OPEN DATA SCIENCE CONFERENCE

San Francisco | Oct. 28 - Nov. 1, 2019

mlflow: Platform for Complete Machine Learning Lifecycle

https://dbricks.co/odsc-tutorial
https://mlflow.org

Jules S. Damji
ODSC West San Francisco

Oct 29 , 2019
@2twitme

databricks

Guest WiFi:

Password:

I have used ML Frameworks Before…

$ whoami

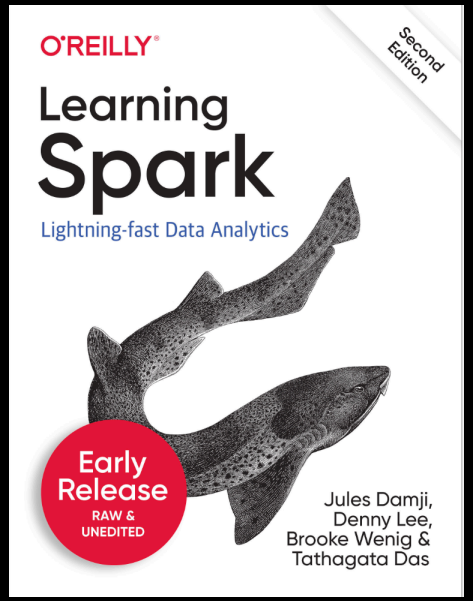
Apache Spark Developer & Community Advocate @ Databricks

Developer Advocate @ Hortonworks

Software engineering @ Sun Microsystems, Netscape, @Home, Excite@Home, VeriSign, Scalix, Centrify, LoudCloud/Opsware, ProQuest

Program Chair Spark + AI Summit

https://www.linkedin.com/in/dmatrix
@2twitme



databricks

VISION

Accelerate innovation by unifying data science, engineering and business

---

SOLUTION

Unified Data Analytics Platform

WHO WE ARE

- Original creators of 

- 2000+ global companies use our platform across big data & machine learning lifecycle

Outline

- Overview of ML development challenges
- How MLflow tackles these
- MLflow Components
- Managed MLflow ML + Registry Demo
- Q & A
- (Break?)
- Set up Environment
- Hands of Tutorial

Machine Learning Development is Complex

ML Development Challenges

1. Zoo of software tools
2. Tracking & reproducing results
3. Productionizing models
4. Scaling
5. Different from Traditional Software Development

Traditional Software

Goal: Meet a functional specification

Quality depends only on code

Typically pick one software stack

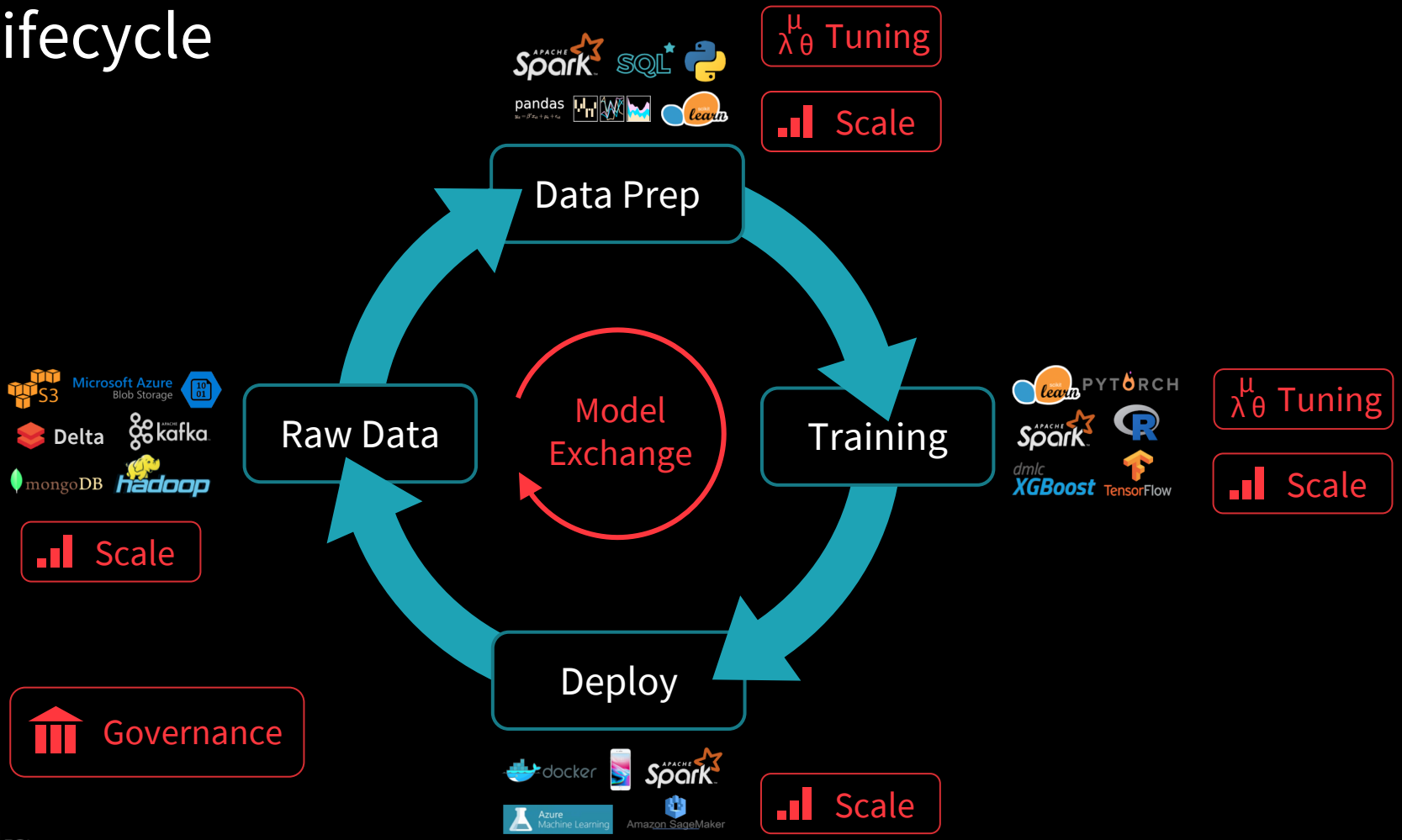
Machine Learning

Goal: Optimize a metric (e.g., accuracy)
Constantly experiment to improve it

Quality depends on input data and tuning parameters

Compare + combine many libraries, models & algorithms for the same task

ML Lifecycle



databricks

Custom ML Platforms

Facebook FBLearner, Uber Michelangelo, Google TFX

- + **Standardize the data prep / training / deploy loop: if you work with the platform, you get these!**
- – **Limited to a few algorithms or frameworks**
- – **Tied to one company's infrastructure**
- – **Out of luck if you left the company….**

Can we provide similar benefits in an open manner?

Introducing mlflow

Open machine learning platform

- Works with popular ML library & language
- Runs the same way anywhere (e.g., any cloud or locally)
- Designed to be useful for 1 or 1000+ person orgs
- *Simple. Easy-to-use.*
- *Offers positive Developer Experience to get started!*

MLflow Design Philosophy

"API-first"

- Submit runs, log models, metrics, etc. from popular library & language
- Abstract "model" lambda function that MLflow can then deploy in many places (Docker, Azure ML, Spark UDF)
- Open interface allows easy integration

Key enabler: built around REST APIs and CLI

Modular design

- Allow different components individually (e.g., use MLflow's project format but not its deployment tools)
- Not monolithic
- But Distinctive and Selective

Key enabler: distinct components (Tracking/Projects/Models)

MLflow Components

mlflow Tracking

Record and query experiments: code, configs, results, … etc.

mlflow Projects

Packaging format for reproducible runs on any platform

mlflow Models

General model format that supports diverse deployment tools

mlflow Registry

Repository of named, versioned models w/tags, comments , etc

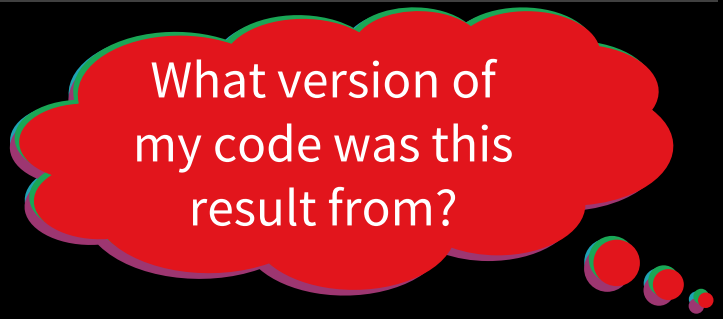
Model Development without MLflow

```
data   = load_text(file)
ngrams = extract_ngrams(data, N=n)
model  = train_model(ngrams,
             learning_rate=lr)
score  = compute_accuracy(model)

print("For n=%d, lr=%f: accuracy=%f"
      % (n, lr, score))

pickle.dump(model, open("model.pkl"))
```

```
For n=2, lr=0.1: accuracy=0.71
For n=2, lr=0.2: accuracy=0.79
For n=2, lr=0.5: accuracy=0.83
For n=2, lr=0.9: accuracy=0.79
For n=3, lr=0.1: accuracy=0.83
For n=3, lr=0.2: accuracy=0.82
For n=4, lr=0.5: accuracy=0.75
...
```



Key Concepts in Tracking

Parameters: key-value inputs to your code

Metrics: numeric values (can update over time)

Tags and Notes: information about a run

Artifacts: files, data and models

Source: what code ran?

Version: what of the code?

MLflow Tracking API: *Simple!*

mlflow
Tracking

Record and query experiments: code, configs, results, …etc

```
import mlflow

 # log model's tuning parameters

with mlflow.start_run():
  mlflow.log_param("layers", layers)
  mlflow.log_param("alpha", alpha)

  # log model's metrics
  mlflow.log_metric("mse", model.mse())
  mlflow.log_artifact("plot", model.plot(test_df))
  mlflow.tensorflow.log_model(model)
```

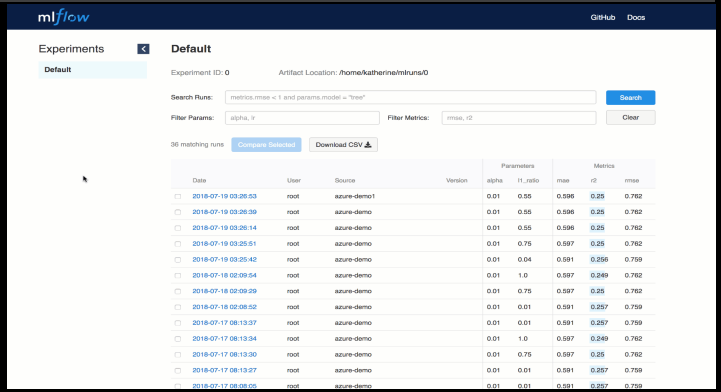
Model Development *with* MLflow is *Simple!*

```
data    = load_text(file)
ngrams = extract_ngrams(data, N=n)
model   = train_model(ngrams,
                learning_rate=lr)
score   = compute_accuracy(model)

mlflow.log_param("data_file", file)
mlflow.log_param("n", n)
mlflow.log_param("learning_rate", lr)
mlflow.log_metric("score", score)

mlflow.sklearn.log_model(model)
```

```
$ mlflow ui
```

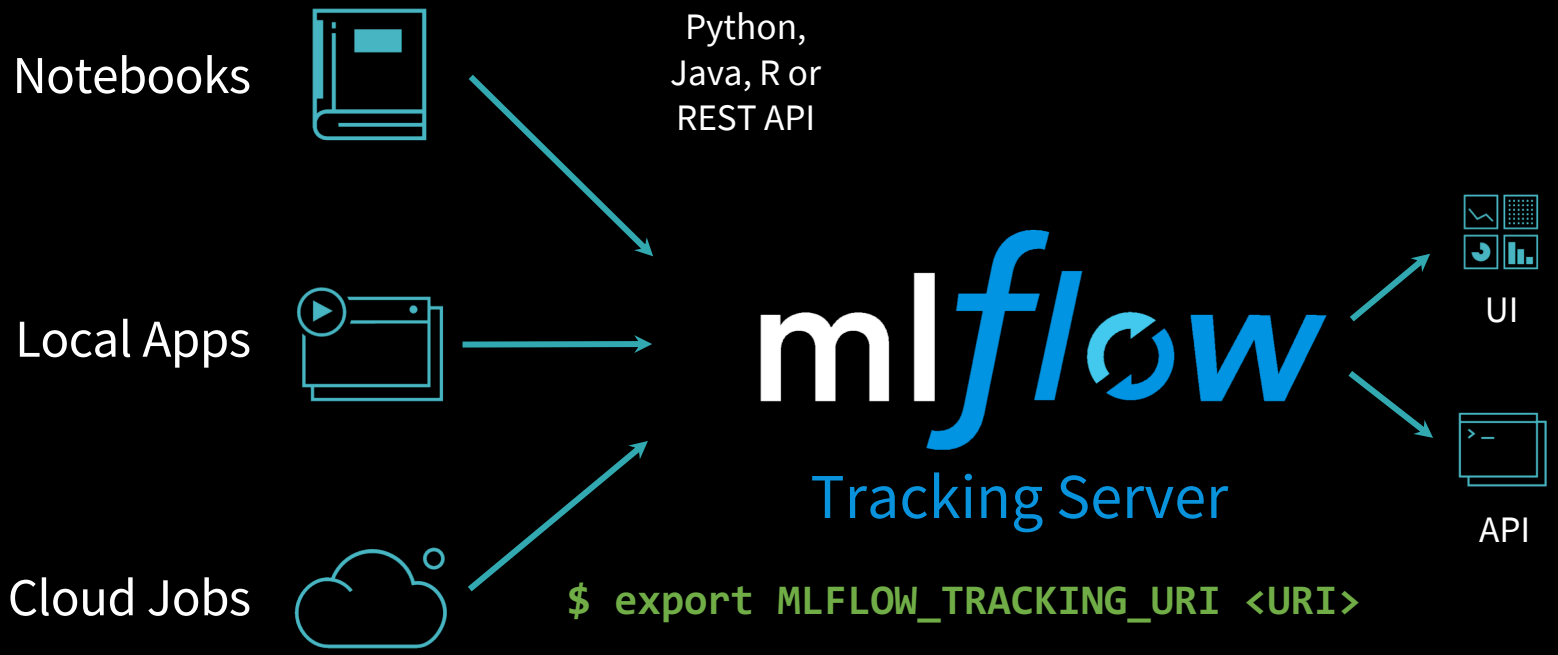


Track parameters, metrics, output files & code version

Search using UI or API

databricks

MLflow Tracking

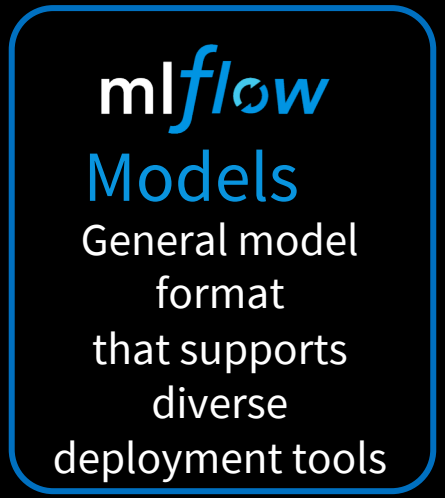


MLflow Tracking Backend Stores

1. **Entity (Metadata) Store**
   - FileStore (local filesystem)
   - SQLStore (via SQLAlchemy)
     - PostgreSQL, MySQL, SQLlite

2. **Artifact Store**
   - S3 backed store
   - Azure Blob storage
   - Google Cloud storage
   - DBFS artifact repo

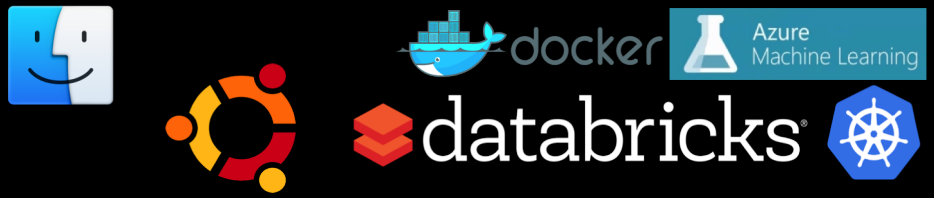
MLflow Components



MLflow Projects Motivation

Diverse set of tools

Diverse set of environments

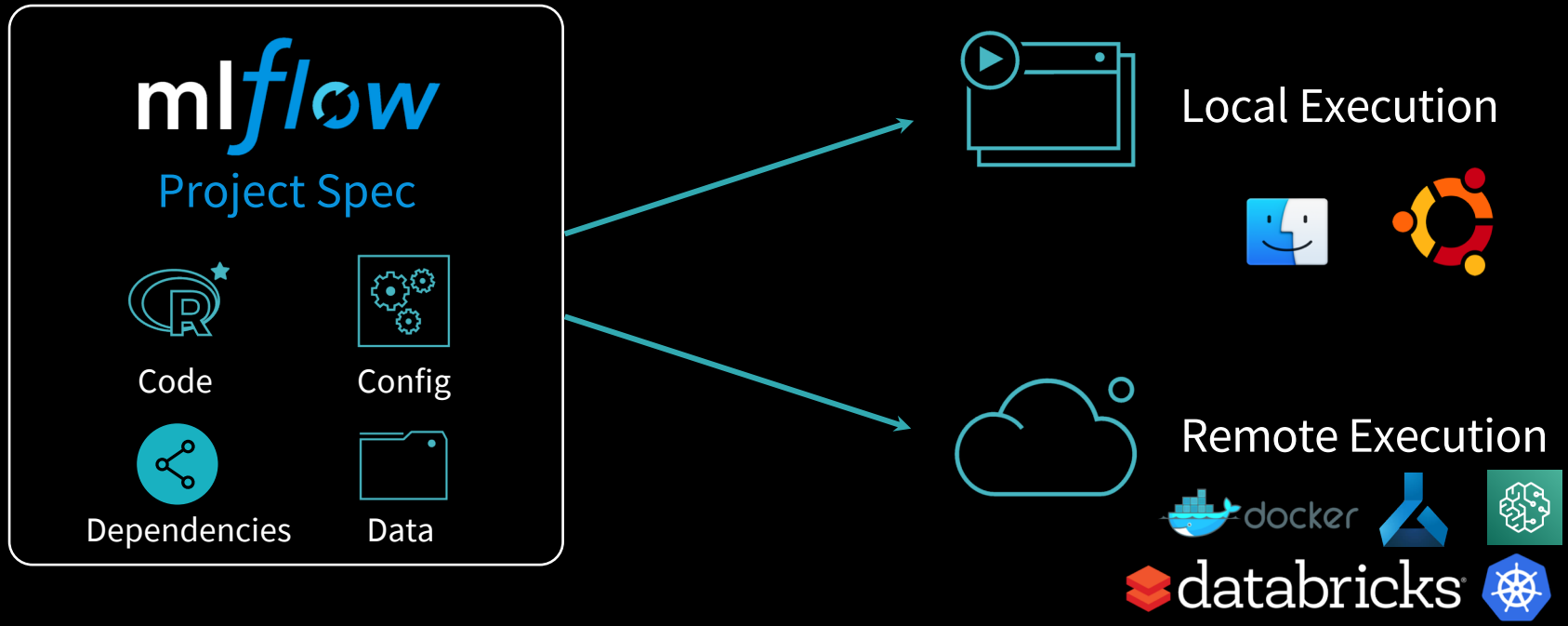


mlflow Projects

Packaging format
for reproducible
runs
on any platform

Challenge: ML results difficult to reproduce

MLflow Projects



MLflow Projects

Packaging format for reproducible ML runs

- Any code folder or GitHub repository
- Optional MLproject file with project configuration

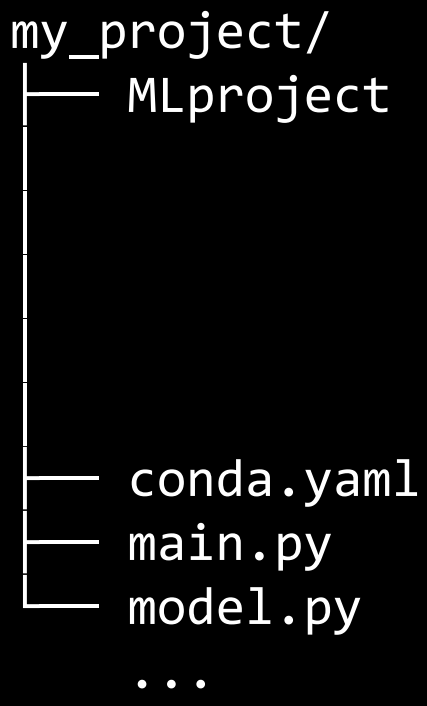
Defines dependencies for reproducibility

- Conda (+ R, Docker, …) dependencies can be specified in MLproject
- Reproducible in (almost) any environment

Execution API for running projects

- CLI / Python / R / Java
- Supports local and remote execution

Example MLflow Project




```
conda_env: conda.yaml

entry_points:
  main:
    parameters:
      training_data: path
      lambda: {type: float, default: 0.1}
    command: python main.py {training_data} {lambda}
```

```
$ mlflow run git://<my_project> -P lambda=0.2

mlflow.run("git://<my_project>", ...)

mlflow run . -e main -P lambda=0.2
```

MLflow Components

mlflow Tracking

Record and query experiments: code, configs, results, … etc.

mlflow Projects

Packaging format for reproducible runs on any platform

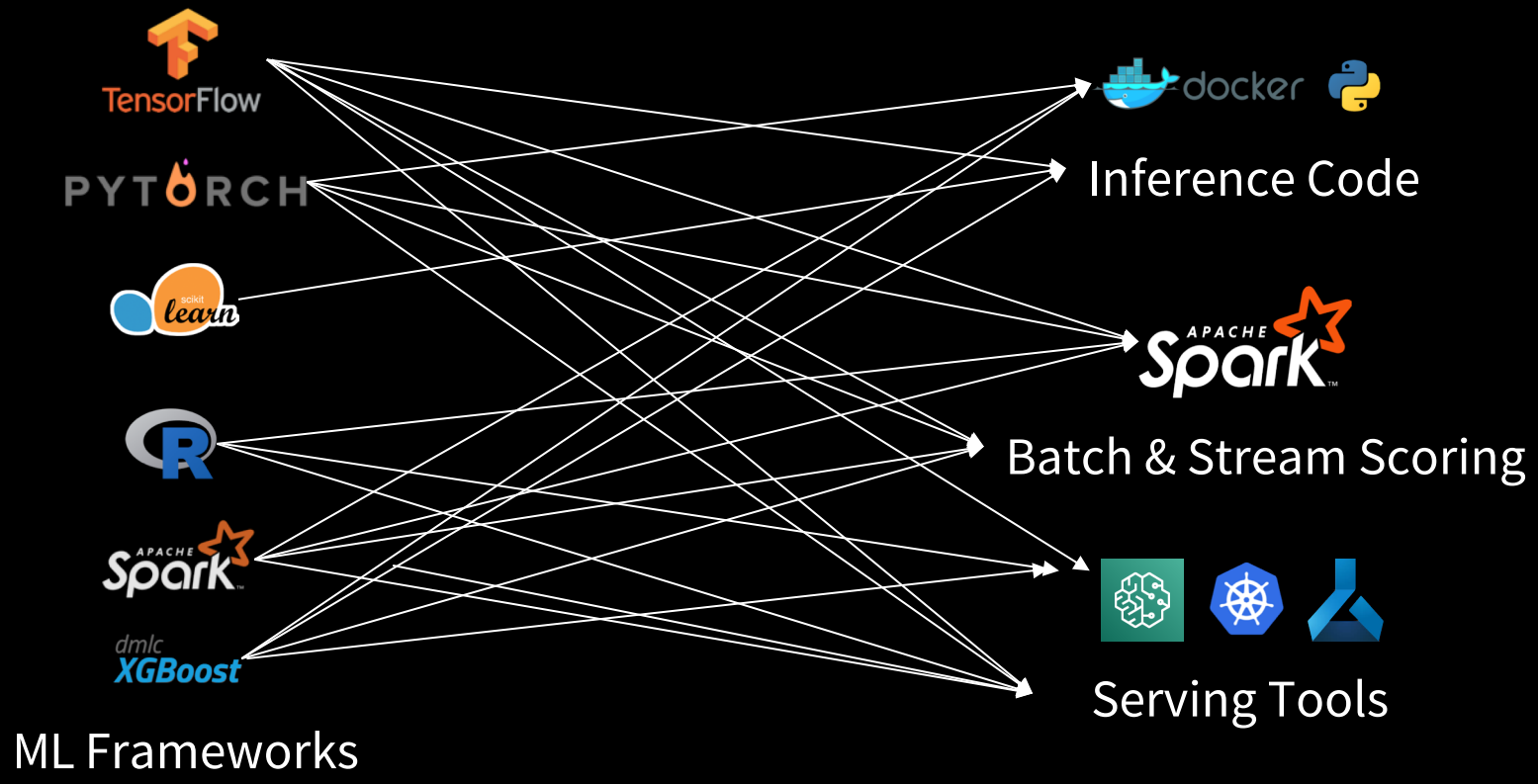
mlflow Models

General model format that supports diverse deployment tools

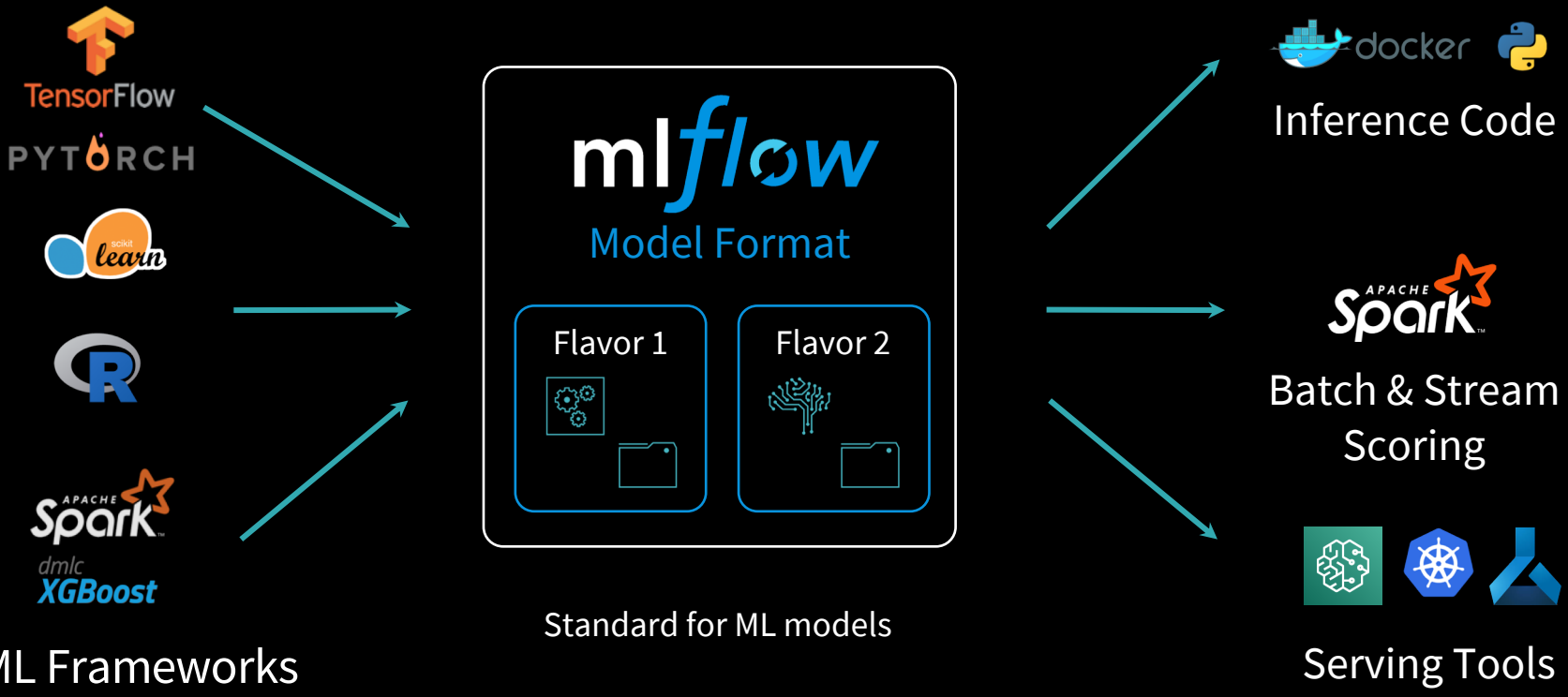
mlflow Registry

Repository of named, versioned models w/tags, comments , etc

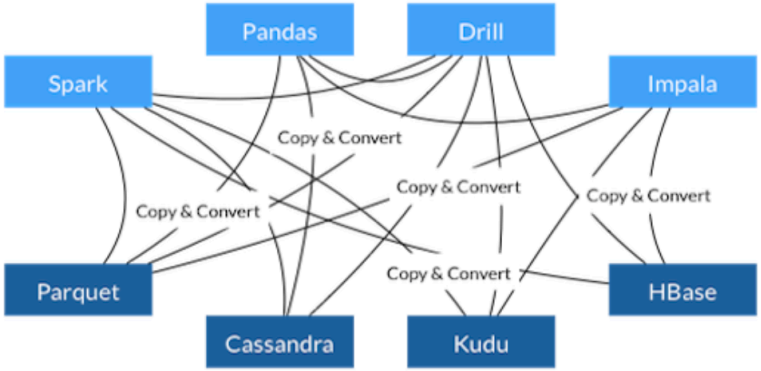
MLflow Model Motivation



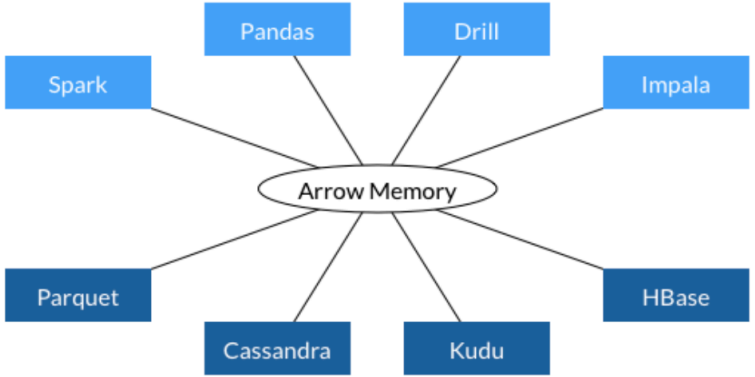
MLflow Models



Advantages of a Common Data Layer



- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects



- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (eg, Parquet-to-Arrow reader)

MLflow Models

Packaging format for ML Models

- Any directory with MLmodel file

Defines dependencies for reproducibility

- Conda environment can be specified in MLmodel configuration

Model creation utilities

- Save models from any framework in MLflow format

Deployment APIs

- CLI / Python / R / Java

Example MLflow Model

`mlflow.tensorflow.log_model(...)`

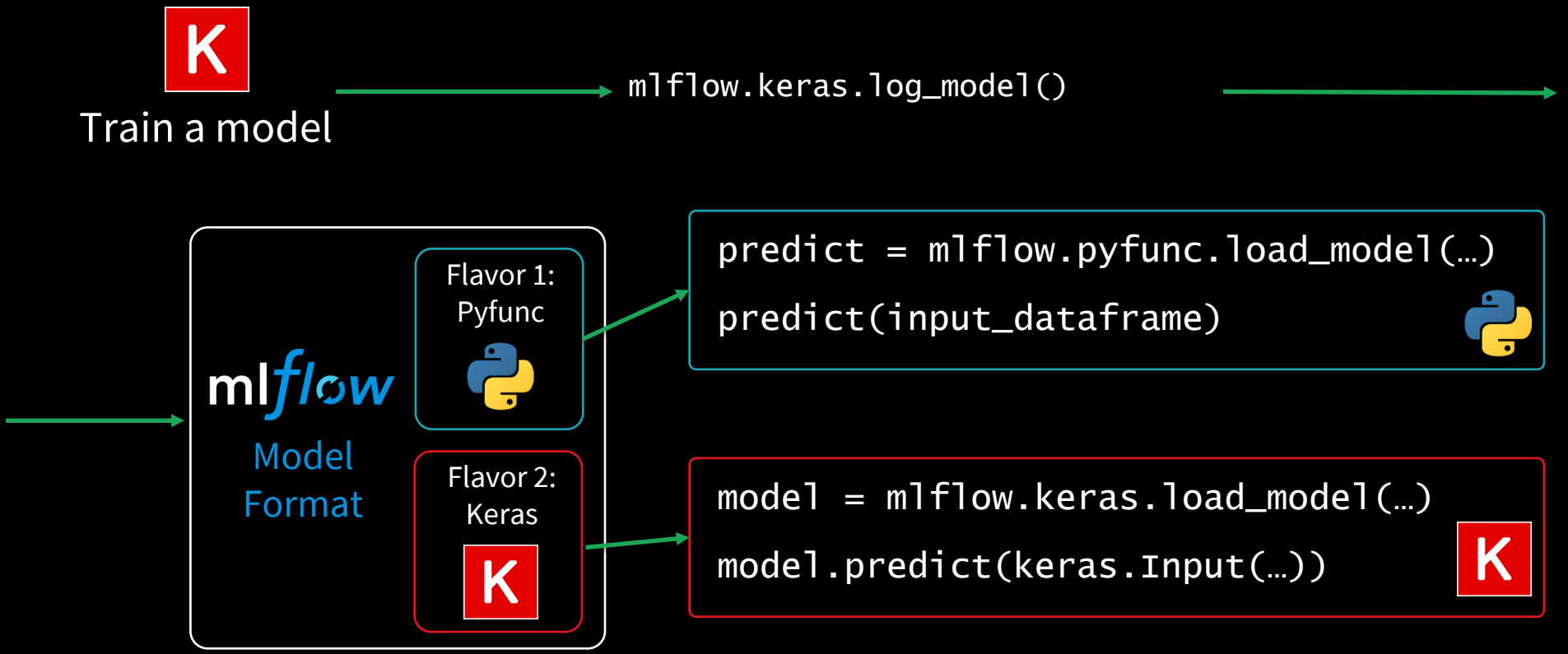
```
my_model/
├── MLmodel
└── estimator/
    ├── saved_model.pb
    └── variables/
        ...
```

```
run_id: 769915006efd4c4bbd662461
time_created: 2018-06-28T12:34
flavors:
  tensorflow:
    saved_model_dir: estimator
    signature_def_key: predict
  python_function:
    loader_module: mlflow.tensorflow
```

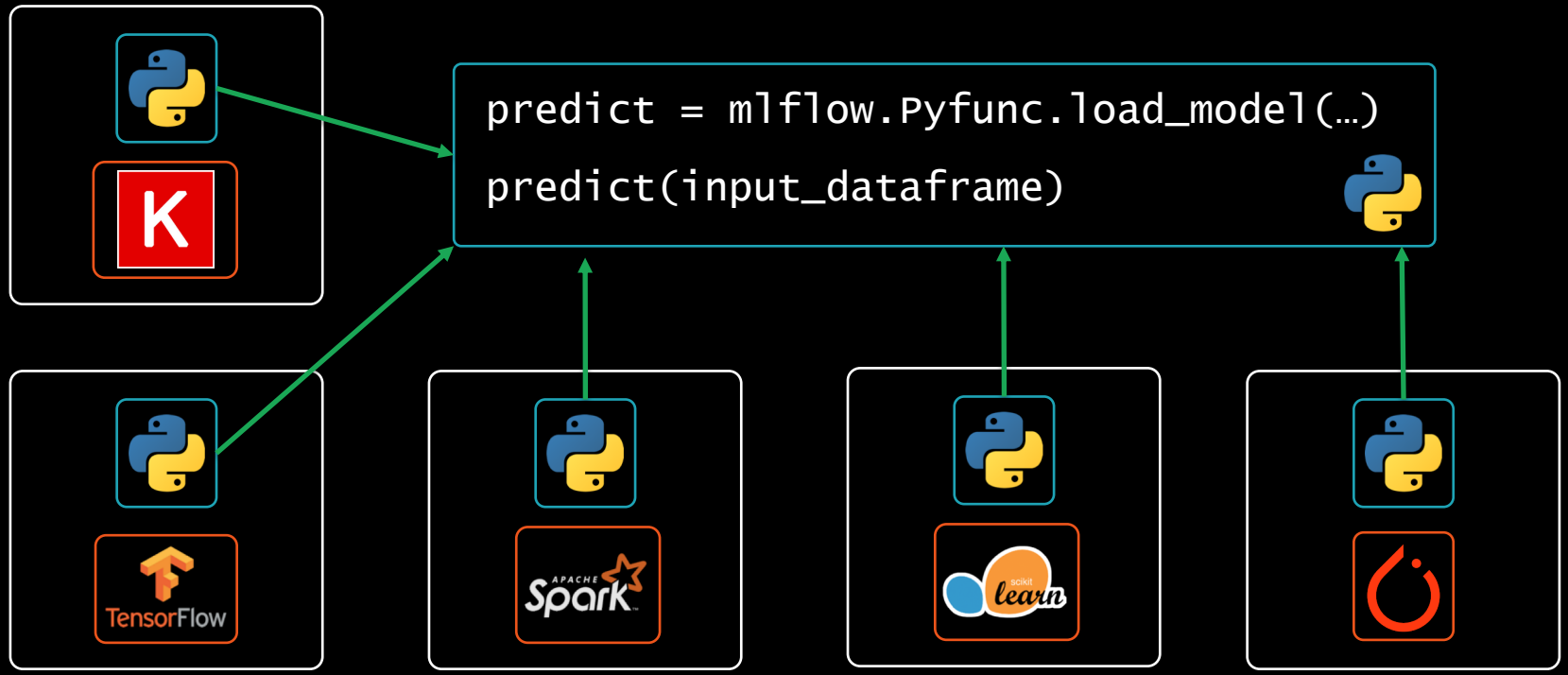
Usable by tools that understand TensorFlow model format

Usable by any tool that can run Python (Docker, Spark, etc!)

Model Flavors Example



Model Flavors Example



MLflow Components

mlflow Tracking

Record and query experiments: code, configs, results, … etc.

mlflow Projects

Packaging format for reproducible runs on any platform

mlflow Models

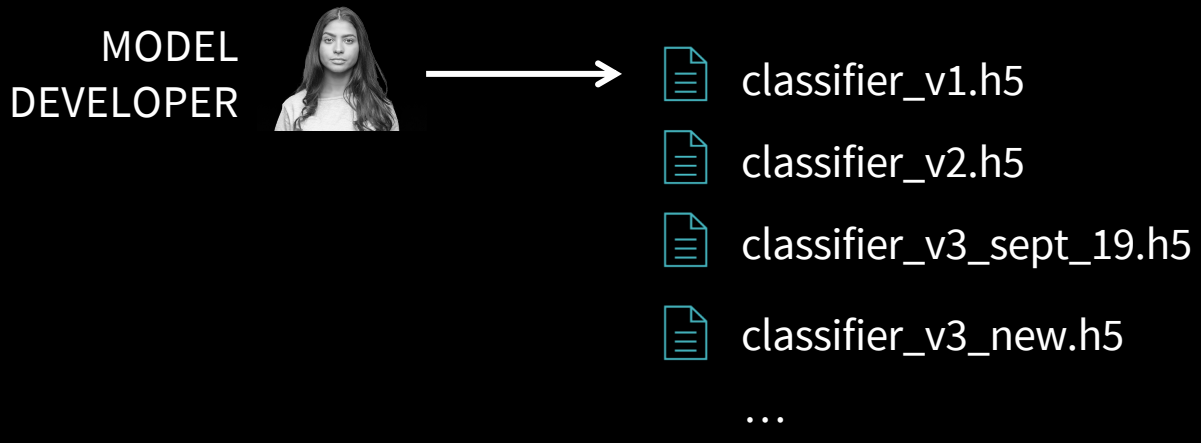
General model format that supports diverse deployment tools

mlflow Registry

Repository of named, versioned models w/tags, comments , etc

The Model Management Problem

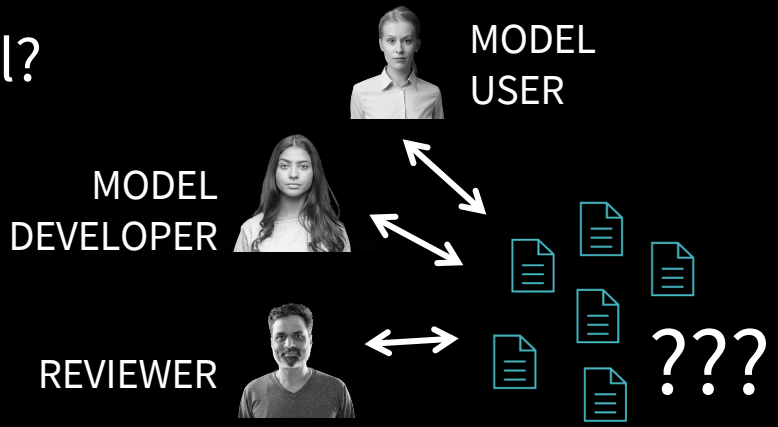
When you're working on one ML app alone, storing your models in files is manageable



The Model Management Problem

When you work in a **large organization** with **many models**, management becomes a major challenge:

- Where can I find the best version of this model?
- How was this model trained?
- How can I track docs for each model?
- How can I review models?

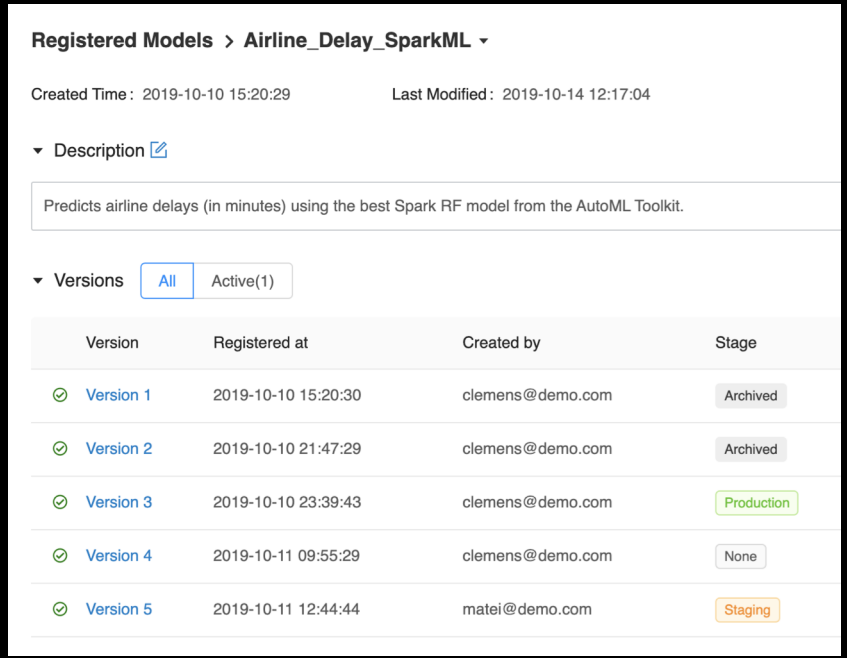


MLflow Model Registry

Repository of named, versioned models with comments & tags

Track each model's stage: dev, staging, production, archived

Easily load a specific version



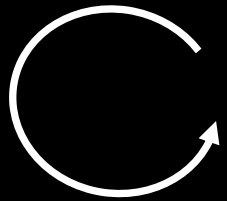
Model Registry Workflow API

```
mlflow.register_model("runs:/e148278107274..832/artifacts/model",
            "EmailSpamClassifier")
```

```
client = mlflow.tracking.Mlflowclient()
client.get_latest_versions(name =” EmailSpamClassifier",
stages = [" Production "])
```

MODEL DEVELOPER

REVIEWERS, CI/CD TOOLS



DOWNSTREAM USERS

AUTOMATED JOBS

REST SERVING

```
client = mlflow.tracking.Mlflowclient()
client.update_model_version(name = "EmailSpamClassifier ",
                            version = 12,
                           stage = "Production")
```

Model Registry Availability

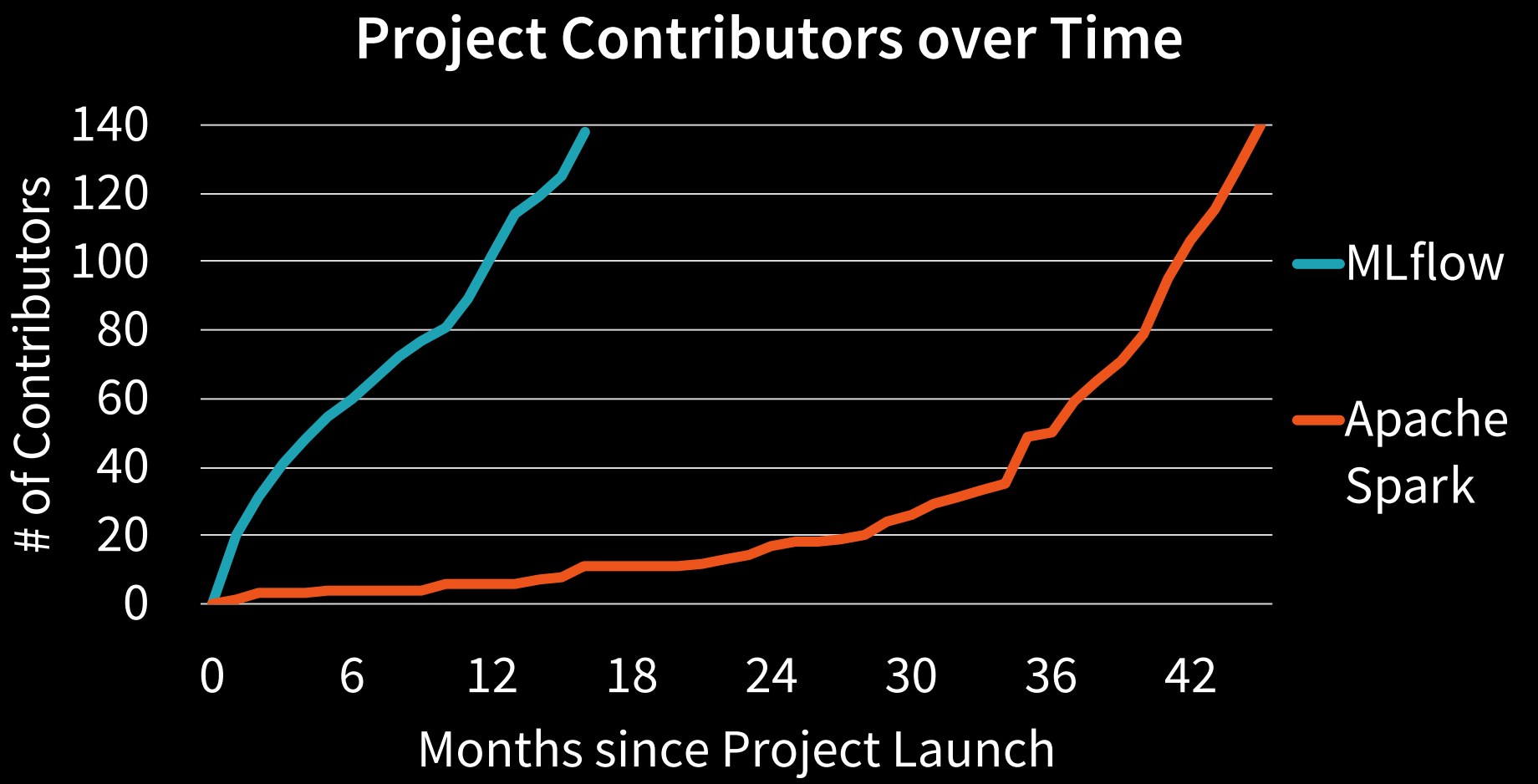
Pull request available: tinyurl.com/registry-pr

Available to Databricks customers

https://mlflow.org/docs/latest/model-registry.html

New in Last 6 Months

- MLflow 1.0 (and 1.1, 1.2, 1.3)
  - Model Registry
  - Support for TensorFlow 2.0
  - TensorFlow/Keras auto-logging
  - “Step” axis for metrics & Improved UI
  - Improved search capabilities
  - Support for ONNX models
  - Kubernetes, HDFS & Seldon Integrations


Project Contributors over Time
# of Contributors
140
120
100
80
60
40
20
0
0
6
12
18
24
30
36
42
Months since Project Launch
MLflow
Apache Spark

What Did We Talk About? mlflow

Workflow tools can greatly simplify the ML lifecycle

- Simplify lifecycle development
- Lightweight, open platform that integrates easily
- Available APIs: Python, Java & R (Soon Scala)
- Easy to install and use
- Develop locally and track locally or remotely
- Deploy locally, cloud, on premise…
- Visualize experiments

https://academy.databricks.com

Setup Environment & Hands on Tutorials

https://dbricks.co/odsc-tutorial
https://mlflow.org

Thank You ☺

Q & A

jules@databricks.com
@2twitme
https://www.linkedin.com/in/dmatrix/