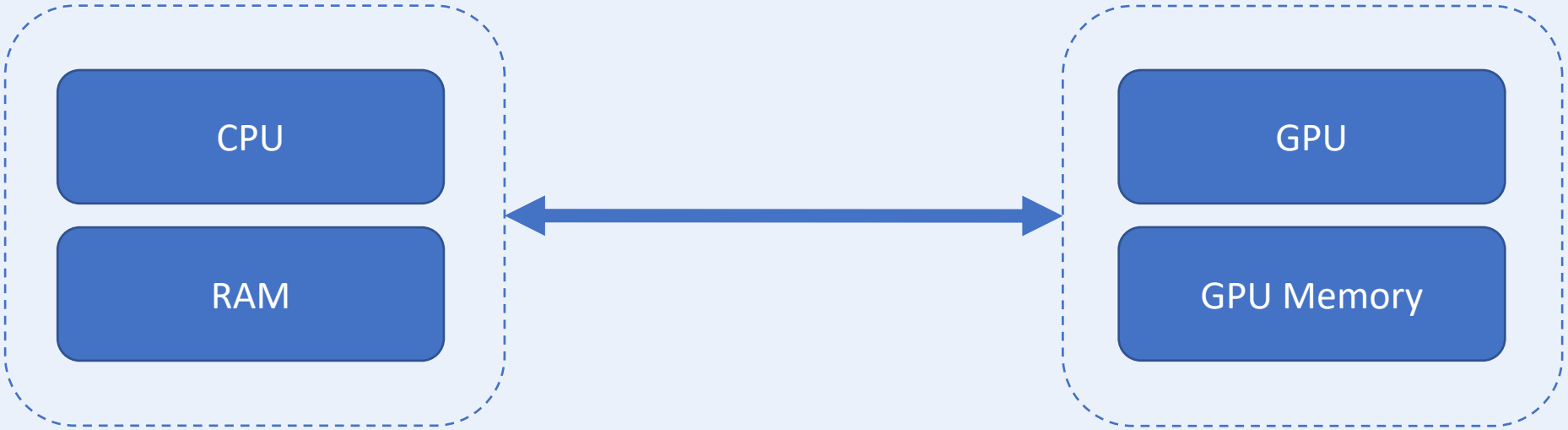
Change data type

# Syntax (3/3)

- Other popular methods
  - Sort values
  - Merge / Join / Concatenate
  - Group by / agg

- Limits
  - Plotting with matplotlib/seaborn
  - Covariance matrix
  - ➢ Have to use `gdf.to_pandas()` to get the full pandas API

# CPU/GPU Bottleneck

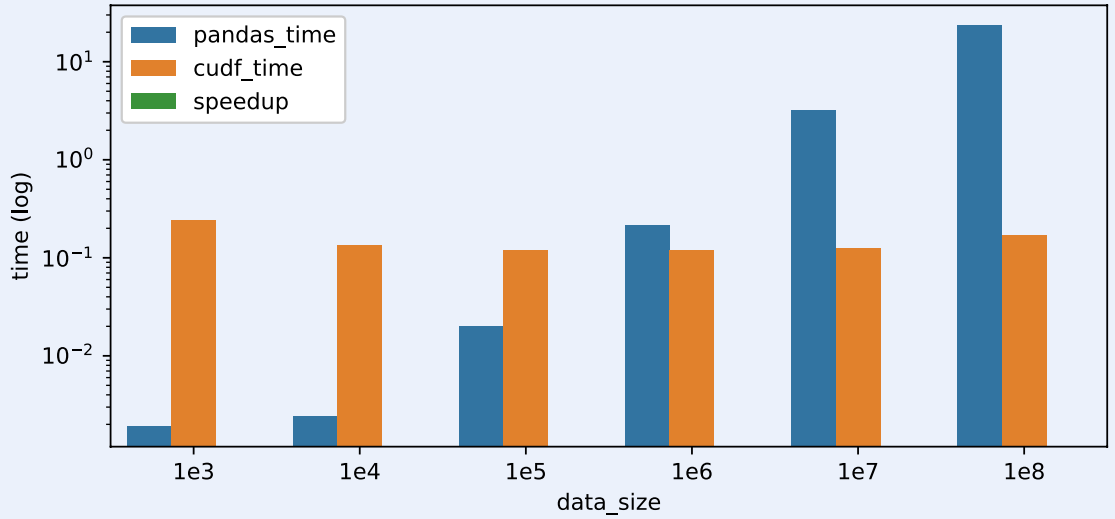


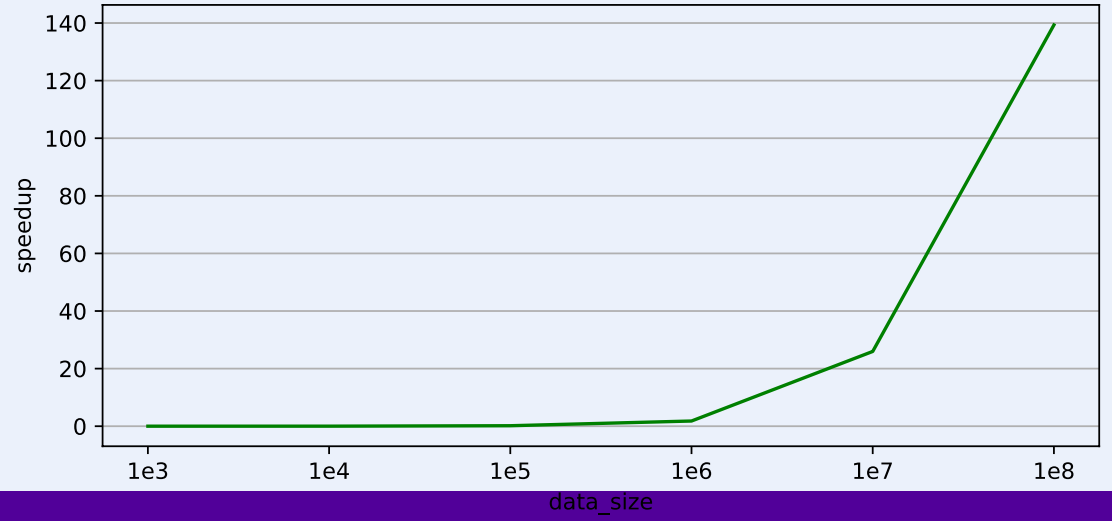
# Speedup & Trade-offs

| data_size | pandas_time (s) | cudf_time (s) | speedup |
|---|---|---|---|
| 1e3 | 0.001797 | 0.245914 | 0.007 |
| 1e4 | 0.002417 | 0.118758 | 0.02 |
| 1e5 | 0.020341 | 0.120459 | 0.2 |
| 1e6 | 0.211835 | 0.120047 | 1.8 |
| 1e7 | 2.247396 | 0.127621 | 18 |
| 1e8 | 24.353611 | 0.170310 | **143** |

- DataFrame
  - Single-column
  - data_size rows
- Operation
  - multiply each row by 2

# SPEEDUP & TRADE-OFFS




- DataFrame
  - Single-column
  - data_size rows
- Operation
  - multiply each row by 2

# Machine Learning (1/2)

- Objective: seamless integration
  - Replace `sklearn` by `cuML`

```
from cuml import PCA

pca = PCA(n_components)
pca.fit_transform(X)
```

```
from cuml.cluster import KMeans

kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
```

  - Most common algorithms implemented
  - Not all dataset processing methods are
    - Dataset generators, Scalers, K-folds…

# Machine Learning (2/2)

- One million rows
- 40 features

| Algorithm | Scikit-learn training time (s) | cuML training time (s) | Speedup |
|---|---|---|---|
| K-Means | 7.23 | 0.476 | **15** |
| PCA | 4.49 | 0.311 | **14** |
| Linear Regression | 1.68 | 0.080 | **21** |
| Random Forest Classification | 581 | 26.9 | **22** |

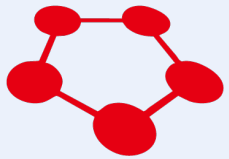
# Deep Learning



- DataFrame to Tensor conversion (naïve)

```
from torch import from_numpy
from torch.autograd import Variable

X = Variable(from_numpy(gdf.as_matrix()))
X = X.to('cuda:0')
```



Chainer

- Stay in GPU Memory with DLPack by dmlc

```
from torch.utils.dlpack import from_dlpack


X = from_dlpack(gdf.to_dlpack())
```

Distributed (Deep) Machine Learning Community

A Community of Awesome Machine Learning Projects

# How to scale pipelines

**CPU**





**GPU**


**Multi-Node, Multi-GPU**

# Why is it interesting

- Very **easy** to learn
- Very **fast** compared to single core, single node pandas (see experiments)

- **Machine Learning**
  - Large datasets
  - ETL can be long
  - You're probably already on GPU for training

# Current Limitations

- No support for the latest NVIDIA CUDA drivers (10.1 & up)
  - Versions 9.2 and 10.0 supported
- Supports XGBoost and LightGBM for inference only
  - Forest Inference Library (FIL)
  - Up to 100x faster
- dask-cudf has been merged with cudf in update 0.9
  - Tutorials have not been updated

# More GPU/RAPIDS Tools

**BlazingSQL**

- GPU-accelerated SQL engine


https://blazingsql.com

**Nuclio**

- Serverless ML application, with GPU support


https://nuclio.io

# How to get started

- On your machine (with GPU !)

```
conda install -c rapidsai -c nvidia -c numba -c conda-forge -c anaconda \
        cudf=0.9 cuml=0.9 cugraph=0.9 python=3.7 anaconda::cudatoolkit=9.2
```

- With docker

```
docker pull rapidsai/rapidsai:cuda9.2-runtime-ubuntu16.04

docker run --runtime=nvidia \
        --rm -it \
        -p 8888:8888 \
        -p 8787:8787 \
        -p 8786:8786 \
        rapidsai/rapidsai:cuda9.2-runtime-ubuntu16.04
```

RAPIDS

Scaleway

PyData

# THANK YOU!

Arnaud Wald

http://linkedin.com/in/arnaudwald
http://github.com/arnaudwald