mlflow

Platform for Machine Learning Lifecycle

Jules S. Damji
@2twitme

databricks

GitHub and Slides URL

GitHub: https://github.com/dmatrix/ds4g-workshop

Slides: Under the slides directory

About Me

Jules S. Damji
Senior Developer Advocate @ Databricks

Joined Databricks in 2016
20+ Developer at software companies:
Sun, Netscape, VeriSign, @Home, Scalix, LoudCloud/Opsware, Hortonworks etc

databricks

Unified data analytics platform for data science, data engineering, and business analytics to solve tough data problems

Global company with 5,000 customers and 450+ partners

Original creators of popular data and machine learning open source projects

Machine Learning Development is Complex

Traditional Software vs. Machine Learning

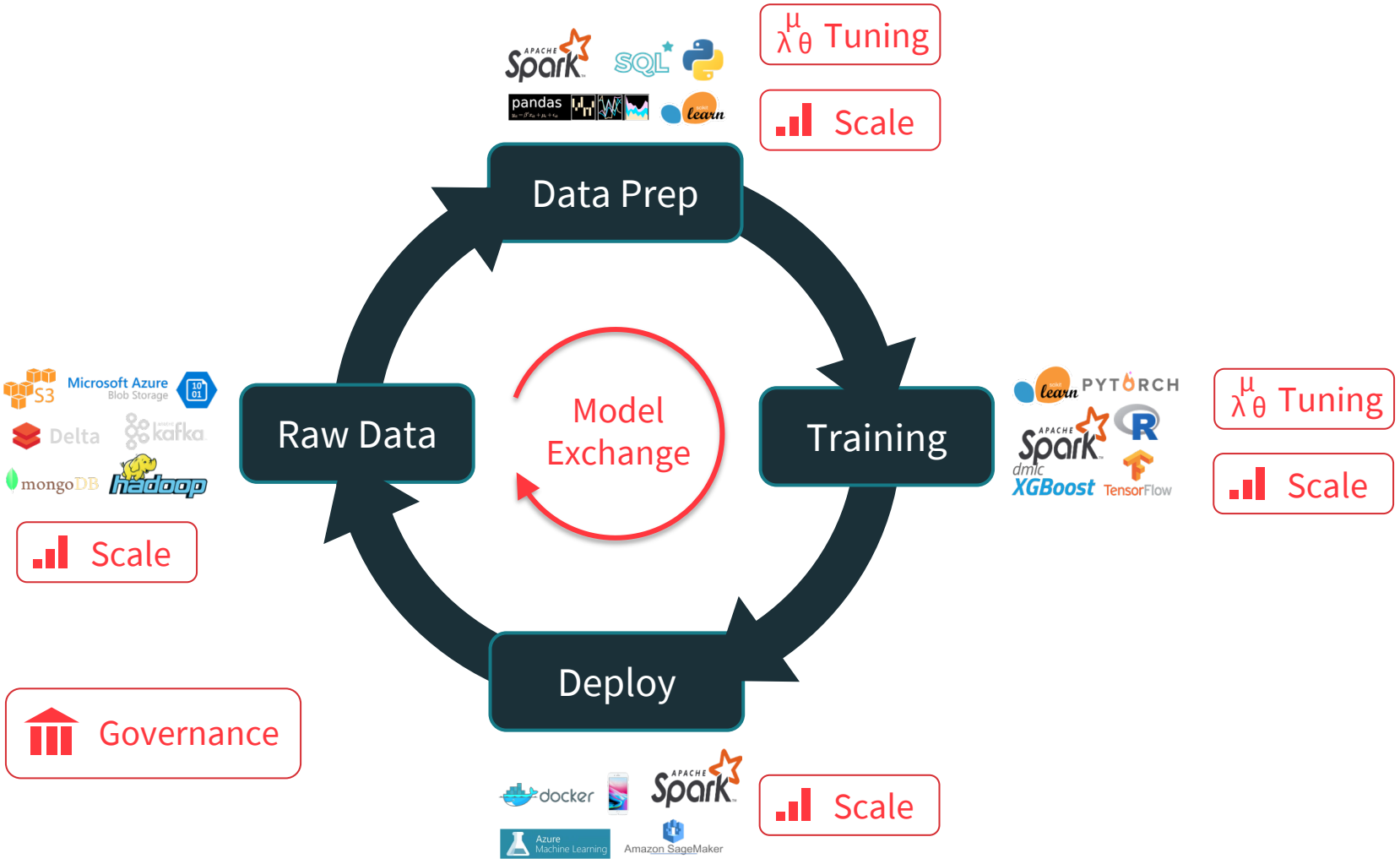
Traditional Software

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

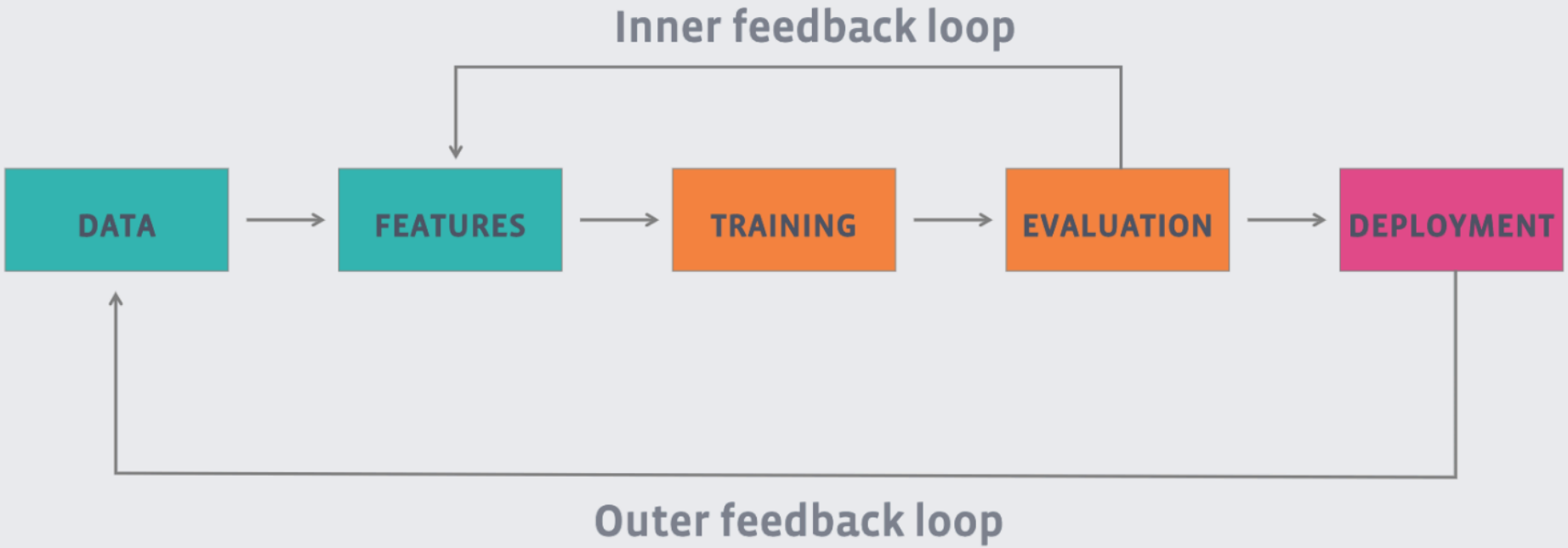
Machine Learning

- 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

Machine Learning Lifecycle



ML Pipeline Lifecycle



Source: https://atscaleconference.com/videos/scale-2018-applied-machine-learning-at-facebook-an-infrastructure-perspective/

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

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?

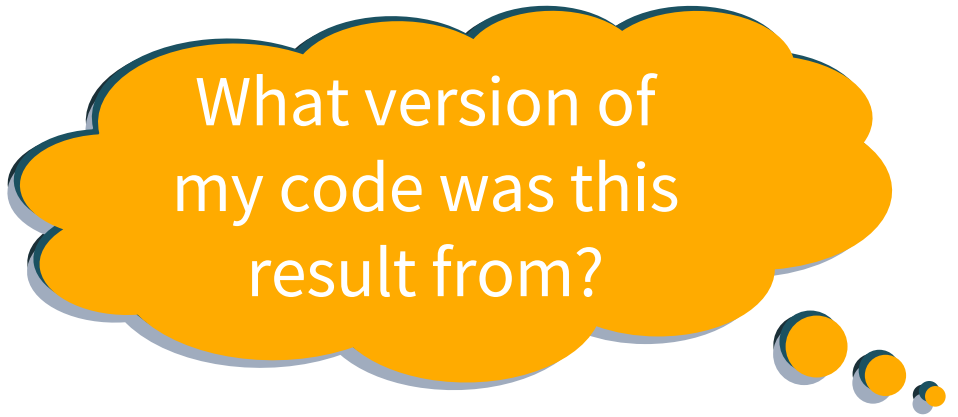
Run: an instance of code that runs by MLflow

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"))
```

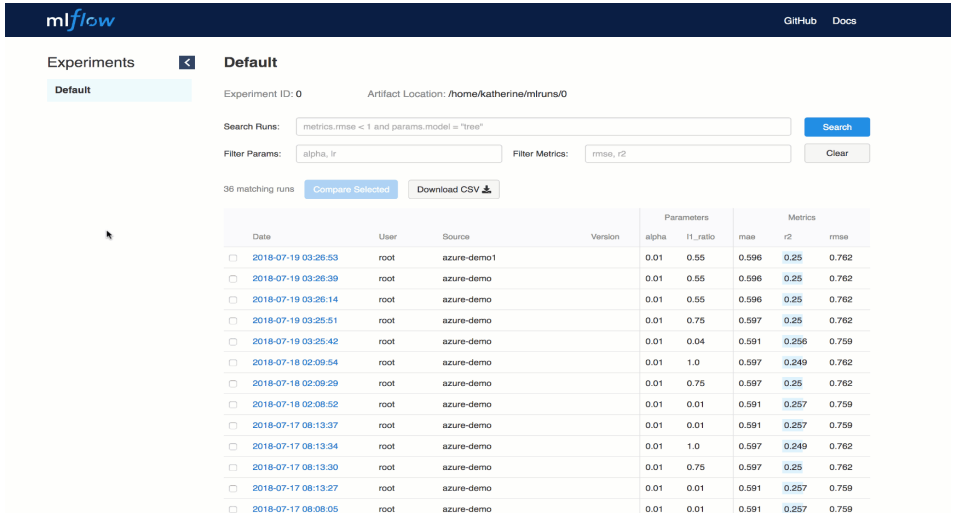
```
For n=2, lr=0.1: accuracy=0.71
For n=2, lr=0.2: accuracy=0.79
For n=2, lr=0.5: accuracy=0.83
For n=2, lr=0.9: accuracy=0.79
For n=3, lr=0.1: accuracy=0.83
For n=3, lr=0.2: accuracy=0.82
For n=4, lr=0.5: accuracy=0.75
...
```



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
```



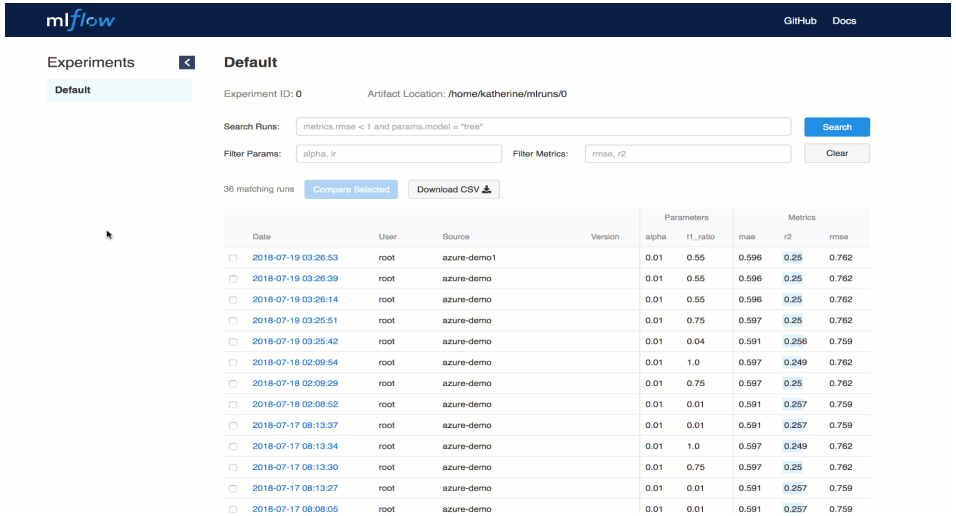
Track parameters, metrics, artifacts, output files & code version

databricks

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

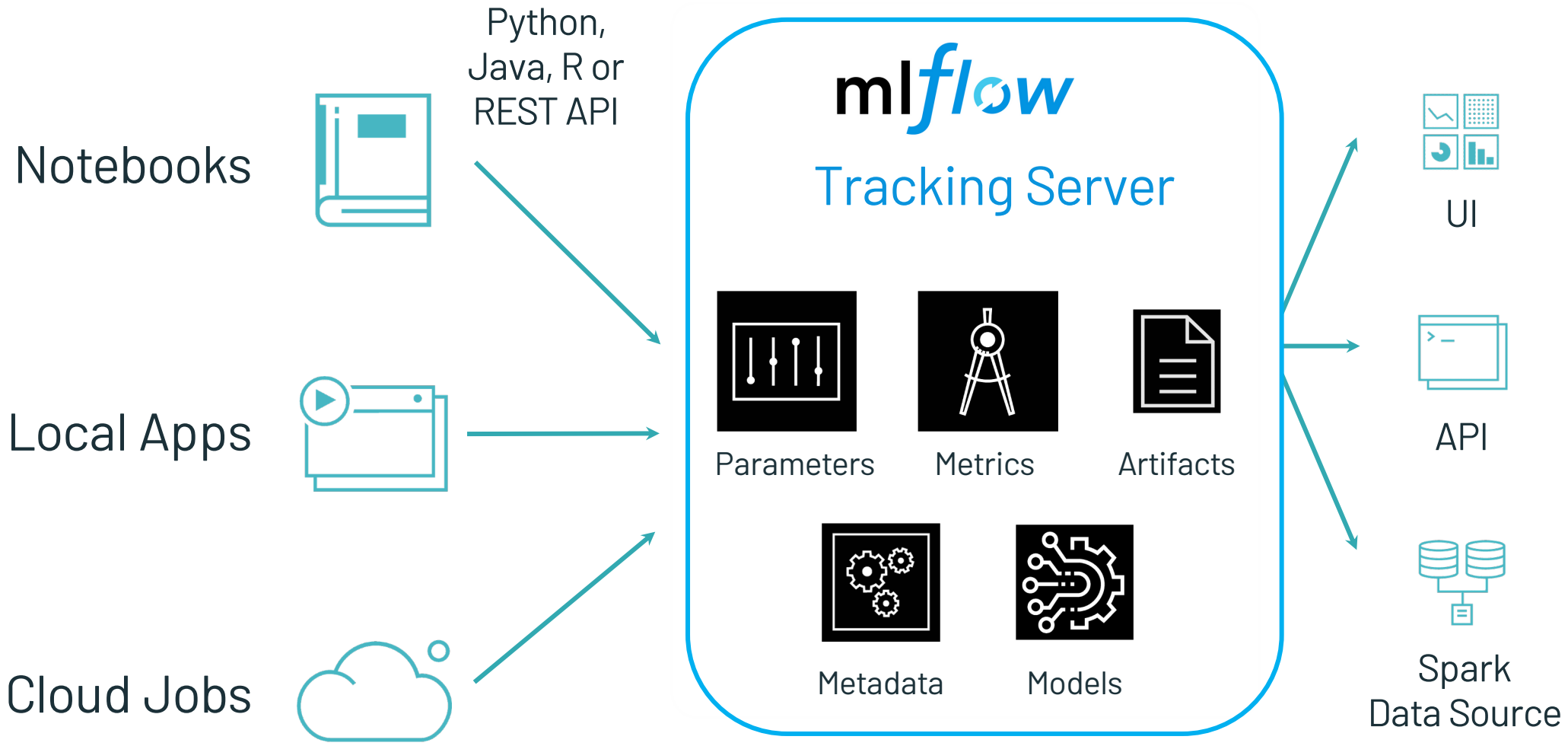
```
import mlflow
import mlflow.keras

# 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
```



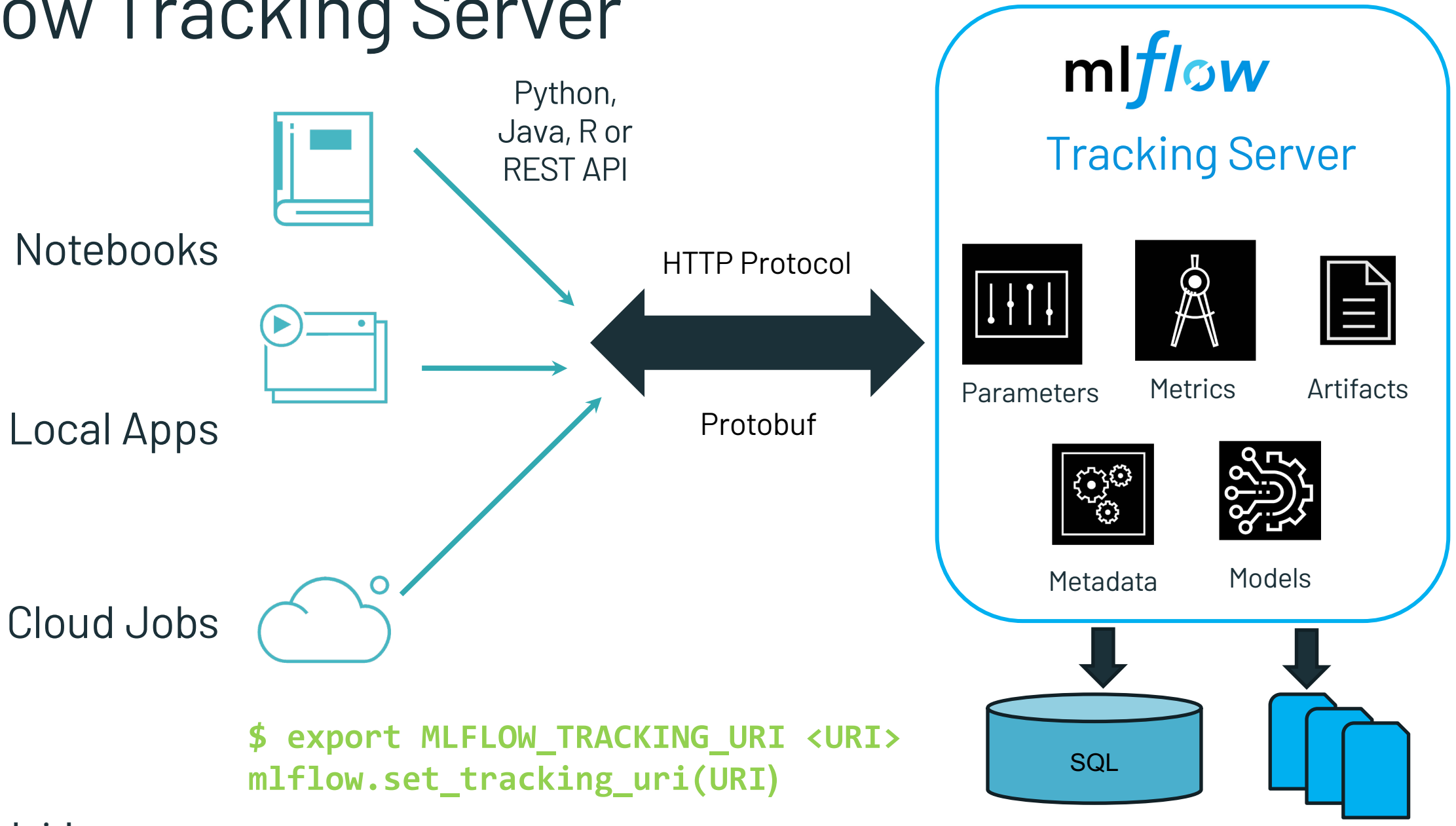
Track parameters, metrics, artifacts, output files & code version

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)
```

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 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`

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

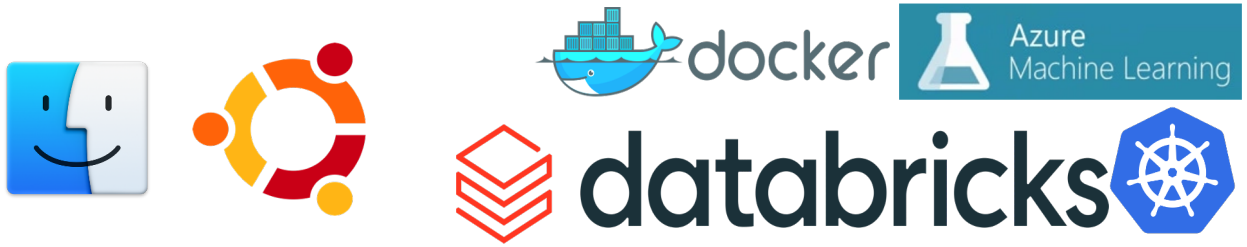
twitter.com/MLflow

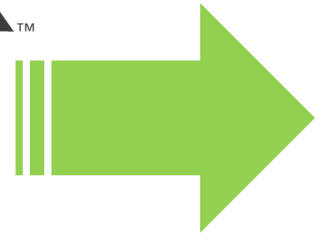
databricks

MLflow Projects Motivation

Diverse set of tools

Diverse set of environments





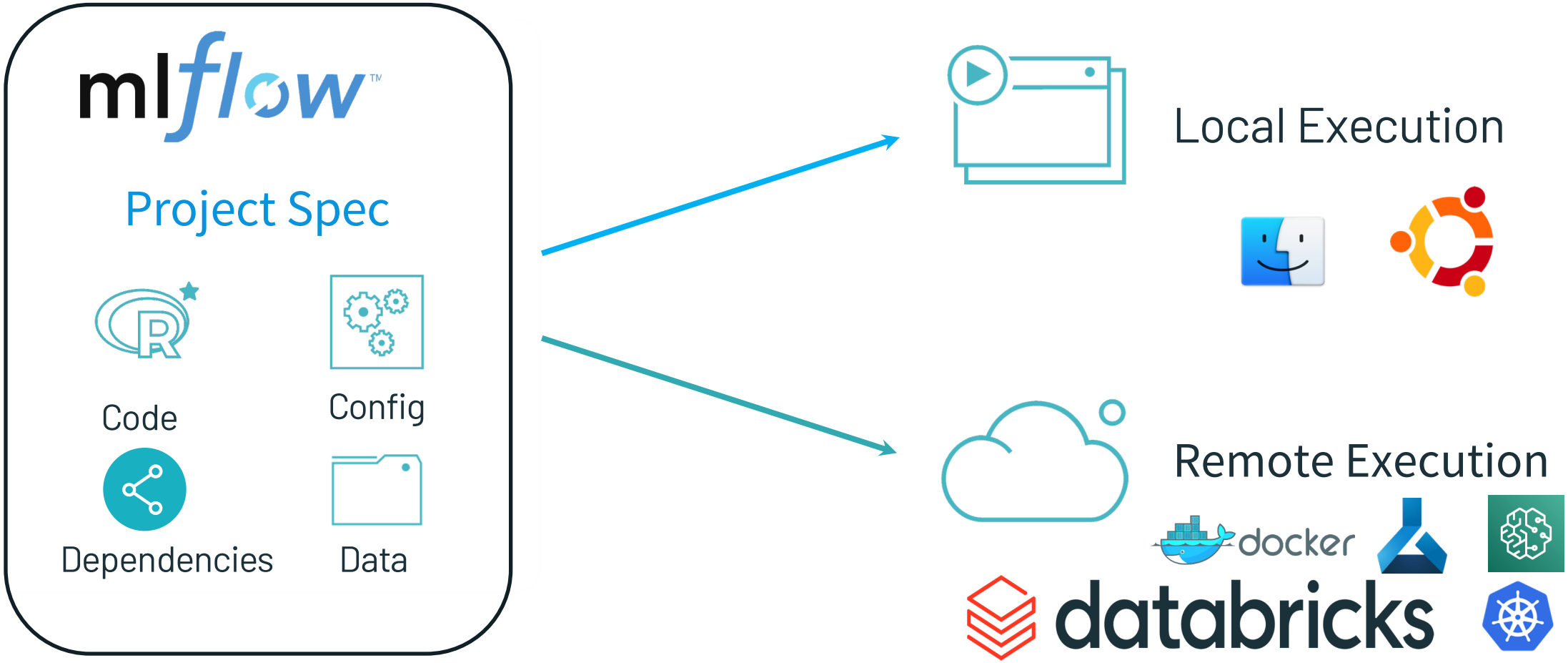
mlflow™

Projects

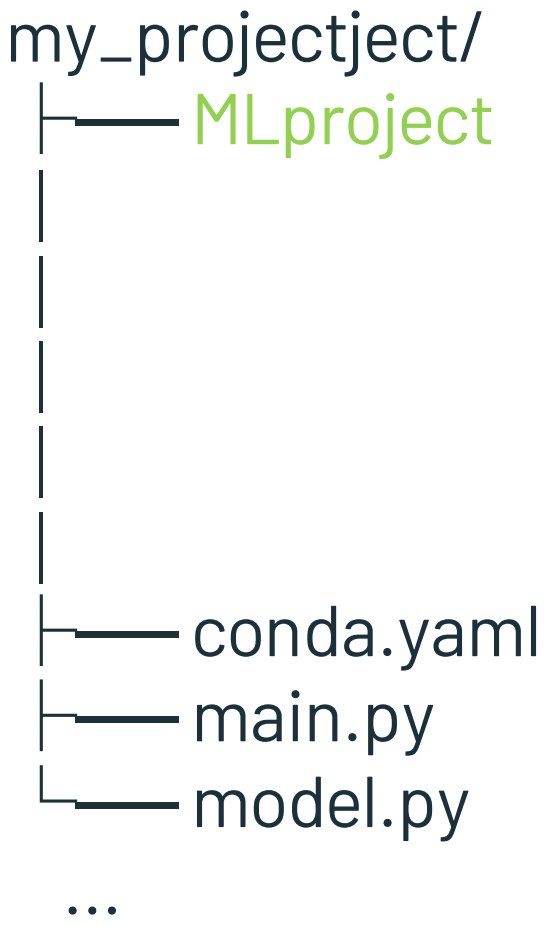
Package data science code in a format that enables reproducible runs on any platform

Challenge: ML results difficult to reproduce

MLflow Projects



1. Example MLflow Project File




```
name: tutorial

conda_env: conda.yaml

entry_points:
  main:
    parameters:
      batch_size: {type: int, default: 10}
      epochs: {type: int, default: 100}
    command: "python train_keras.py {batch_size} {epochs}
```

```
$ mlflow run git://<my_project>.git -P epochs=200

mlflow.run("git://<my_project>", parameters={..})

mlflow run . -e main -P epochs=200
```

2. Example Conda.yaml

```
my_project/
├── MLproject
│
│
│
│
│
│
├── conda.yaml
├── main.py
└── model.py

....
```

```
channels:
- defaults
- conda-forge
dependencies:
- python=3.7.5
- pip
- pip:
  - mlflow
  - keras==2.3.1
  - tensorflow==2.0.0
name: mlflow-env
```

MLflow Projects

Packaging format for reproducible ML runs

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

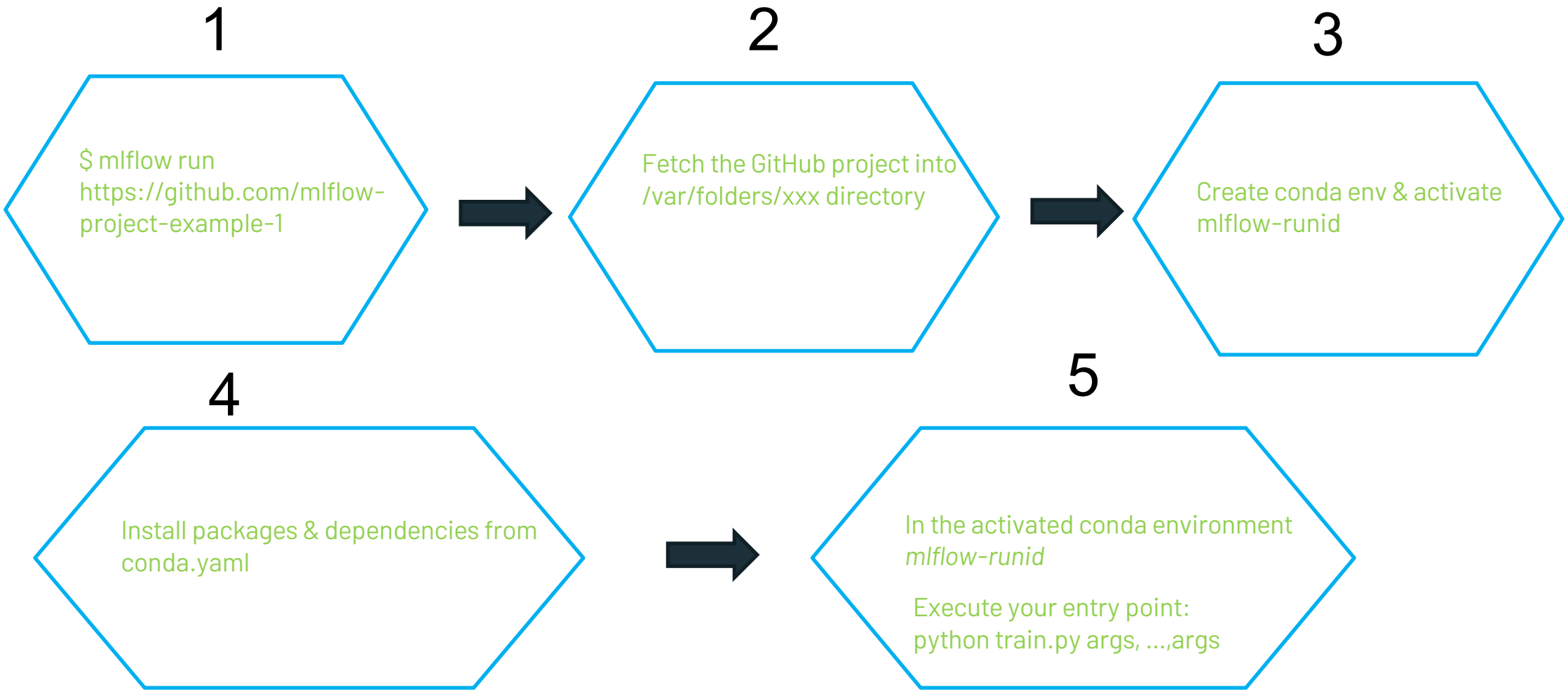
Defines dependencies for reproducibility

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

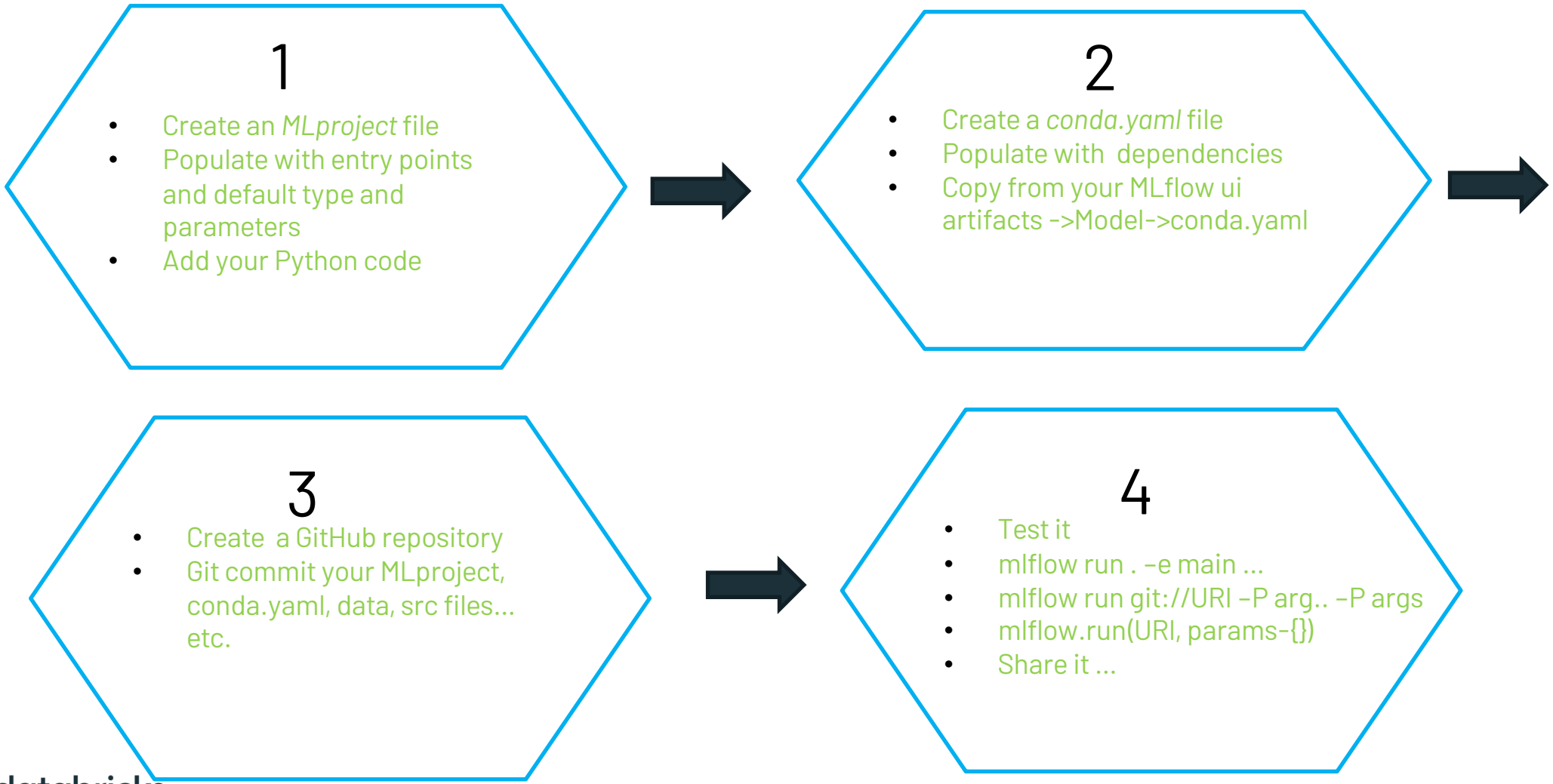
Execution API for running projects

- CLI / Python / R / Java
- Supports local and remote execution
  - mlflow run –help (CLI)
  - mlflow run https://github.com/dmatrix/jsd-mlflow-examples.git#keras/imdbclassifier (CLI)
  - mlflow.run (<project_uri>, parameters={}) or mlflow.projects.run((<project_uri>, parameters={}) (API)

Anatomy of MLflow Project Execution



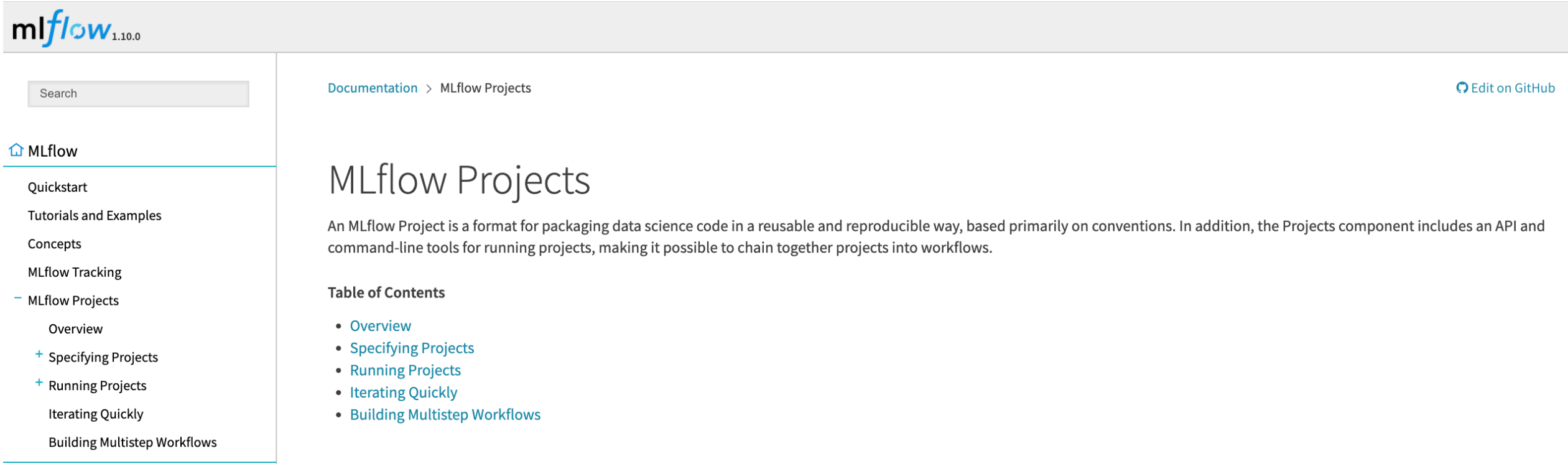
How to build an MLflow Project



MLflow Projects Usage

- Reproducibility and Experimentation ✓
  - Reproduce experiments in a target environment
  - Experiment with different hyperparameters
  - Use of CLI and Programmatic interface
- Shareability ✓
  - Use other people's models and experiment via a MLFlow Project GitHub
- Flexibility ✓
  - Execute on three different software environments
    - Conda
    - Docker
    - System
- Extensibility
  - Create complicated Project Workflows

MLflow Project Documentation



https://mlflow.org/docs/latest/projects.html

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 any platform

mlflow

Models

Deploy machine learning models in diverse serving environments 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

MLflow Model Motivations

ML Frameworks

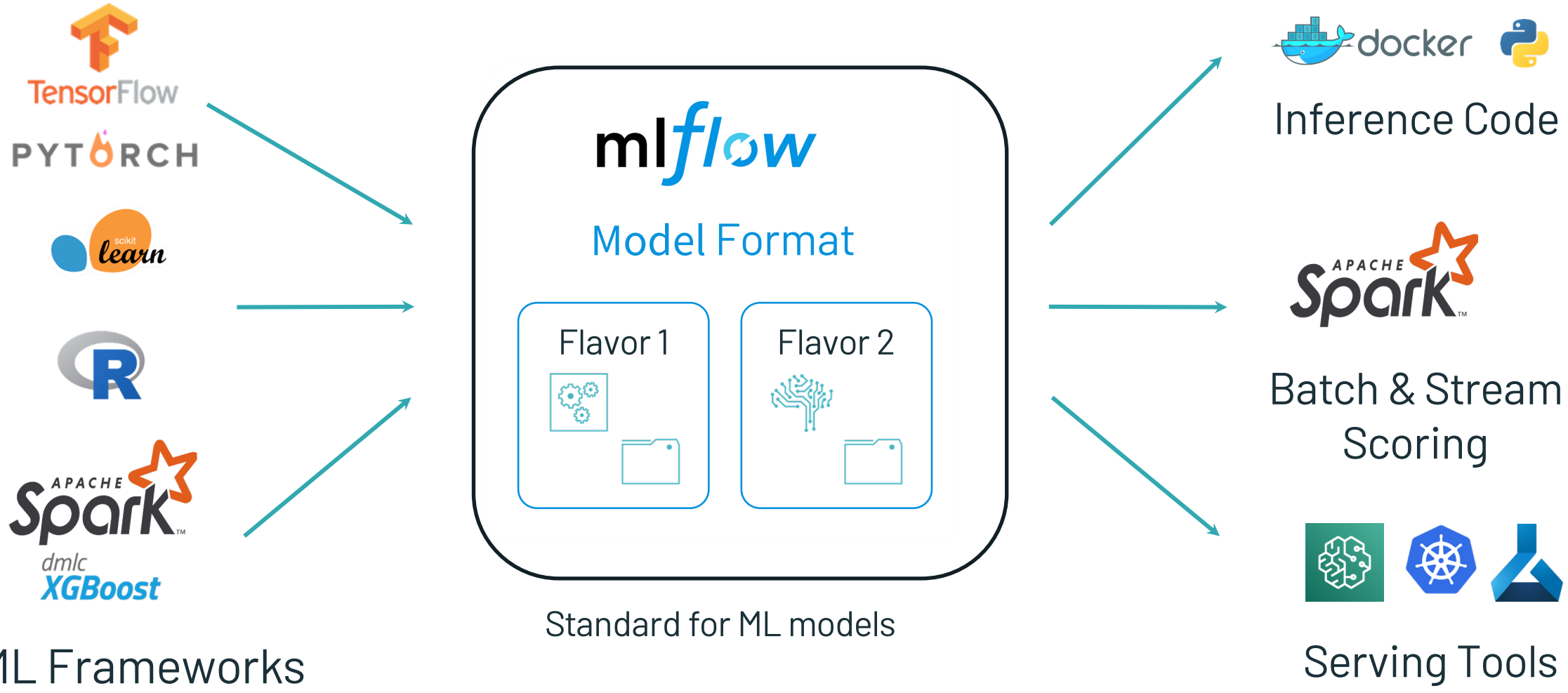
N x M
Combination of
Model support for
all Serving tools

Inference Code

Batch & Stream Scoring

Serving Tools

MLflow Models

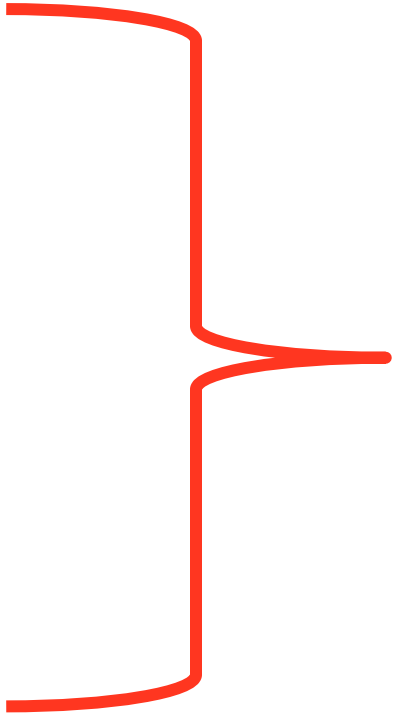


MLflow Model API Documentation

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`)
- Spacy(`spaCy`)
- Fastai(`fastai`)



Model flavor has generic methods:

- log_model()
- save_model()
- load_model()
- add_model_flavor()
- auto_log() *

* Only some model flavors

databricks

https://mlflow.org/docs/latest/models.html#built-in-model-flavors

1. Example MLflow Model

```
mlflow.keras.log_model(...)
```

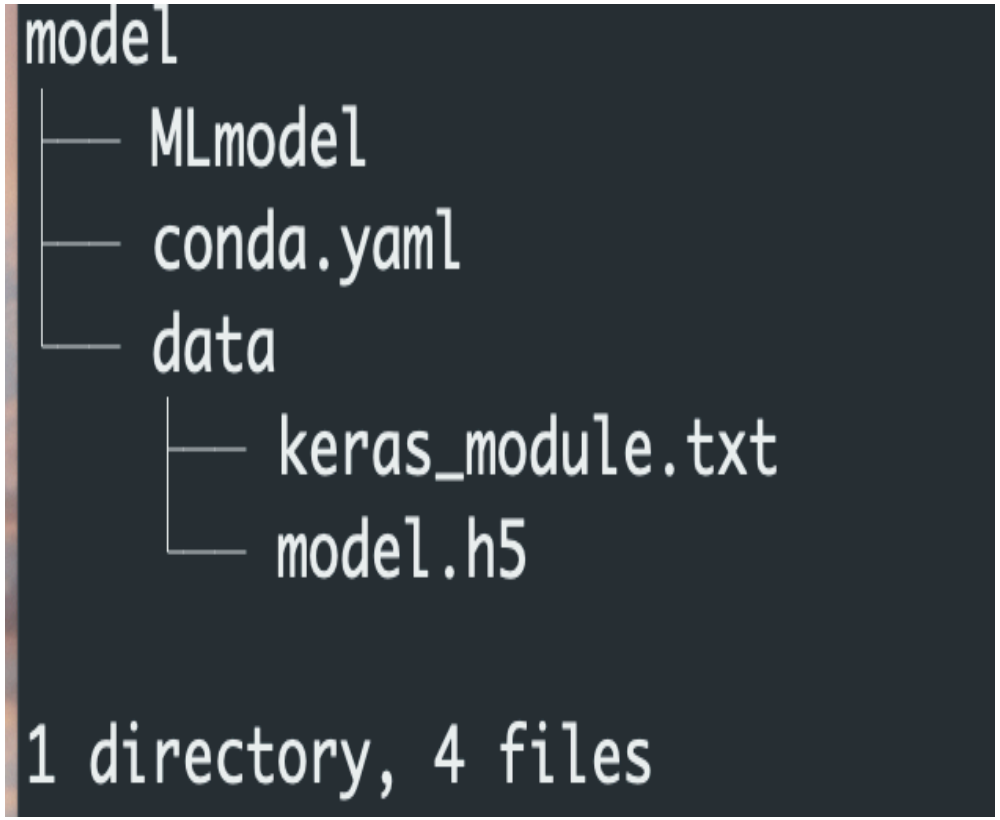
```
model
├── MLmodel
├── conda.yaml
└── data
    ├── keras_module.txt
    └── model.h5

1 directory, 4 files
```

```
artifact_path: model
flavors:
  keras:
    data: data
    keras_module: keras
    keras_version: 2.3.1
  python_function:
    data: data
    env: conda.yaml
    loader_module: mlflow.keras
    python_version: 3.7.5
…
run_id: 714d418027bc43bd8adc768f0f9ecb51
utc_time_created: '2020-08-25 19:04:53.887587'
```

Usable by tools that understand Keras model format

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

2. Example Conda.yaml




```
channels:
- defaults
- conda-forge
dependencies:
- python=3.7.5
- pip
- pip:
  - mlflow
  - keras==2.3.1
  - tensorflow==2.0.0
name: mlflow-env
```

Model Keras Flavor Example

Train a model

`mlflow.keras.log_model(…)`

mlflow

Model Format

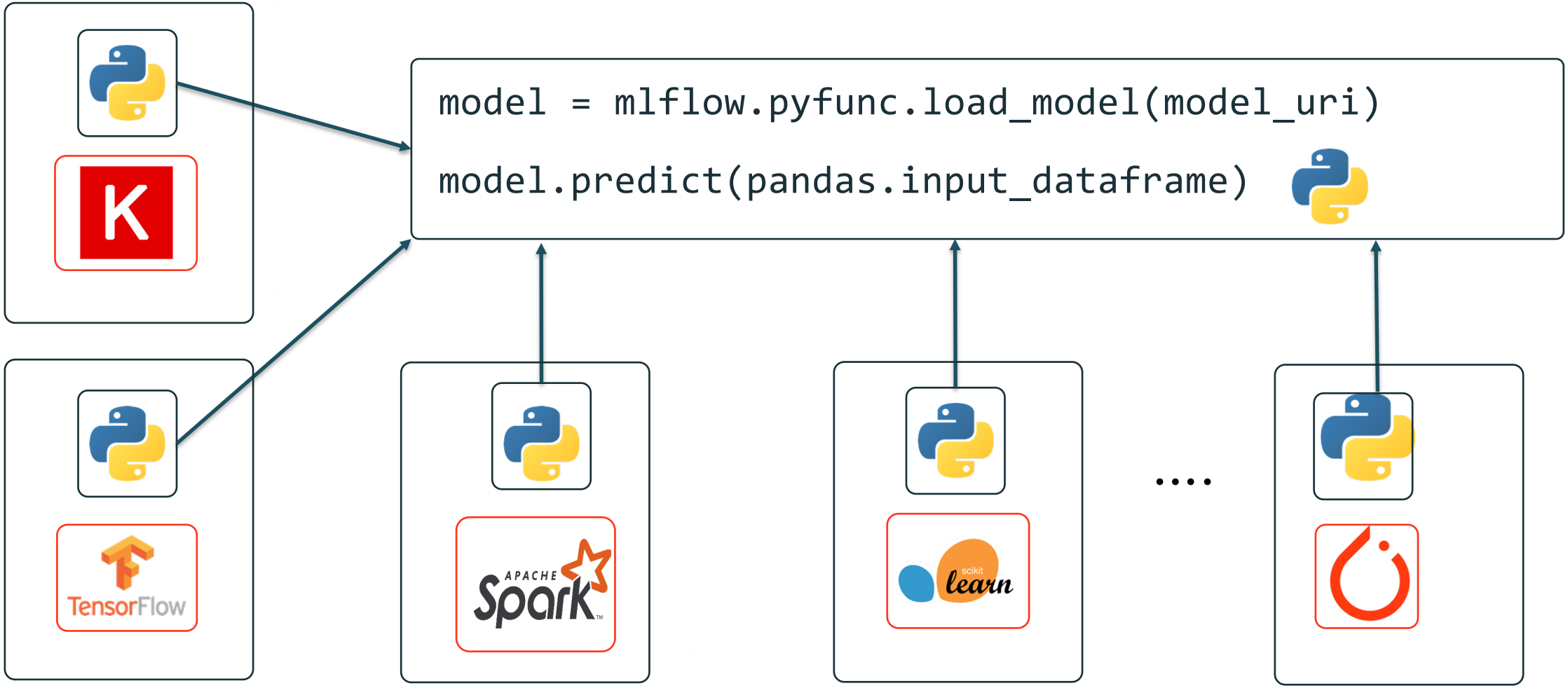
Flavor 1: Pyfunc

Flavor 2: Keras

```
model = mlflow.pyfunc.load_model(…)
Model.predict(pandas.input_dataframe
```

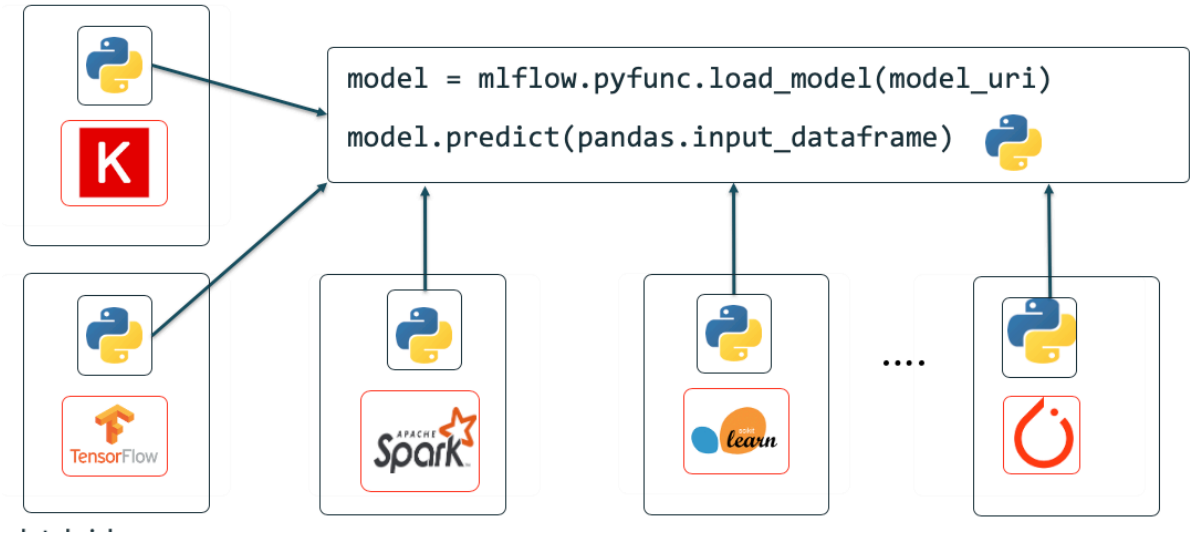
```
model = mlflow.keras.load_model(…)
model.predict(keras.Input(…))
```

Model Flavors Example



MLflow PythonModel

- Serves as Generic Python model interface



- Use with Spark UDFs

```
predict = mlflow.pyfunc.spark_udf(spark, ”runs://run_id/model")
df.withColumn("prediction", predict("name", "age")).show()
```

Recap: 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 and loading utilities

- mlflow.<model_flavor>.save_model(...) or log_model(...) or autolog()
- mlflow.<model_flavor>.load_model(...)

Deployment APIs

- CLI / Python / R / Java
- mlflow models [OPTIONS] COMMAND [ARGS]...
  - mlflow models serve [OPTIONS [ARGS] ....
  - mlflow models predict [OPTIONS [ARGS] ...
  - mlflow sagemaker –help
  - mlflow azureml --help

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 any 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

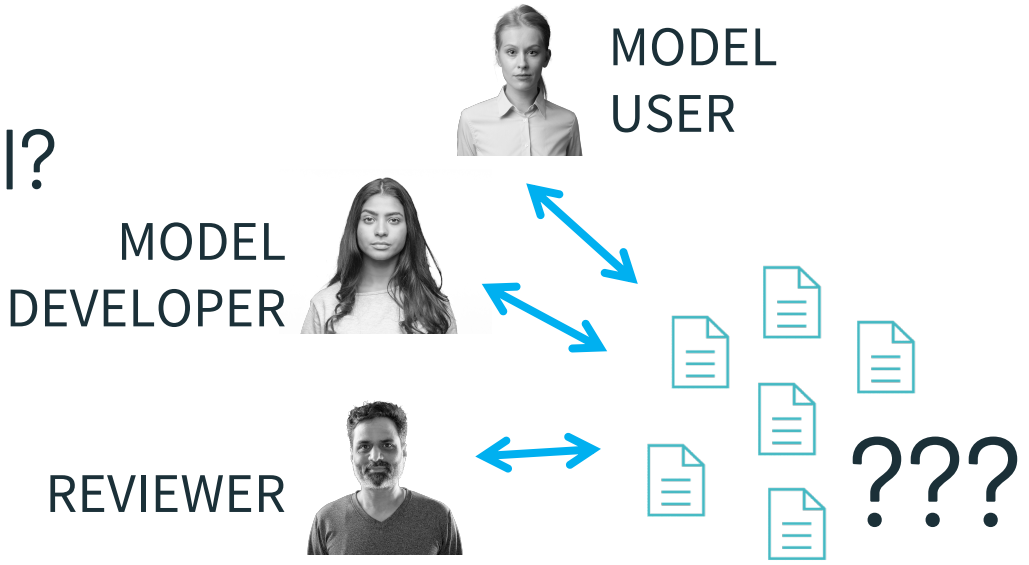
The Model Management Problem

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

The Model Management Problem

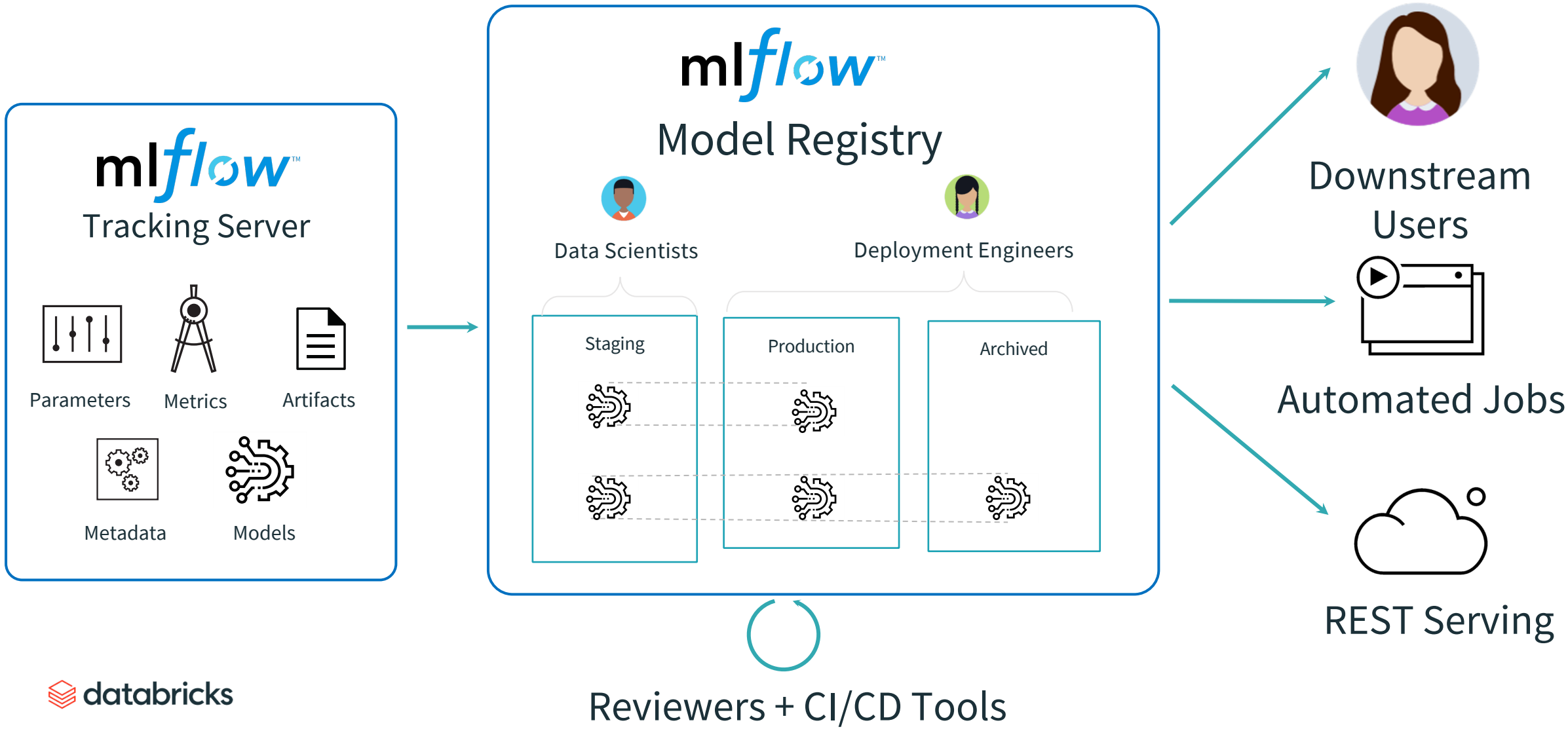
When you work in a large organization with many models, many data teams, management becomes a major challenge:

- 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?
- How can I integrate with CI/CD?



mlflow Model Registry

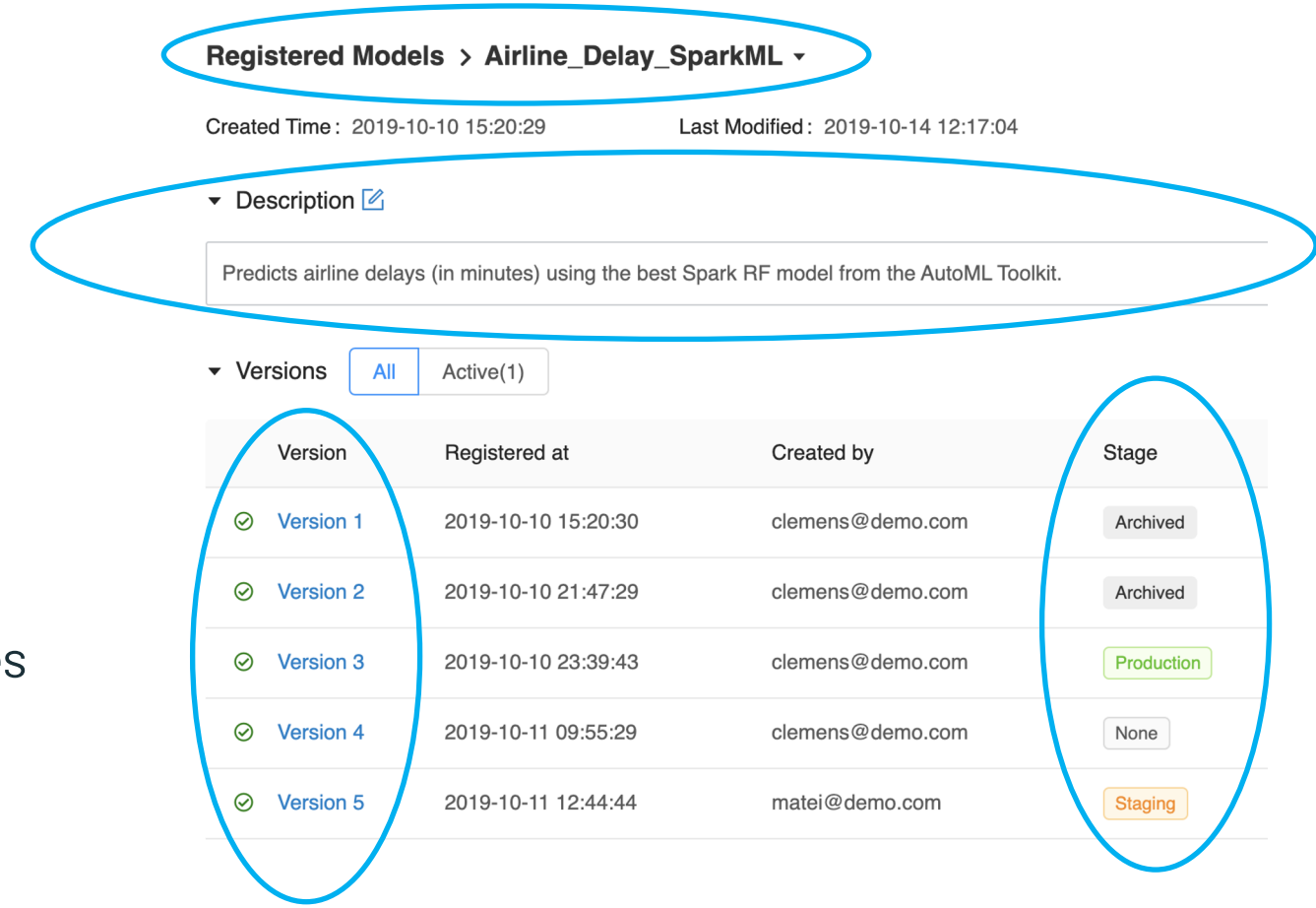
VISION: Centralized and collaborative model lifecycle management



databricks

MLflow Model Registry

- Repository of named, versioned models with controlled Access to Models
- Track each model's stage: none, staging, production, or archived
- Easily inspect a specific version and its run info
- Easily load a specific version
- Provides model description, lineage and activities



databricks

MLflow Model Registry

The MLflow Model Registry component is a centralized model store, set of APIs, and UI, to collaboratively manage the full lifecycle of an MLflow Model. It provides model lineage (which MLflow experiment and run produced the model), model versioning, stage transitions (for example from staging to production), and annotations.

Table of Contents

`mlflow.register_model(model_uri, name)` [source]

`mlflow.get_registry_uri()` [source]

`mlflow.set_registry_uri(uri)` [source]

MLflow Model Registry

The MLflow Model Registry component is a centralized model store, set of APIs, and UI, to collaboratively manage the full lifecycle of an MLflow Model. It provides model lineage (which MLflow experiment and run produced the model), model versioning, stage transitions (for example from staging to production), and annotations.

Table of Contents

Model Registry CRUD Operations
MLflowClient()

create_model_version(*name, source, run_id, tags=None, run_link=None, description=None*) [source]

create_registered_model(*name, tags=None, description=None*) [source]

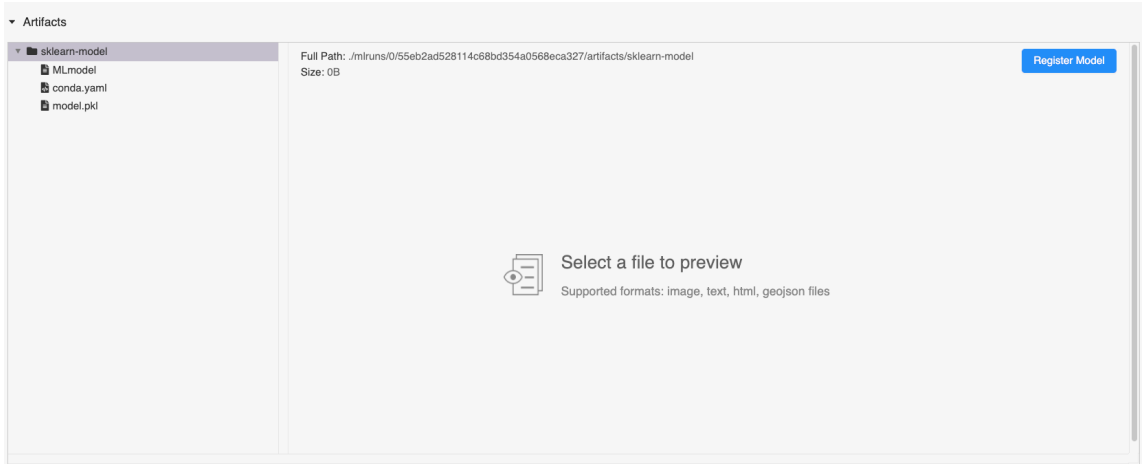
delete_model_version(*name, version*) [source]

get_latest_versions(*name, stages=None*) [source]

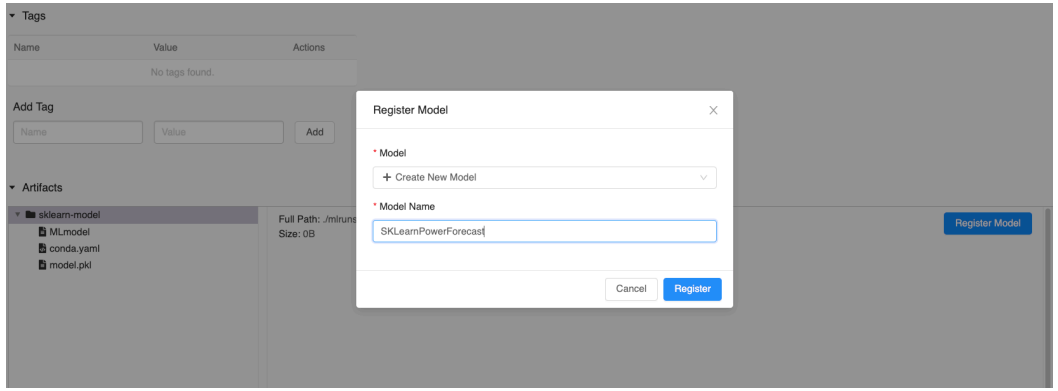
transition_model_version_stage(*name, version, stage, archive_existing_versions=False*) [source]

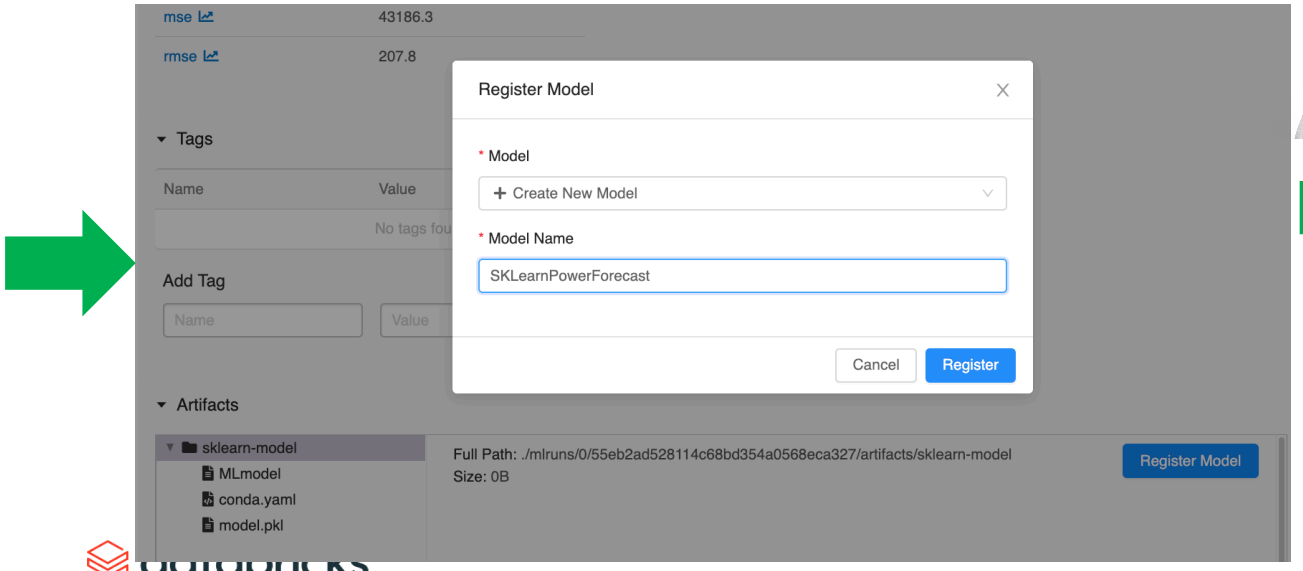
databricks

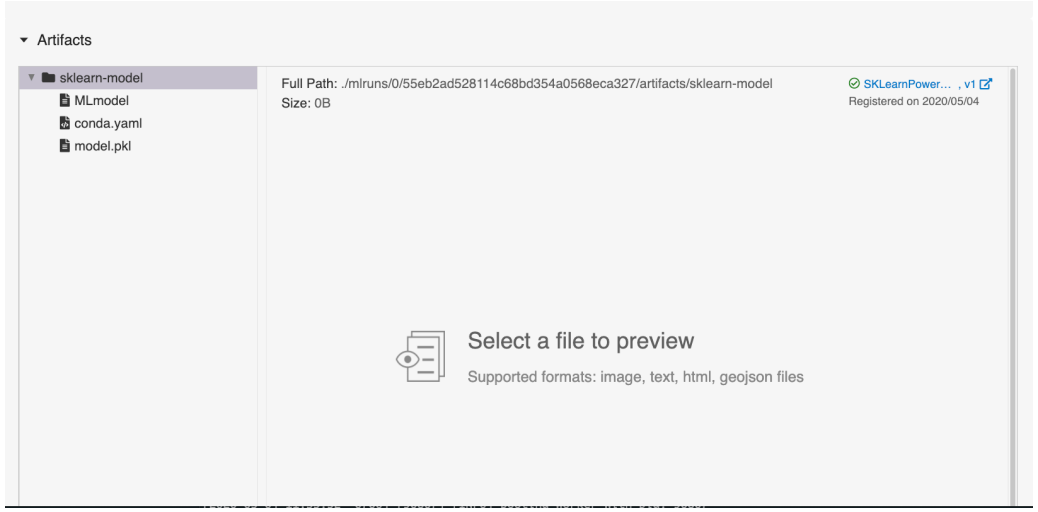
Model Registry Workflow UI



MODEL DEVELOPER

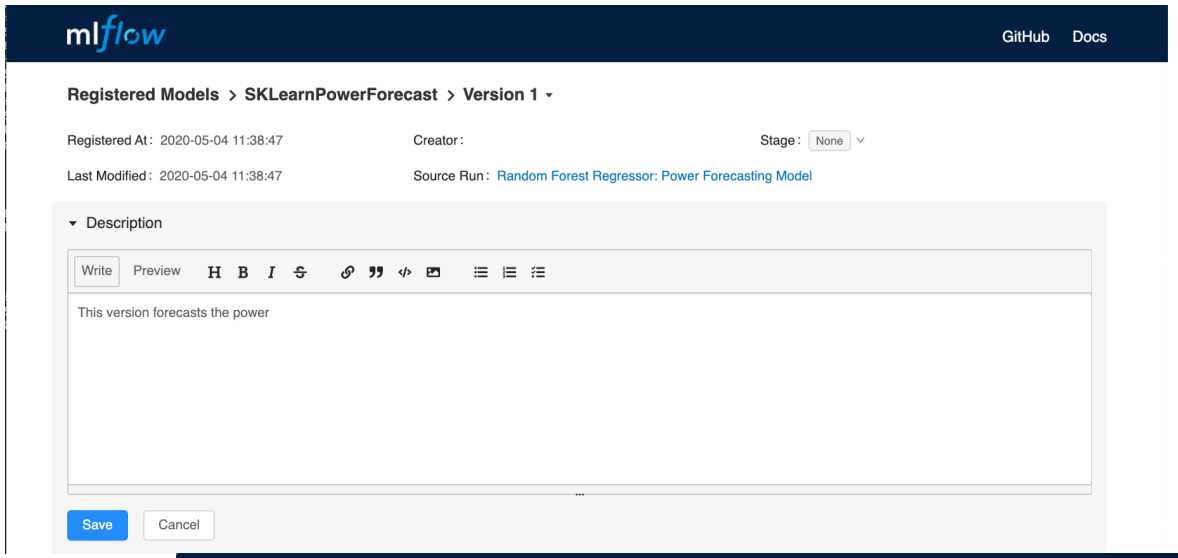




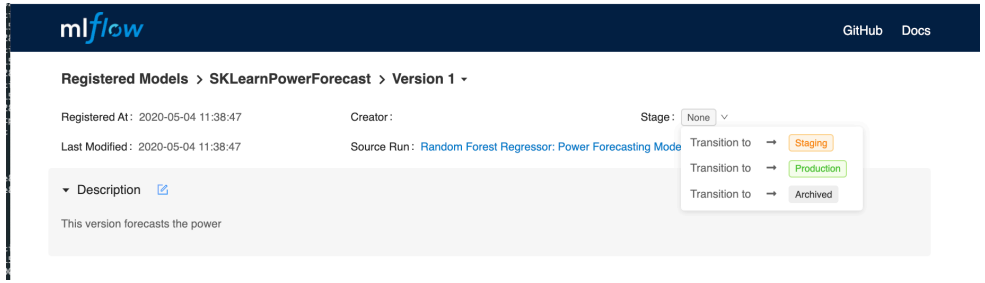


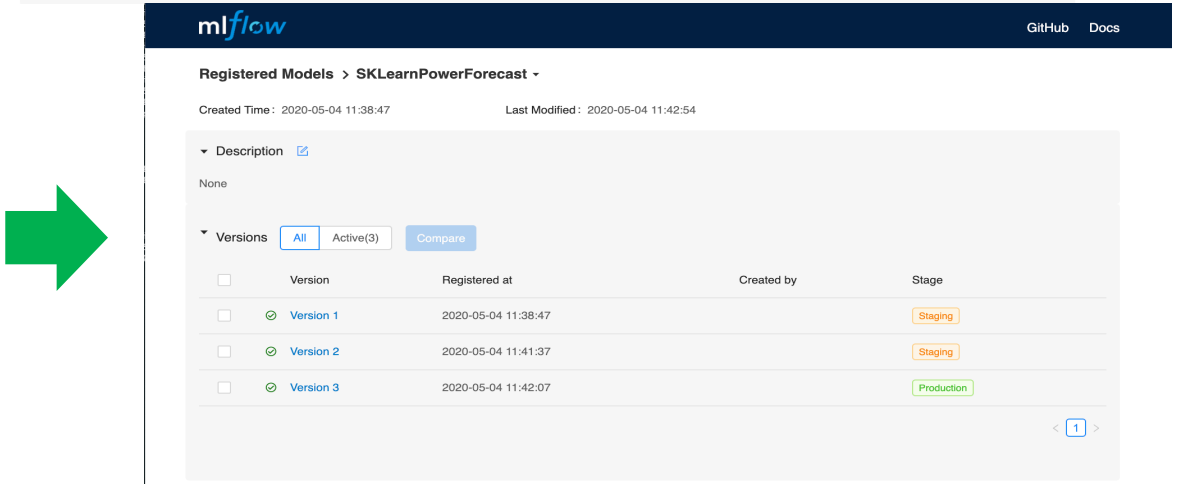
databricks

Model Registry Workflow UI



MODEL REVIEWER





DOWNSTREAM USERS

AUTOMATED JOBS

REST SERVING

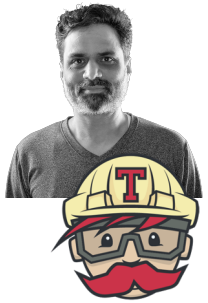
databricks

Model Registry Workflow API

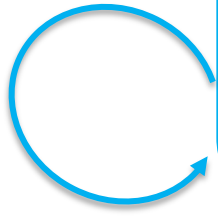
```
mlflow.register_model(model_uri,"WeatherForecastModel")

mlflow.sklearn.log_model(model,
          artifact_path="sklearn_model",
          registered_model_name= "WeatherForecastModel")
```

```
model_uri= "models:/{model_name}/production".format(
          model_name="WeatherForecastModel")
model_prod =  mlflow.sklearn.load_model(model_uri)
model_prod.predict(data)
```

MODEL DEVELOPER

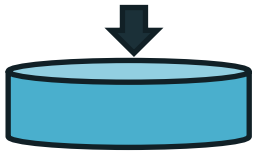


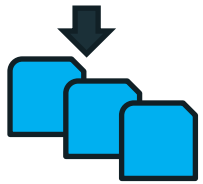
REVIEWERS, CI/CD TOOLS



mlflow™

Model Registry





DOWNSTREAM USERS

AUTOMATED JOBS

REST SERVING

```
client = mlflow.tracking.Mlflowclient()
client.transition_model_version_stage(name="WeatherForecastModel",
                                      version=5,
                                      stage="Production")
```

databricks

Model Registry Workflow API

```
mlflow.register_model(model_uri,"WeatherForecastModel")

mlflow.sklearn.log_model(model,
          artifact_path=”sklearn_model”,
          registered_model_name= “WeatherForecastModel”)
```

```
model_uri = "models:/{model_name}/production".format(
          model_name="WeatherForecastModel")
model_prod = mlflow.pyfunc.load_model(model_uri))
model_prod.predict(data)
```

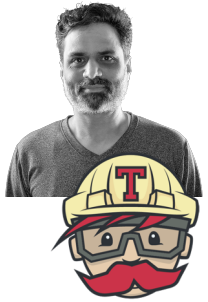
MODEL DEVELOPER

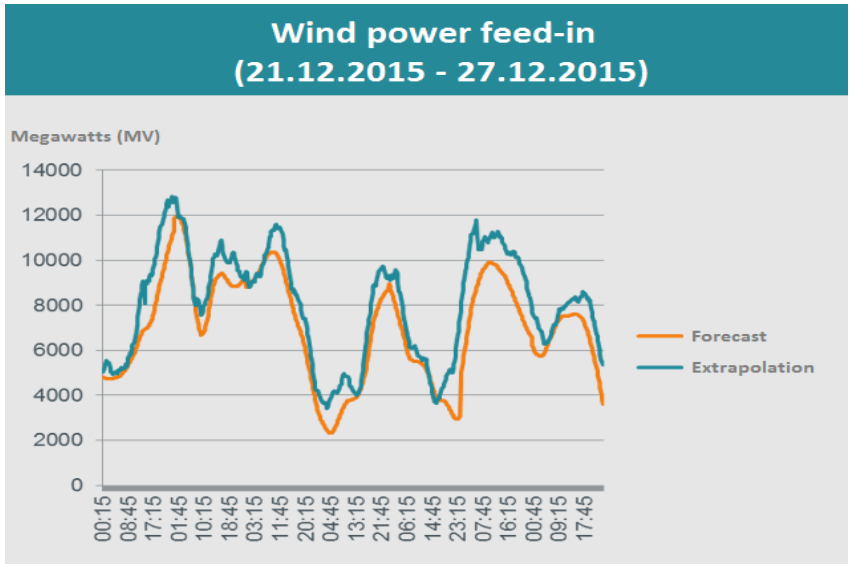
mlflow™

Model Registry

DOWNSTREAM FORCASTING APP

REVIEWERS, CI/CD TOOLS





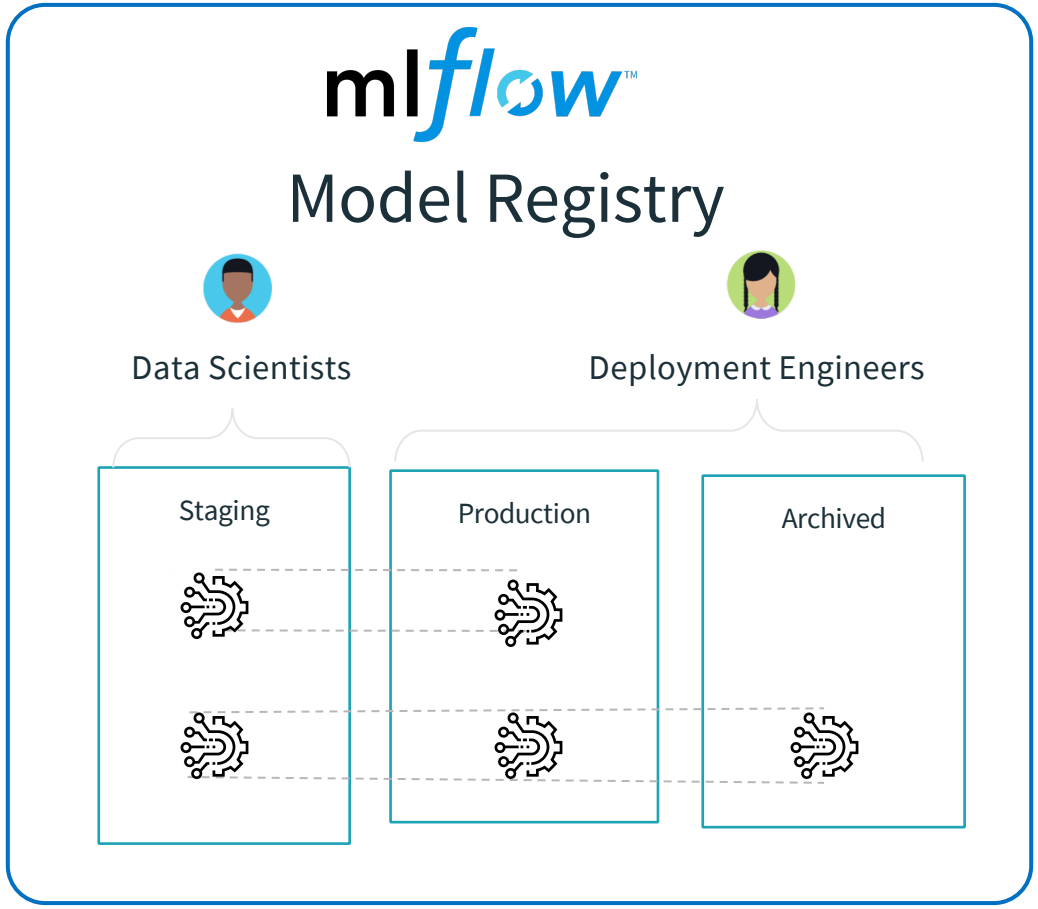

```
client = mlflow.tracking.Mlflowclient()
client.transition_model_version_stage(name=”WeatherForecastModel”,
                                      version=5,
                                      stage="Production")
```

databricks

O'REILLY

MLflow Model Registry Recap

- Central Repository: Unique named registered models for discovery across data teams
- Model Registry Workflow: Provides UI and API for registry operations
- Model Versioning: Allow multiple versions of model in different stages
- Model Stages: Allow stage transition: none, staging, production, or archived
- CI/CD Integration: Easily load a specific version for testing and inspection
- Model Lineage: Provides model description, lineage and activities



Recap of all parts: What Did We Talk About?

- Modular Components greatly simplify the ML lifecycle
- Easy to install & Great Developer experience
- Develop & Deploy locally; track locally or remotely
- Available APIs: Python, Java & R (Soon Scala)
- REST APIs and CLI tools
- Visualize experiments and compare runs
- Centrally register and manage model lifecycle

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 Tutorial

Tutorials: https://github.com/dmatrix/ds4g-workshop

Thank you! ☺

Q & A

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