Platform for Machine Learning Lifecycle

Jules S. Damji
@2twitme

Outline – Introduction to MLflow: How to Use MLflow Tracking - Module 1

- Overview of ML development challenges
- How MLflow tackles these
- Concepts and Motivations
- MLFlow Components
  - MLflow Tracking
  - How to use MLflow Tracking APIs
  - Use Databricks Community Edition
  - Explore MLflow UI
  - Tutorials & Exercises
- Q & A

https://github.com/dmatrix/olt-mlflow

Machine Learning Development is Complex

Traditional Software vs. Machine Learning

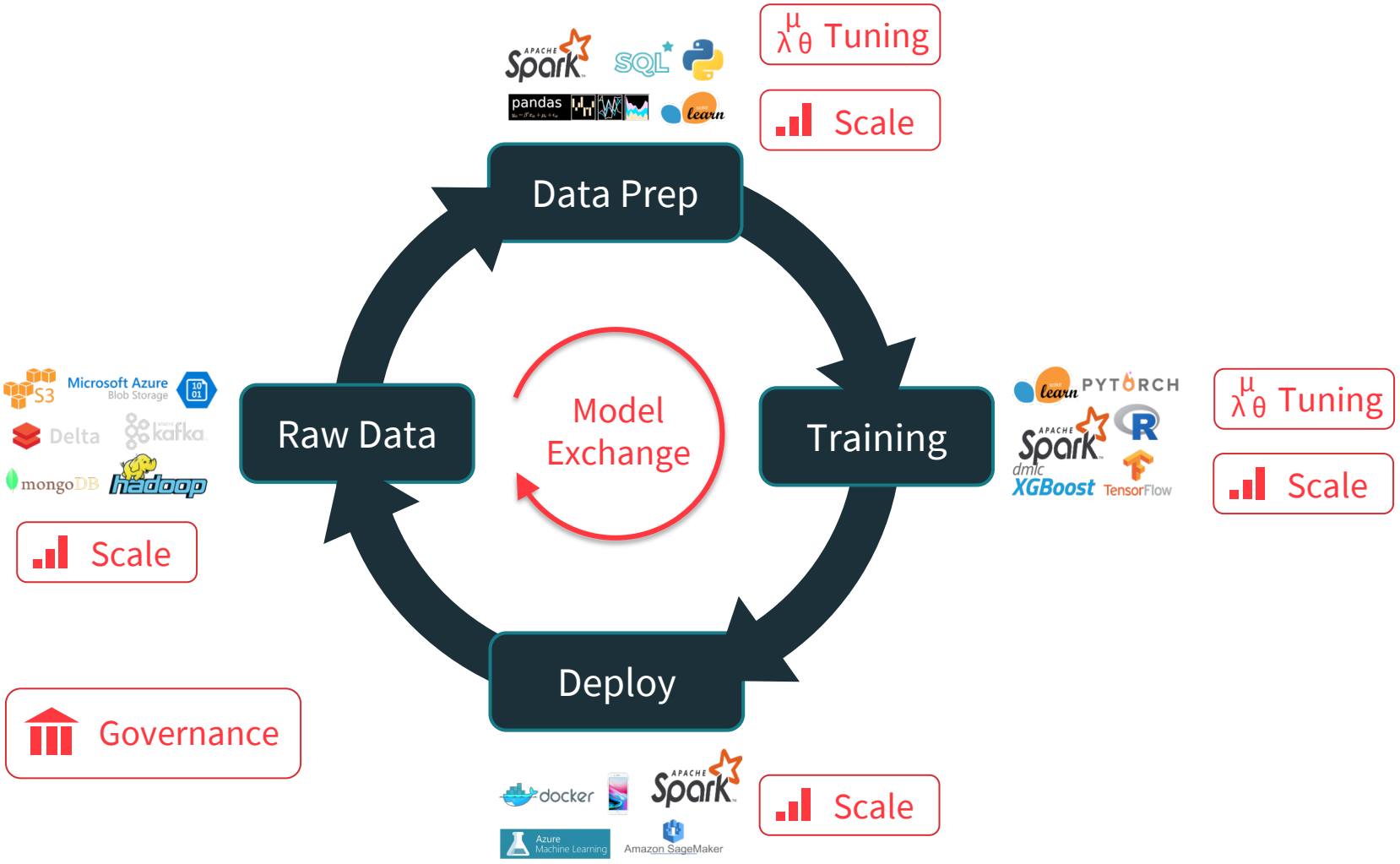
Traditional Software

- Goal: Meet a functional specification
- Quality depends only on code
- Typically pick one software stack w/ fewer libraries and tools
- Limited deployment environments

Machine Learning

- Goal: Optimize metric (e.g., accuracy. Constantly experiment to improve it
- Quality depends on input data and tuning parameters
- Compare + combine many libraries, model
- Diverse deployment environments

Machine Learning Lifecycle



Custom Machine Learning Platforms

Some Big Data Companies

+ 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. Modular. 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 from the community

Key enabler: built around Programmatic APIs, REST APIs & 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/Registry)

MLflow Components

mlflow

Tracking

Record and query experiments: code, data, config, and results

mlflow

Projects

Package data science code in a format that enables reproducible runs on many platform

mlflow

Models

Deploy machine learning models in diverse serving environments

new

mlflow

Model Registry

Store, annotate and manage models in a central repository

databricks.com/mlflow

mlflow.org

github.com/mlflow

twitter.com/MLflow

databricks

O'REILLY®

Key Concepts in MLflow 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?

Run: an instance of code that runs by MLflow

Experiment: {Run, … Run}

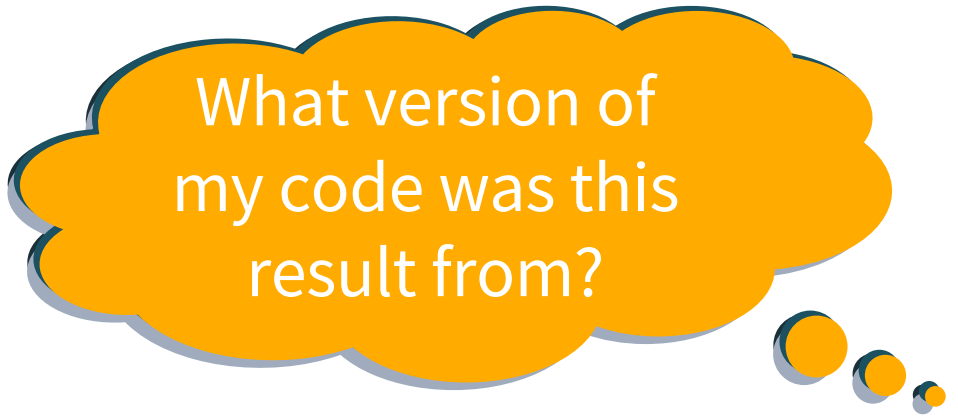
Model Development without MLflow Tracking

```
data   = load_text(file_name=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"))
```

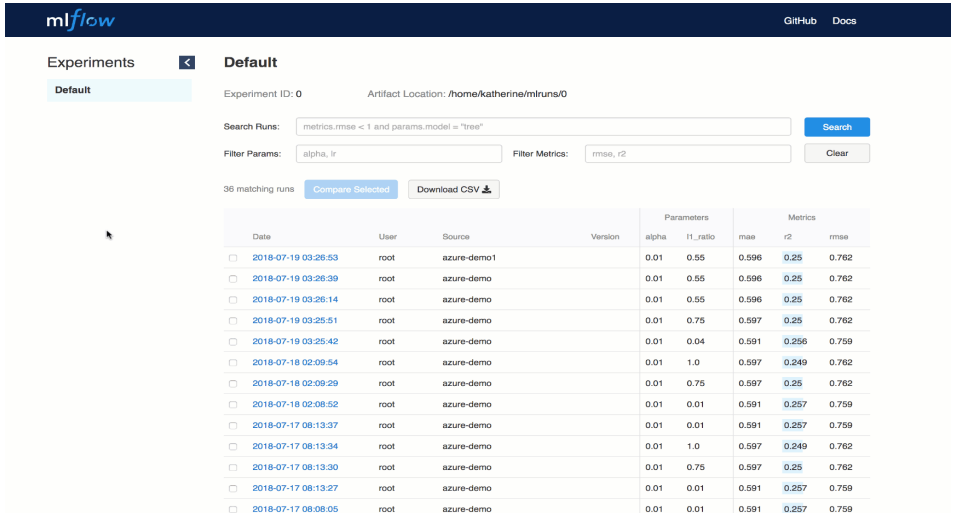
```
For n=2, lr=0.1: accuracy=0.71
For n=2, lr=0.2: accuracy=0.79
For n=2, lr=0.5: accuracy=0.83
For n=2, lr=0.9: accuracy=0.79
For n=3, lr=0.1: accuracy=0.83
For n=3, lr=0.2: accuracy=0.82
For n=4, lr=0.5: accuracy=0.75
...
```



Model Development *with* MLflow is *Simple!*

```
import mlflow
data   = load_text(file_name=file)
ngrams = extract_ngrams(data, N=n)
model  = train_model(ngrams,
                     learning_rate=lr)
score  = compute_accuracy(model)
with mlflow.start_run():
  mlflow.log_param("data_file", file)
  mlflow.log_param("n", n)
  mlflow.log_param("learn_rate", lr)
  mlflow.log_metric("score", score)
  mlflow.sklearn.log_model(model)
```

```
$ mlflow ui
```



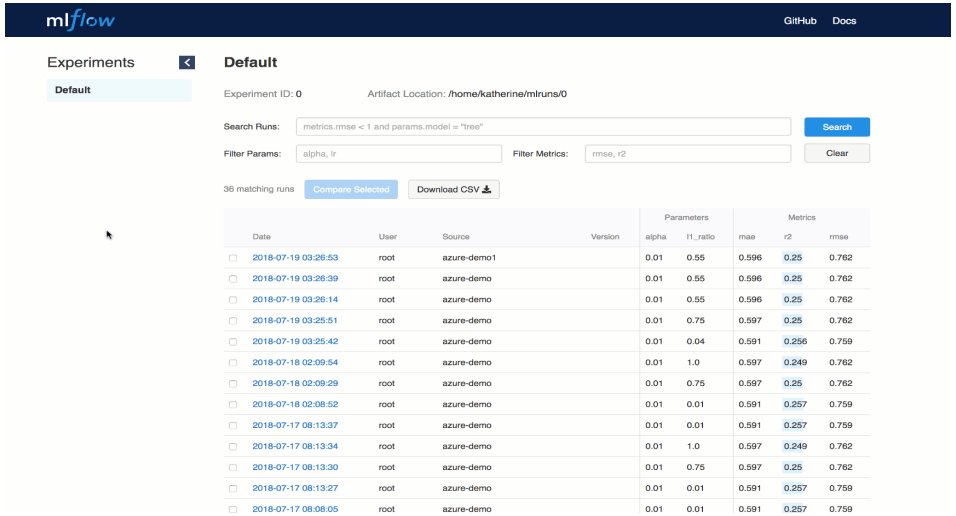
Track parameters, metrics, artifacts, output files & code version

Model Development *with* MLflow is *Simple!* mlflow.<flavor>.autolog() APIs

```
import mlflow
import mlflow.keras

# Build, compile, enable autologging, and
train your model
keras_model = ...
keras_model.compile(optimizer="rmsprop”,
        loss="mse”,
        metrics=["accuracy"])
# autolog your metrics, parameters, and
model
mlflow.keras.autolog()
results = keras_model.fit( x_train, y_train,
        epochs=20, batch_size=128,
        validation_data=(x_val, y_val))
```

```
$ mlflow ui
```



Track parameters, metrics, artifacts, output files & code version

O'REILLY®

Model Development *with* MLflow is *Simple!*
mlflow.<flavor>.autolog() APIs

mlflow Auto-Logging

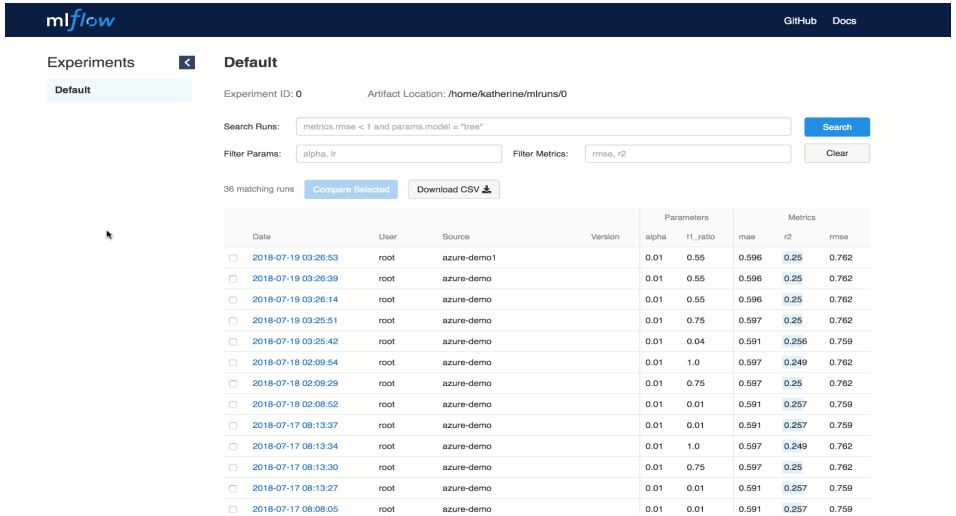
Currently supported libraries:

TensorFlow · Keras · GLUON · LightGBM · dmlc XGBoost · Apache Spark · fast.ai

Coming soon:

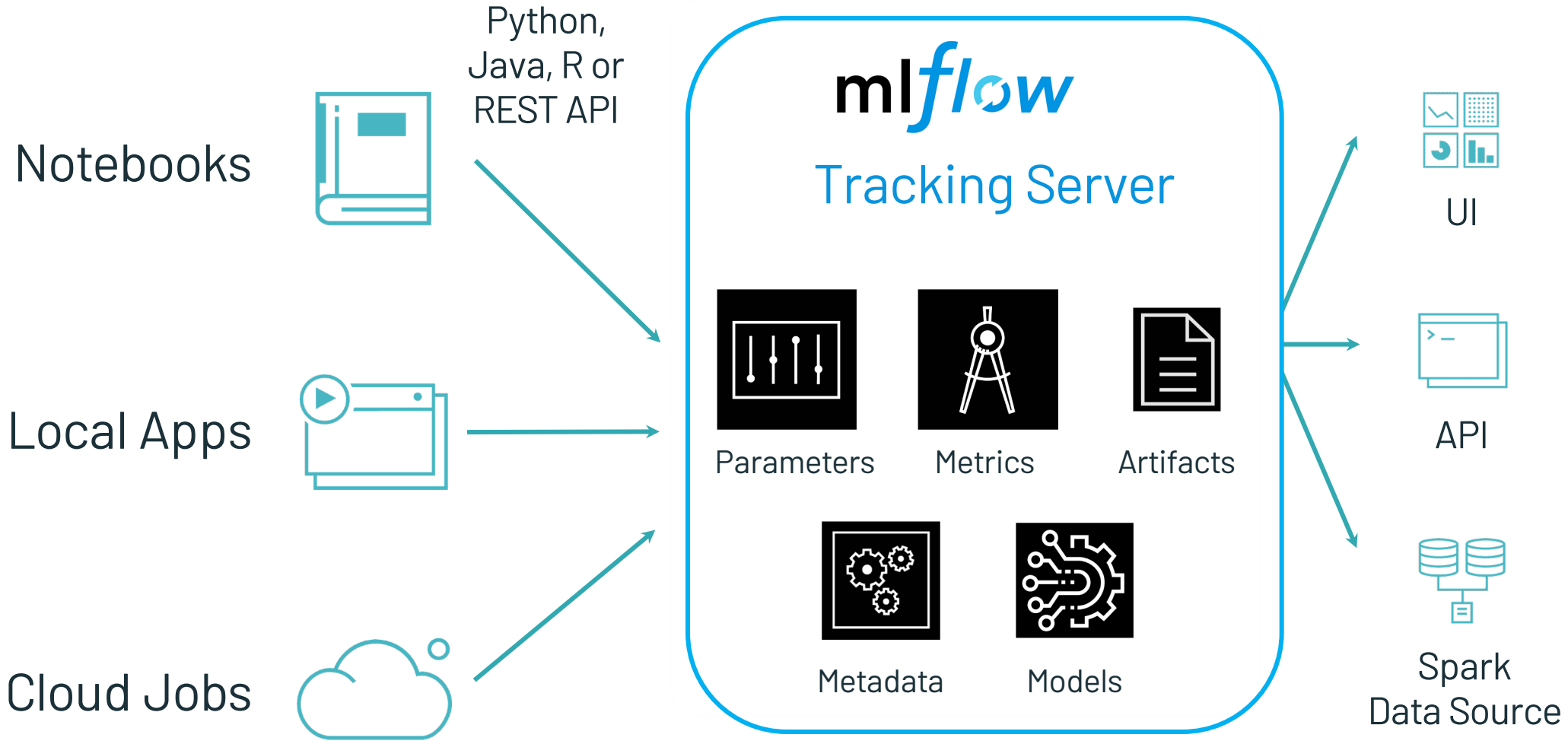
PyTorch · scikit-learn

```
$ mlflow ui
```



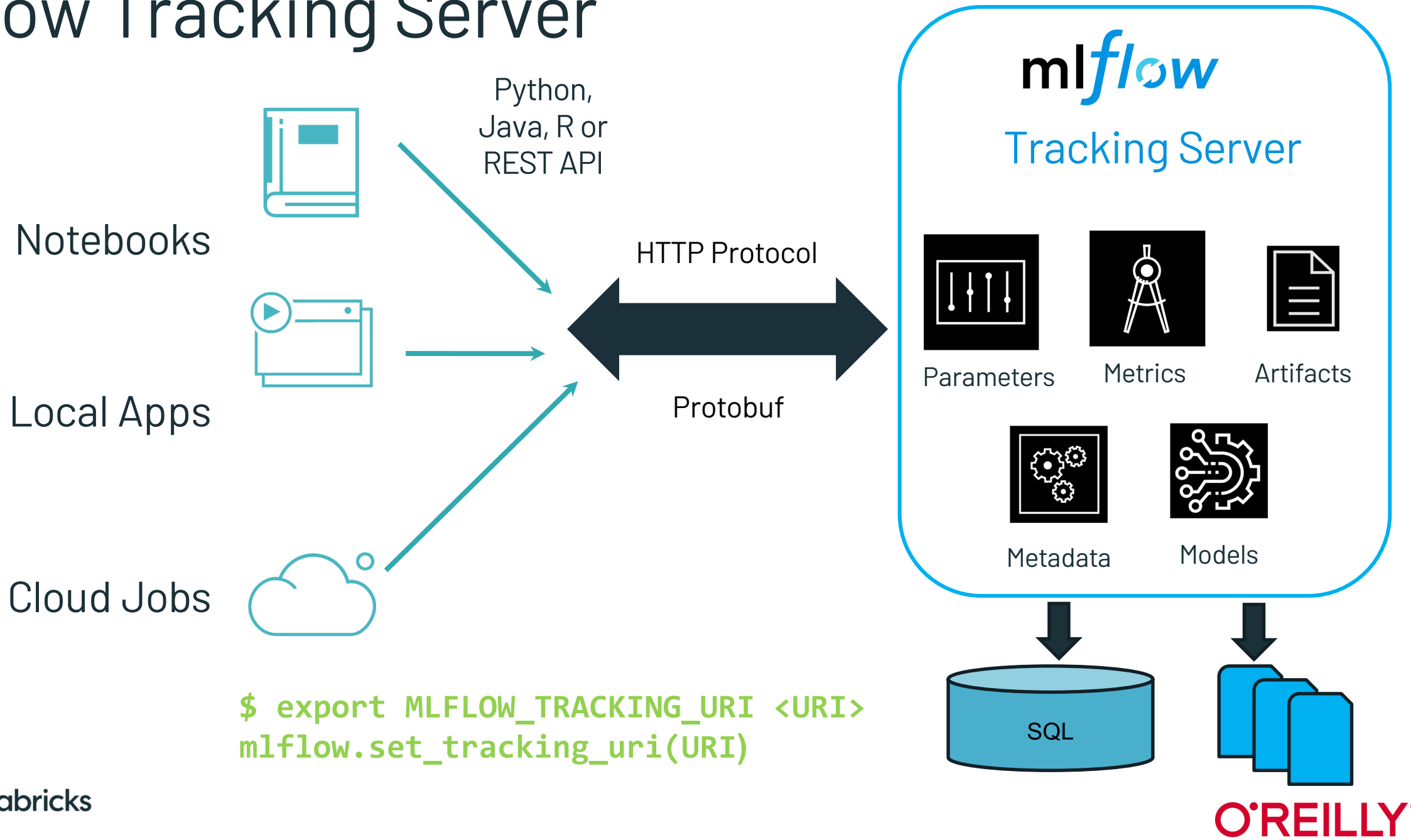
Track parameters, metrics, artifacts, output files & code version

MLflow Tracking Server




```
$ export MLFLOW_TRACKING_URI <URI>
mlflow.set_tracking_uri(URI)
```

MLflow Tracking Server




```
$ export MLFLOW_TRACKING_URI <URI>
mlflow.set_tracking_uri(URI)
```

MLflow Tracking Server Storage

Entity (Metadata) Backend Store

- FileStore (local filesystem)
  - ***mlruns*** directory by default
- SQLStore (via SQLAlchemy)
  - PostgreSQL, MySQL, SQLite
- MLflow Plugins Scheme
  - Customized Entity Metastore
- Managed MLflow on Databricks
  - MySQL on AWS and Azure

Artifact Store

- Local Filesystem
  - ***mlruns*** directory
- S3 backed store
- Azure Blob storage
- Google Cloud Storage
- DBFS artifact repo

Python API

The MLflow Python API is organized into the following modules. The most common functions are exposed in the `mlflow` module, so we recommend starting there.

- mlflow
- mlflow.azureml
- mlflow.deployments
- mlflow.entities
- mlflow.fastai
- mlflow.gluon
- mlflow.h2o
- mlflow.keras
- mlflow.lightgbm
- mlflow.mleap
- mlflow.models
- mlflow.onnx
- mlflow.projects
- mlflow.pyfunc
  - Inference API
  - Filesystem format
    - MLModel configuration
  - Creating custom Pyfunc models
    - Workflows
    - Which workflow is right for my use case?
- mlflow.pytorch
- mlflow.sagemaker
- mlflow.sklearn
- mlflow.spacy
- mlflow.spark
- mlflow.tensorflow
- mlflow.tracking

Built-In Model Flavors

MLflow provides several standard flavors that might be useful in your applications. Specifically, many of its deployment tools support these flavors, so you can export your own model in one of these flavors to benefit from all these tools:

- Python Function (`python_function`)
- R Function (`crate`)
- H_2O (`h2o`)
- Keras (`keras`)
- MLeap (`mleap`)
- PyTorch (`pytorch`)
- Scikit-learn (`sklearn`)
- Spark MLlib (`spark`)
- TensorFlow (`tensorflow`)
- ONNX (`onnx`)
- MXNet Gluon (`gluon`)
- XGBoost (`xgboost`)
- LightGBM (`lightgbm`)

mlflow.sklearn

The `mlflow.sklearn` module provides an API for logging and loading scikit-learn models. This module exports scikit-learn models with the following flavors:

Python (native) pickle format

This is the main flavor that can be loaded back into scikit-learn.

`mlflow.pyfunc`

Produced for use by generic pyfunc-based deployment tools and batch inference.

`mlflow.sklearn.get_default_conda_env`(*include_cloudpickle=False*) [source]

Returns

The default Conda environment for MLflow Models produced by calls to `save_model()` and `log_model()`.

`mlflow.sklearn.load_model`(*model_uri*) [source]

Load a scikit-learn model from a local file or a run.

Parameters

model_uri –

The location, in URI format, of the MLflow model, for example:

- `/Users/me/path/to/local/model`
- `relative/path/to/local/model`

MLFlow API Data Flow: Localhost defaults

- No mlflow server --backend-store-uri --default-root-artifact-uri
- No REST calls to the Tracking Service
- Everything stored under local directory **mlruns**

```
with mlflow.start_run(run_name="DEFAULTS") as run:
    params = {"n_estimators": 3, "random_state": 42}
    sk_learn_rfr = RandomForestRegressor(params)

    # Log params using the MLflow sklearn APIs
     mlflow.log_params(params)
    mlflow.log_param("param_1", randint(0, 100))
    mlflow.log_metric("metric_1", random())
    mlflow.log_metric("metric_2", random() + 1)
    mlflow.sklearn.log_model(sk_learn_rfr,
                             artifact_path="sklearn-model")
```

MLFlow API Data Flow: Localhost w/ SQLAlchemy backend store

- No mlflow server --backend-store-uri --default-root-artifact-uri
- No REST calls to the Tracking Service
- Artifacts stored under local directory **mlruns**
- Mlflow entities stored in **mlruns.db**

```
local_store_uri = "sqlite:///mlruns.db"
mlflow.set_tracking_uri(local_store_uri)
```

```
with mlflow.start_run(run_name="LOCAL_REGISTRY") as run:
    params = {"n_estimators": 3, "random_state": 42}
    sk_learn_rfr = RandomForestRegressor(params)

    # Log params using the MLflow sklearn APIs
    mlflow.log_params(params)
    mlflow.log_param("param_1", randint(0, 100))
    mlflow.log_metric("metric_1", random())
    mlflow.log_metric("metric_2", random() + 1)
    mlflow.sklearn.log_model(sk_learn_rfr,
           artifact_path="sklearn-model")
```

```
mlruns
└── 0
    └── 1f896ddc61d34d089d4126eddceb5b52
        └── artifacts
            └── sklearn-model
                ├── MLmodel
                ├── conda.yaml
                └── model.pkl

4 directories, 3 files
```

```
(tutorials) ➜  mlflow_fluent git:(master) ✗ sqlite3 mlruns.db
SQLite version 3.30.1 2019-10-10 20:19:45
Enter ".help" for usage hints.
sqlite> .tables
alembic_version        metrics                registered_model_tags
experiment_tags        model_version_tags     registered_models
experiments            model_versions         runs
latest_metrics         params                 tags
sqlite> select * from params;
n_estimators|3|29c1e723cfad48bbbc20c4ca52414780
random_state|42|29c1e723cfad48bbbc20c4ca52414780
param_1|10|29c1e723cfad48bbbc20c4ca52414780
sqlite> select * from metrics;
metric_1|0.216184107170305|1597259612673|29c1e723cfad48bbbc20c4ca52414780|0|0
metric_2|1.17106399018329|1597259612697|29c1e723cfad48bbbc20c4ca52414780|0|0
sqlite>
```

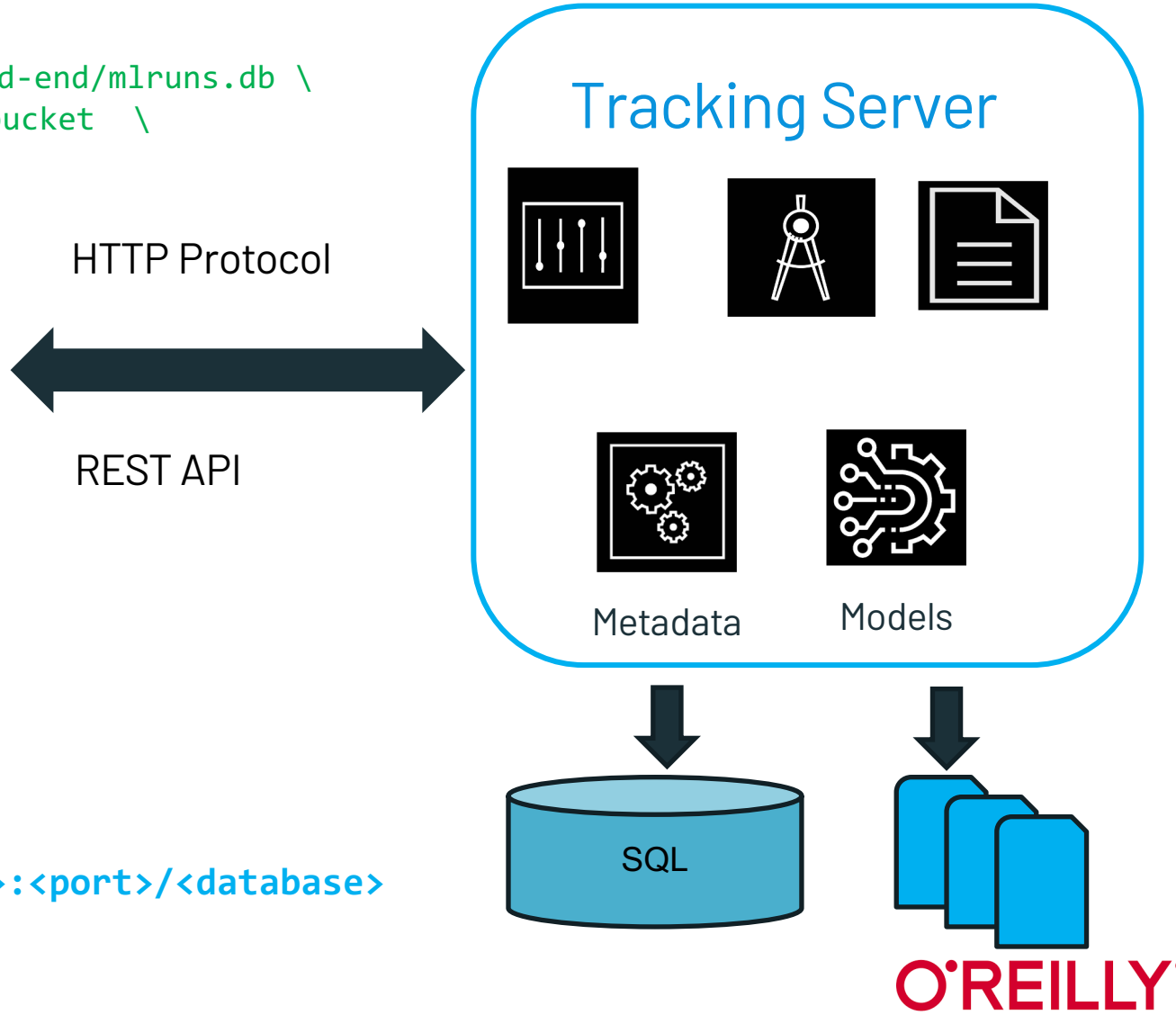
MLFlow API Data Flow: Use Tracking Server SQLAlchemy backend store

```
mlflow server --backend-store-uri sqlite:///mnt/backend-end/mlruns.db \
              --default-artifact-root s3://my-mlflow-bucket  \
              --host host.mydomain.com
```

```
tracking_uri = “https://host.mydomain.com:5000”
mlflow.set_tracking_uri(tracking_uri)
```

```
with mlflow.start_run(run_name="LOCAL_REGISTRY") as run:
    params = {"n_estimators": 3, "random_state": 42}
    sk_learn_rfr = RandomForestRegressor(params)

    # Log params using the MLflow sklearn APIs
    mlflow.log_params(params)
    mlflow.log_param("param_1", randint(0, 100))
    mlflow.log_metric("metric_1", random())
    mlflow.log_metric("metric_2", random() + 1)
    mlflow.sklearn.log_model(sk_learn_rfr,
          artifact_path="sklearn-model")
```



Backend Store URI Format: MySQL, PostgreSQL, SQLite, etc.

```
<dialect>+<driver>://<username>:<password>@<host>:<port>/<database>
```

databricks

O'REILLY®

MLFlow set of APIs

Fluent MLflow APIs

- Python
  - High-level operations for runs and experiments
  - Model Flavor APIs
- Java
  - MLflowContext
  - Experiments, runs, search, etc
- R
  - Experiments, runs, search etc

MLflowClient

- Low Level CRUD interface to experiments, runs, Entities, Model Registry, etc
- `import mflow.tracking`
- `client = MLflowClient(**kwargs)`
- `Metric,Param,Search etc`

MLflow REST API

- Make REST calls to Tracking server with endpoints
- `https://<traciking_server>/api/..`
- `/2.0/mlfow/2.0/runs/log-metric`
- `/2.0/mlflow/runs/log-parameter`
- `/2.0/mlflow/runs/search`

What Did We Talk About? mlflow™

- Modular Components greatly simplify the ML lifecycle
- Easy to install and use & Great Developer Experience
- Develop & Deploy locally and track locally or remotely
- Available APIs: Python, Java & R (Soon Scala)
- Visualize experiments and compare runs

Learning More About MLflow & Get Involved!

- pip install mlflow – to get started
- Find docs & examples at mlflow.org
- Peruse code and contribute at MLflow Github
- Join the Slack channel
- More MLflow tutorials

MLflow Tracking Tutorials

https://github.com/dmatrix/olt-mlflow

Thank you! ☺

Q & A

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