

**mlflow**

# 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/tmfs-workshop>

# 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

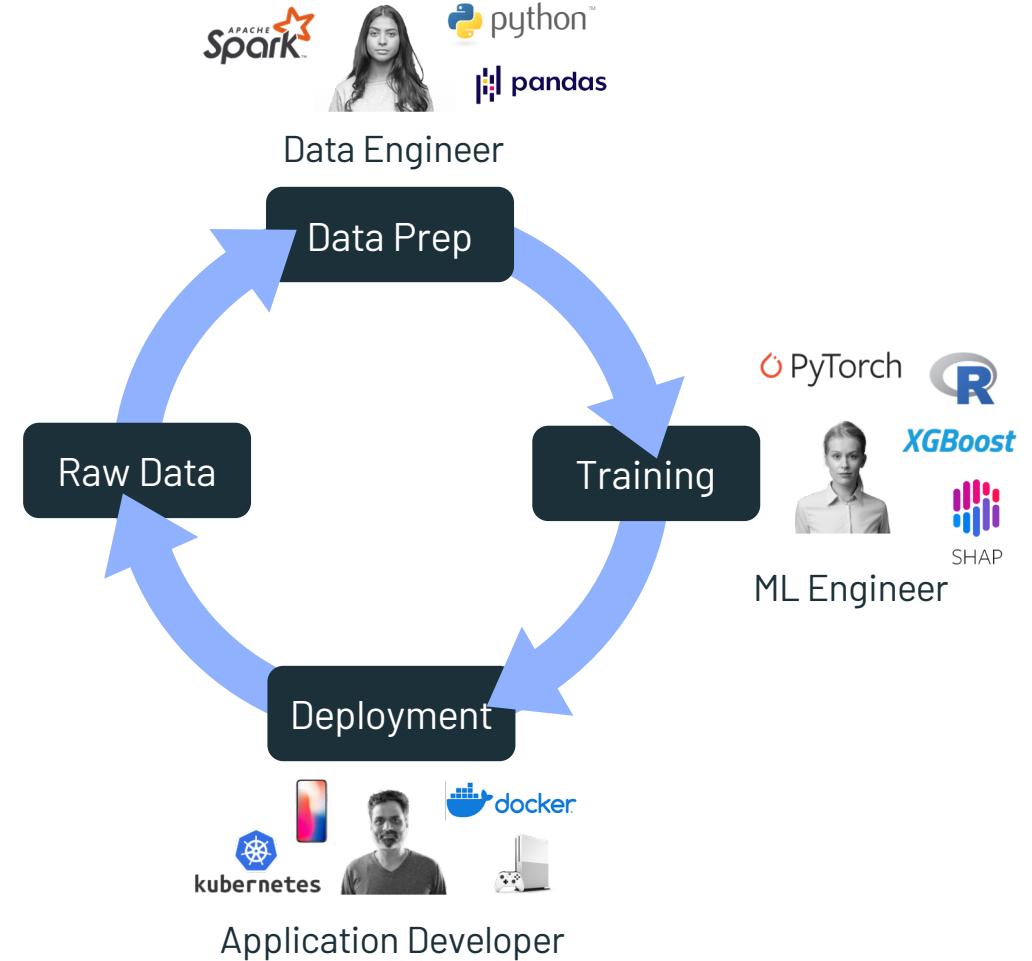
## Machine Learning

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



# But Building ML Applications is Complex

- Continuous, iterative process
- Dependent on data
- Many teams and systems involved



# Solution: ML Platforms

Software platforms to manage the ML application lifecycle, from data to experimentation to production

Examples: Google TFX, Facebook FBLearn, Uber Michelangelo

Can we get these benefits with an open source platform?

# mlflow: An Open Source ML Platform

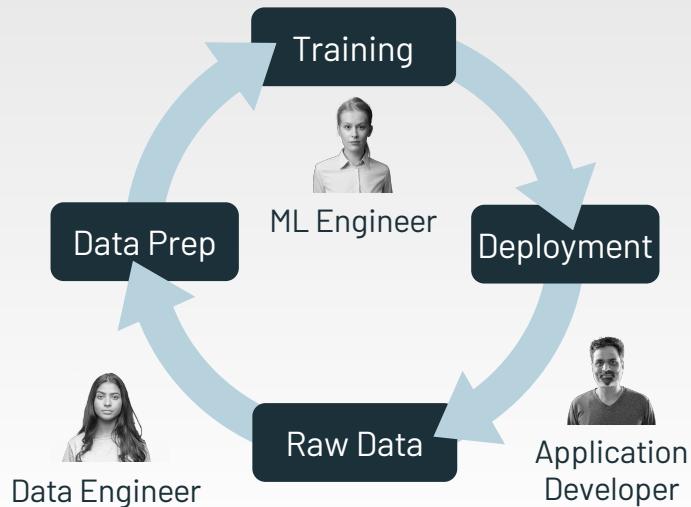
**mlflow**  
**TRACKING**  
Experiment management

**mlflow**  
**PROJECTS**  
Reproducible runs

**mlflow**  
**MODELS**  
Model packaging and deployment

**mlflow**  
**MODEL REGISTRY**  
Model management

Any Language



Any ML Library



# 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"))
```

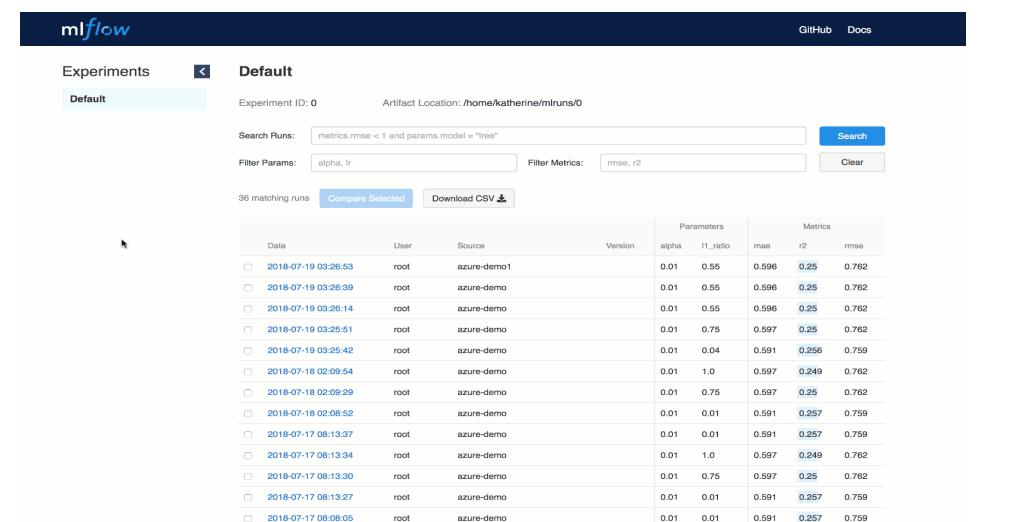
```
For n=2, lr=0.1: accuracy=0.71
For n=2, lr=0.2: accuracy=0.79
For n=2, lr=0.5: accuracy=0.83
For n=2, lr=0.9: accuracy=0.79
For n=3, lr=0.1: accuracy=0.83
For n=3, lr=0.2: accuracy=0.82
For n=4, lr=0.5: accuracy=0.75
...
```

What version of  
my code was this  
result from?

# 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



The screenshot shows the MLflow UI interface. At the top, there's a search bar with the placeholder "Search Runs:" containing the query "metrics.rmse < 1 and params.model = 'tree'". Below the search bar, there are filter parameters: "alpha, lr" and "Filter Metrics: rmse, r2". A "Compare Selected" button and a "Download CSV" button are also present. The main area displays a table of 36 matching runs. The columns in the table are Date, User, Source, Version, Parameters, and Metrics. The Parameters column includes alpha, lr\_ratio, rmse, and r2. The Metrics column includes rmse.

Date	User	Source	Version	Parameters	Metrics
2018-07-19 03:26:53	root	azure-demo1		alpha: 0.01, lr_ratio: 0.55	rmse: 0.596, r2: 0.25
2018-07-19 03:26:39	root	azure-demo		alpha: 0.01, lr_ratio: 0.55	rmse: 0.596, r2: 0.25
2018-07-19 03:26:14	root	azure-demo		alpha: 0.01, lr_ratio: 0.55	rmse: 0.596, r2: 0.25
2018-07-19 03:25:51	root	azure-demo		alpha: 0.01, lr_ratio: 0.75	rmse: 0.597, r2: 0.25
2018-07-19 03:25:42	root	azure-demo		alpha: 0.01, lr_ratio: 0.04	rmse: 0.591, r2: 0.256
2018-07-18 02:09:54	root	azure-demo		alpha: 0.01, lr_ratio: 1.0	rmse: 0.597, r2: 0.249
2018-07-18 02:09:29	root	azure-demo		alpha: 0.01, lr_ratio: 0.75	rmse: 0.597, r2: 0.25
2018-07-18 02:08:52	root	azure-demo		alpha: 0.01, lr_ratio: 0.01	rmse: 0.591, r2: 0.257
2018-07-17 08:13:37	root	azure-demo		alpha: 0.01, lr_ratio: 0.01	rmse: 0.591, r2: 0.257
2018-07-17 08:13:34	root	azure-demo		alpha: 0.01, lr_ratio: 1.0	rmse: 0.597, r2: 0.249
2018-07-17 08:13:30	root	azure-demo		alpha: 0.01, lr_ratio: 0.75	rmse: 0.597, r2: 0.25
2018-07-17 08:13:27	root	azure-demo		alpha: 0.01, lr_ratio: 0.01	rmse: 0.591, r2: 0.257
2018-07-17 08:08:05	root	azure-demo		alpha: 0.01, lr_ratio: 0.01	rmse: 0.591, r2: 0.257

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

# mlflow Tracking for ML Experiments

Easily track parameters, metrics, and artifacts in popular ML libraries

```
with mlflow.start_run(run_name='keras'):
    # log model and datasource
    mlflow.keras.autolog()
```

Library integrations:



PyTorch



Keras



RAPIDS



XGBoost

LightGBM

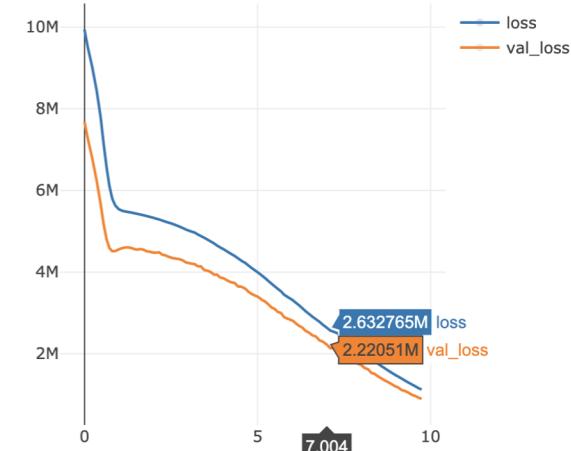
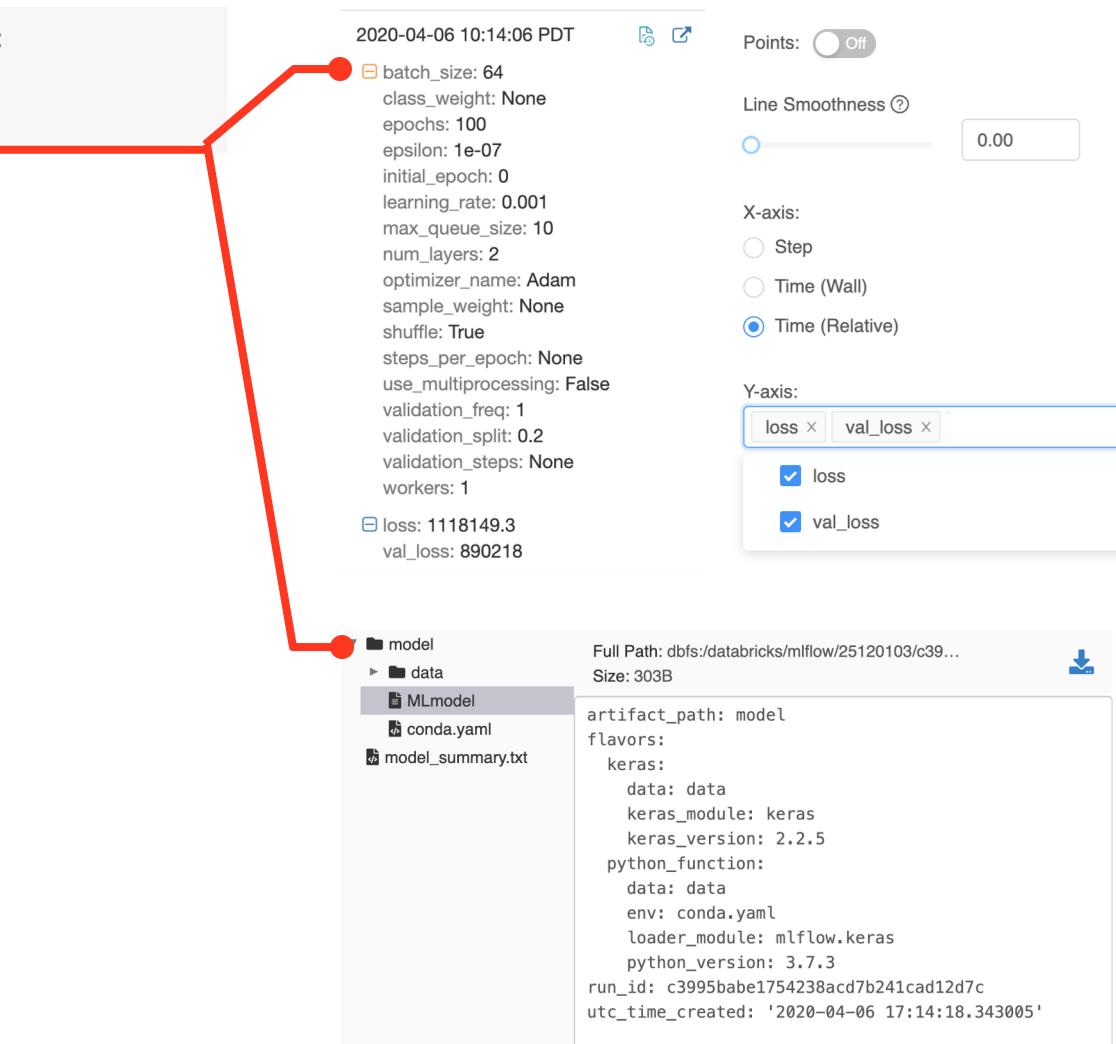
spaCy

fast.ai

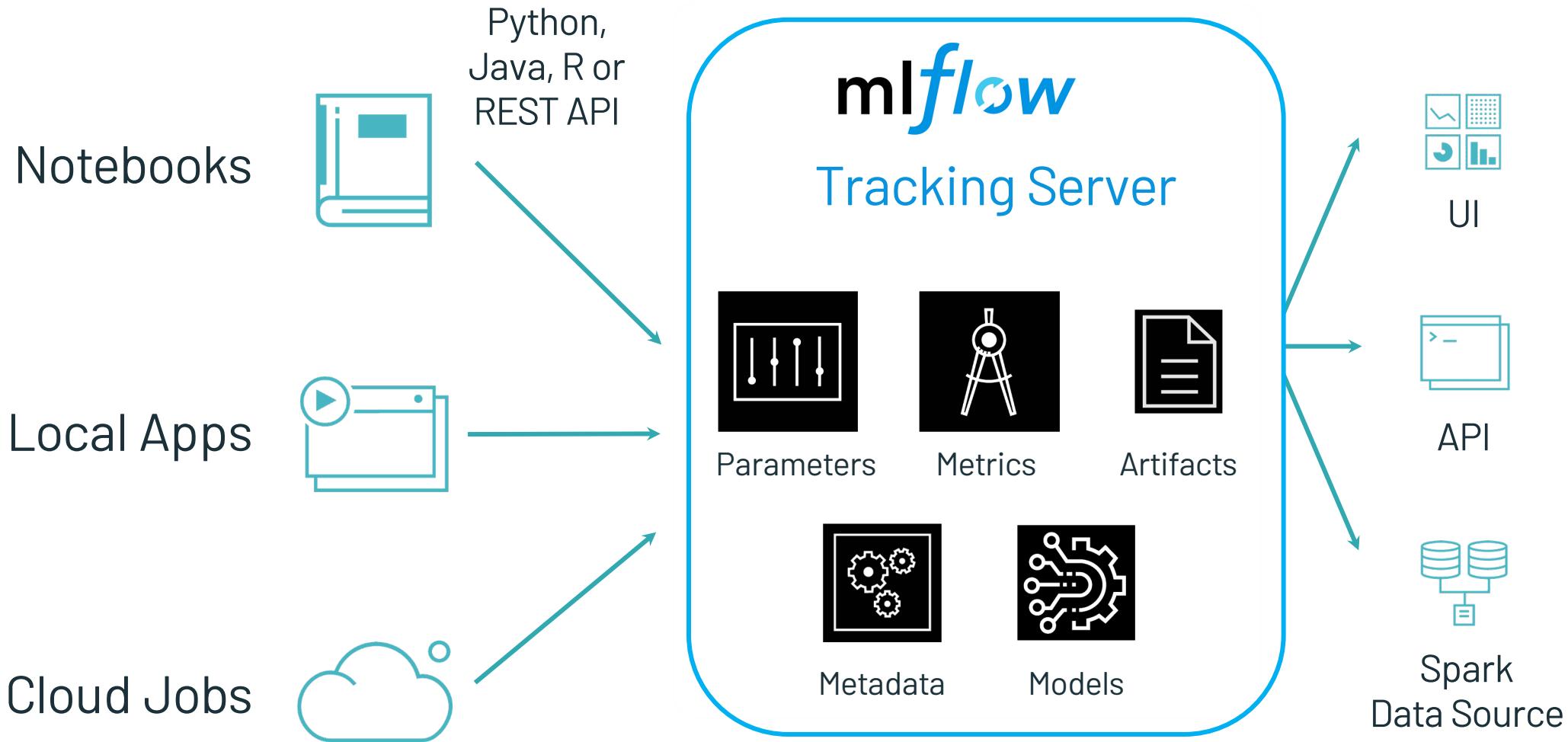
PYCARET



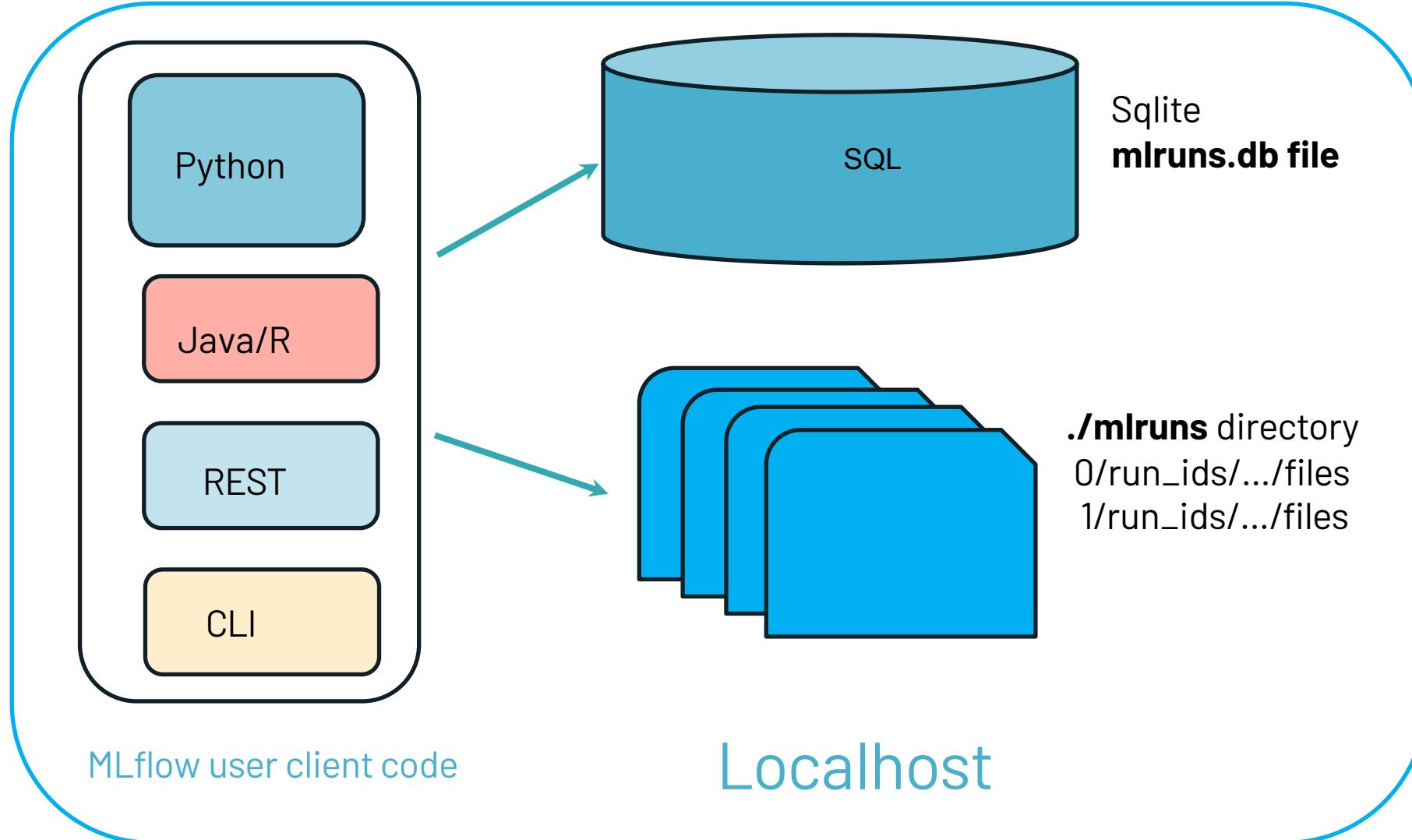
OPTUNA



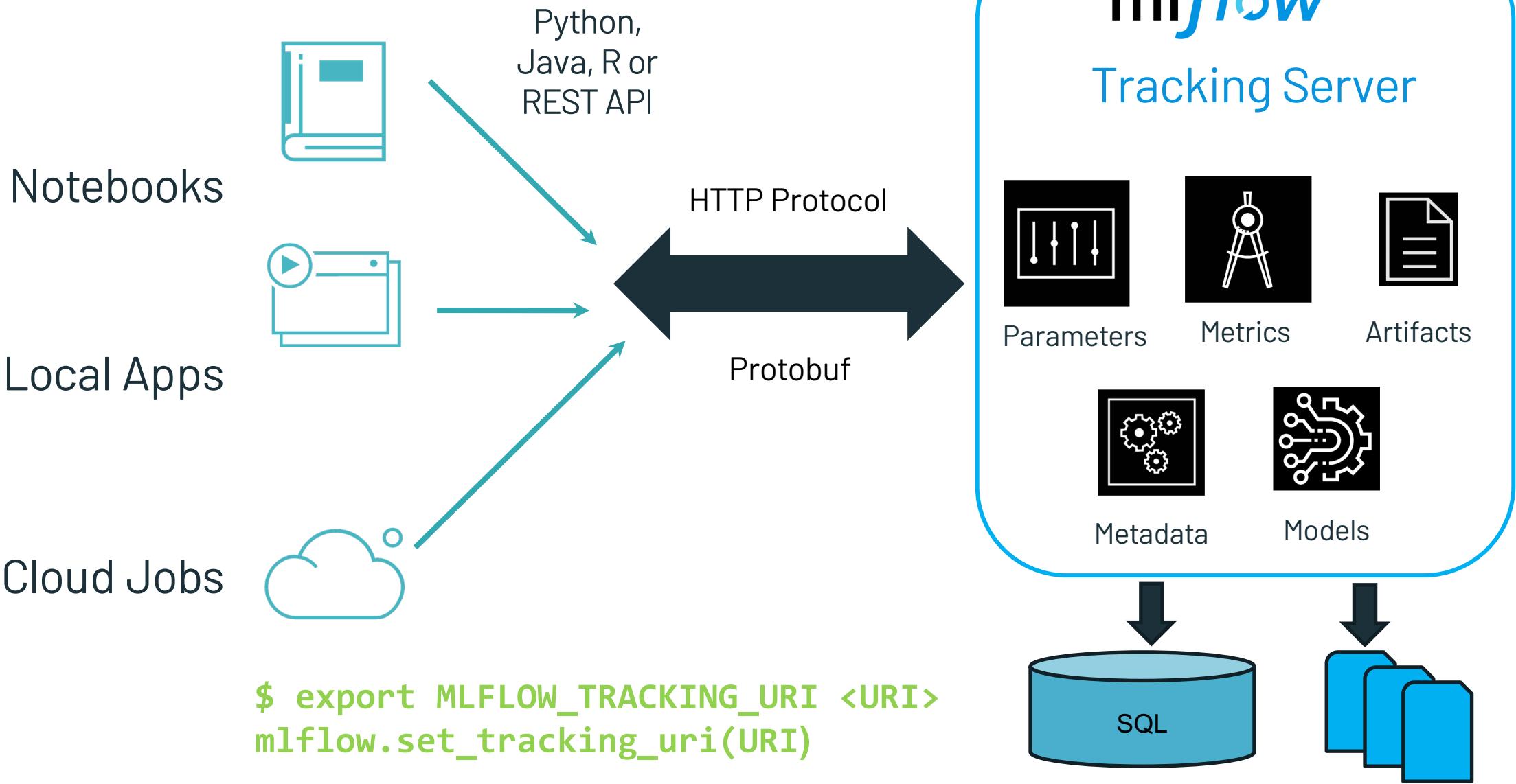
# MLflow Tracking Server



# MLflow Tracking Local Storage

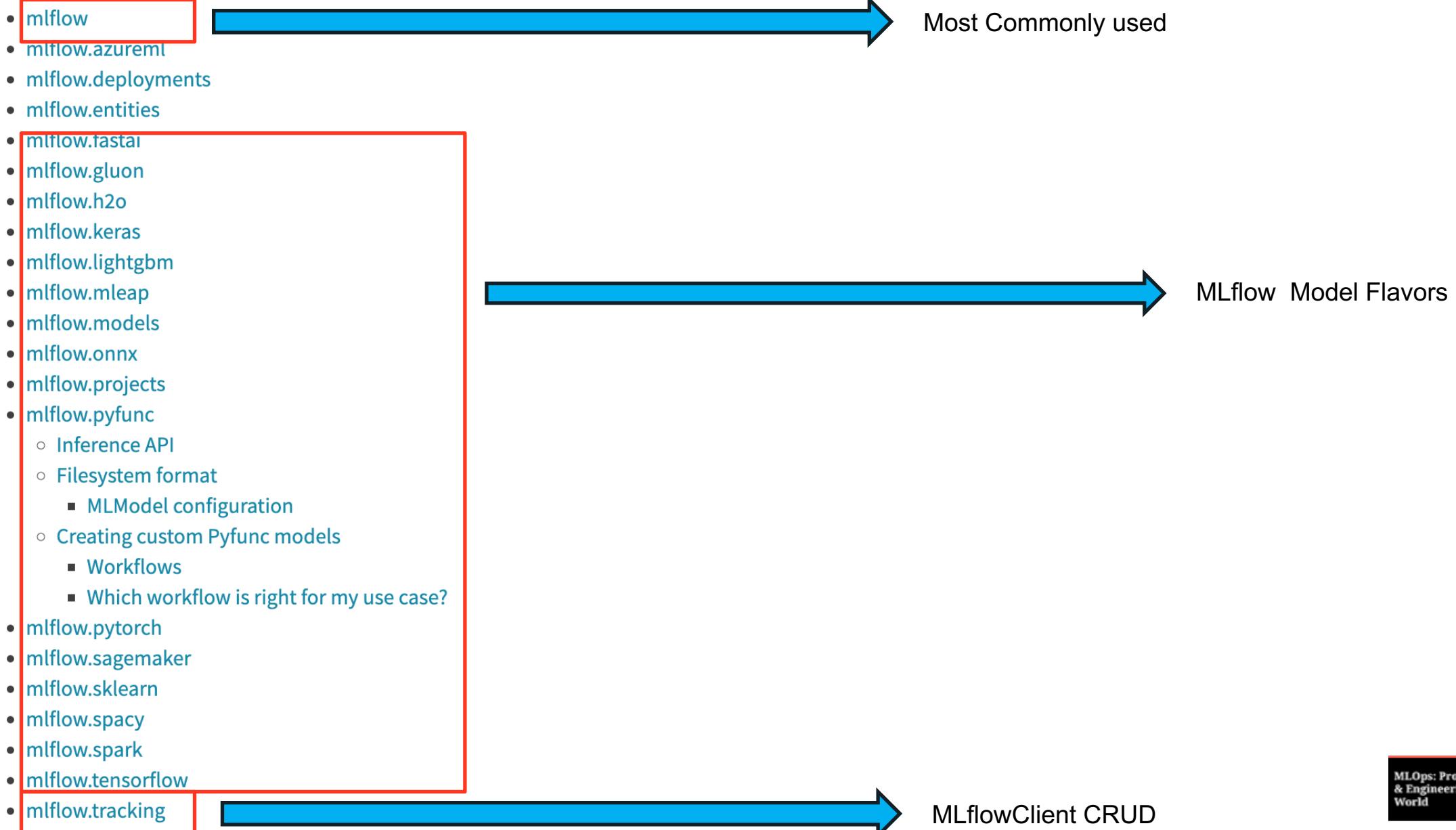


# MLflow Tracking Server



# Python API

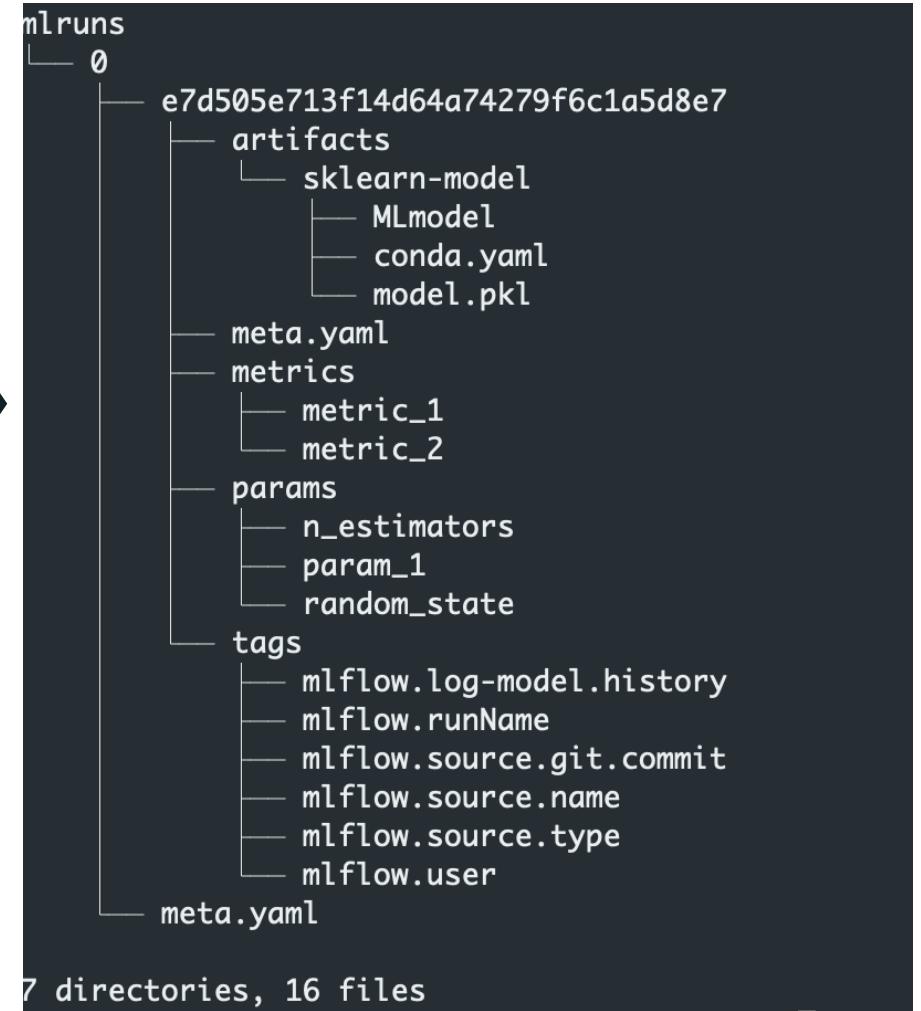
The MLflow Python API is organized into the following modules. The most common functions are exposed in the `mlflow` module, so we recommend starting there.



# MLflow 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")
```



```
mlruns  
└── 0  
    └── e7d505e713f14d64a74279f6c1a5d8e7  
        ├── artifacts  
        │   └── sklearn-model  
        │       ├── MLmodel  
        │       └── conda.yaml  
        │       └── model.pkl  
        ├── meta.yaml  
        ├── metrics  
        │   ├── metric_1  
        │   └── metric_2  
        ├── params  
        │   ├── n_estimators  
        │   └── param_1  
        └── random_state  
        └── tags  
            ├── mlflow.log-model.history  
            ├── mlflow.runName  
            ├── mlflow.source.git.commit  
            ├── mlflow.source.name  
            ├── mlflow.source.type  
            └── mlflow.user  
        └── meta.yaml  
7 directories, 16 files
```



# 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> |
```

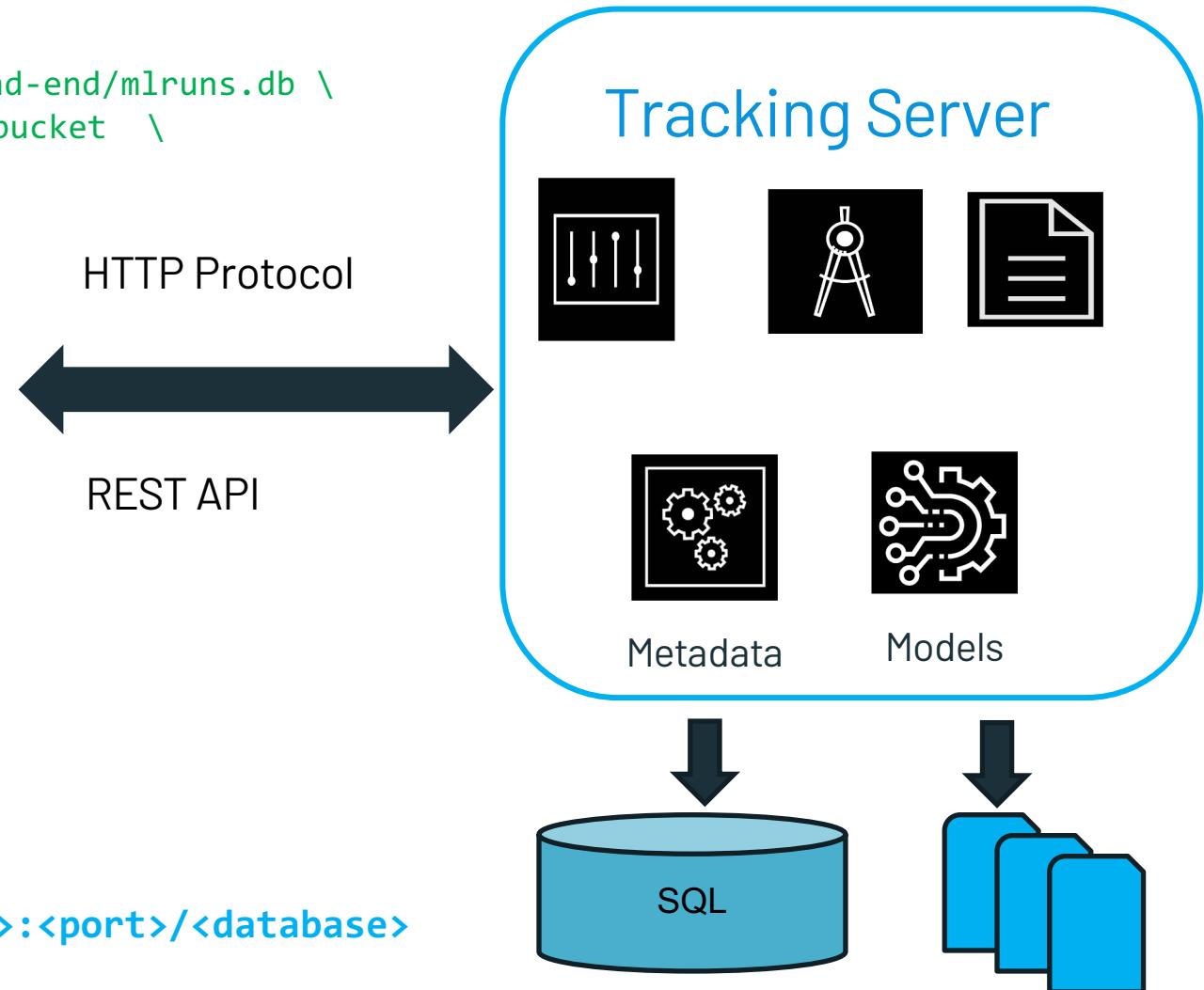
# MLflow API Data Flow: Use Tracking Server SQLAlchemy backend store

```
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>



# 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 mlflow.tracking`
- `client = MLflowClient(**kwargs)`
- Metric, Param, Search etc

## MLflow REST API

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

# mlflow Tracking for Model Schemas

Record what fields are consumed & produced by the model to prevent data mismatches

in MLflow 1.11

```
with mlflow.start_run(run_name='keras'):
    # log model and datasource
    mlflow.keras.autolog()
    mlflow.spark.autolog()

    sig = infer_signature(X_train, y_train)
```

Schema	
Name	Type
Inputs (7)	
user_purchases_7d	long
user_purchases_14d	long
user_location	string
user_last_login	string
item_inventory	long
Outputs (1)	
prediction	double



# mlflow Tracking for Interpretability

SHAP library feature importances and visualizations

in MLflow 1.12

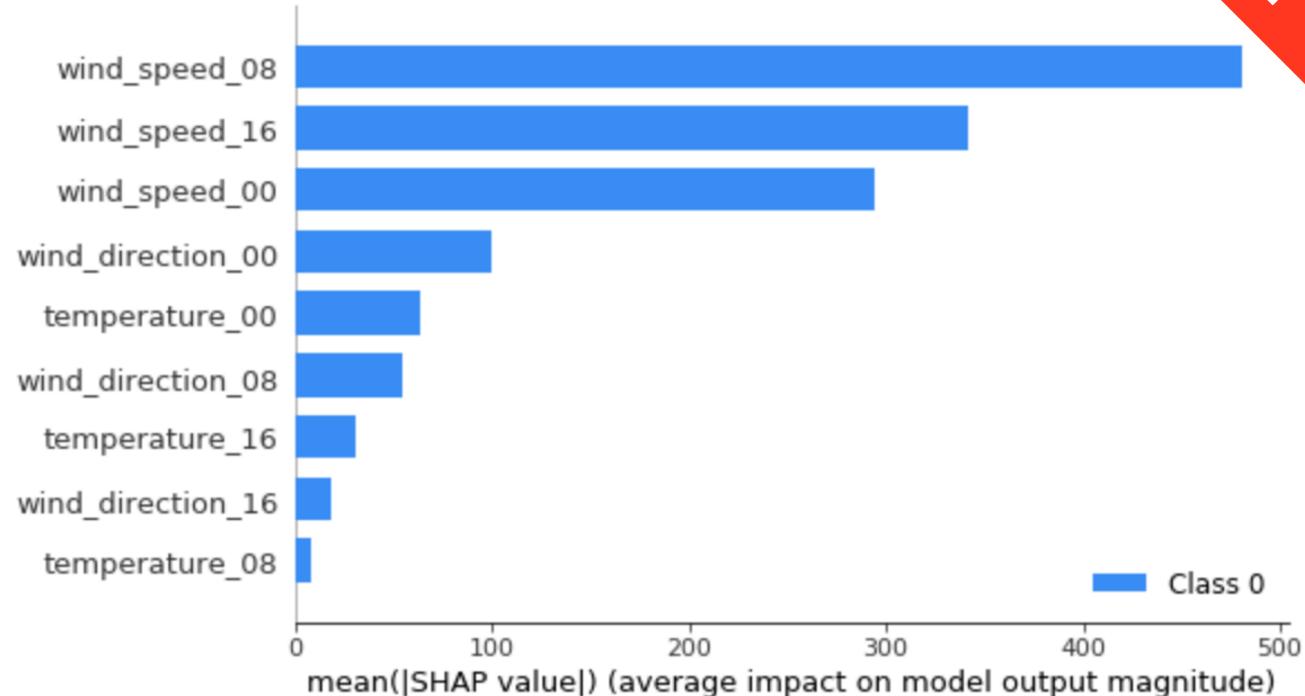
```
with mlflow.start_run(run_name='keras'):
    # log model and datasource
    mlflow.keras.autolog()
    mlflow.spark.autolog()

    sig = infer_signature(X_train, y_train)

    mlflow.shap.log_explanation(model, X_train[:100])
```



SHAP



Easily log and query all aspects of your ML development process

# mlflow + PyTorch

in MLflow 1.12

Collaboration between Facebook and Databricks to bring ML platform features to PyTorch

MLflow autologging for PyTorch Lightning

TorchScript support for faster packaged models

Model deployment to TorchServe

MLflow 1.12 Features Extended PyTorch Integration

And model explainability, autologging, and other enhancements



by Jules Damji, Siddharth Murching and Harutaka Kawamura  
Posted in ENGINEERING BLOG | November 13, 2020

# mlflow + PyTorch

```
# Enable MLflow autologging
import mlflow.pytorch_lightning
mlflow.pytorch_lightning.autolog()

# Train a model with PyTorch Lightning
model = MyModel()
trainer = pl.Trainer()
trainer.fit(model, train, val)
```

```
# Deploy to TorchServe
mlflow deployments create -t torchserve -m models:/mymodel/production -n mymodel
```

mnist\_autolog\_log\_niter > Run b288b477bf3f4ab18c52a8ef26e4c118 ▾

Date: 2020-07-27 11:02:09

Source: \_\_main\_\_.py

User: ec2-user

Duration: 2.3min

Status: FINISHED

▼ Notes ↗

None

▼ Parameters

Name	Value
epsilon	1e-08
learning_rate	0.001
optimizer_name	Adam

▼ Metrics

Name	Value
avg_test_acc ↗	0.968
avg_train_loss ↗	0.029
avg_val_loss ↗	0.113
loss ↗	0.003

▼ Tags

Name	Value	Actions

# What Did We Talk About?

- 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](https://mlflow.org)
- Peruse code and contribute at [MLflow Github](https://github.com/mlflow/mlflow)
- Join the [Slack channel](#)
- More [MLflow tutorials](#)

# Thank you! 😊

## Q & A

[jules@databricks.com](mailto:jules@databricks.com)  
@2twitme

<https://www.linkedin.com/in/dmatrix/>

