A platform for the Complete Machine Learning Lifecycle

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

Machine Learning Lifecycle

Raw data

Data prep

Deploy

Train

Problems

- Reproduce experiments
- Compare experiments
- Fine tune previous experiments across teams
- Share data, parameters or metrics
- Deploy trained models

It is difficult to productionize and share

Are there any similar solutions in an open manner?

MLflow: An Open Source ML Platform

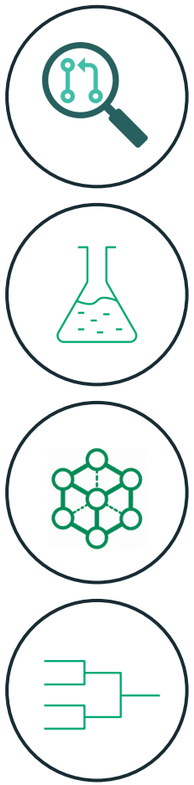
Works with any ML library, programming language, deployment tool

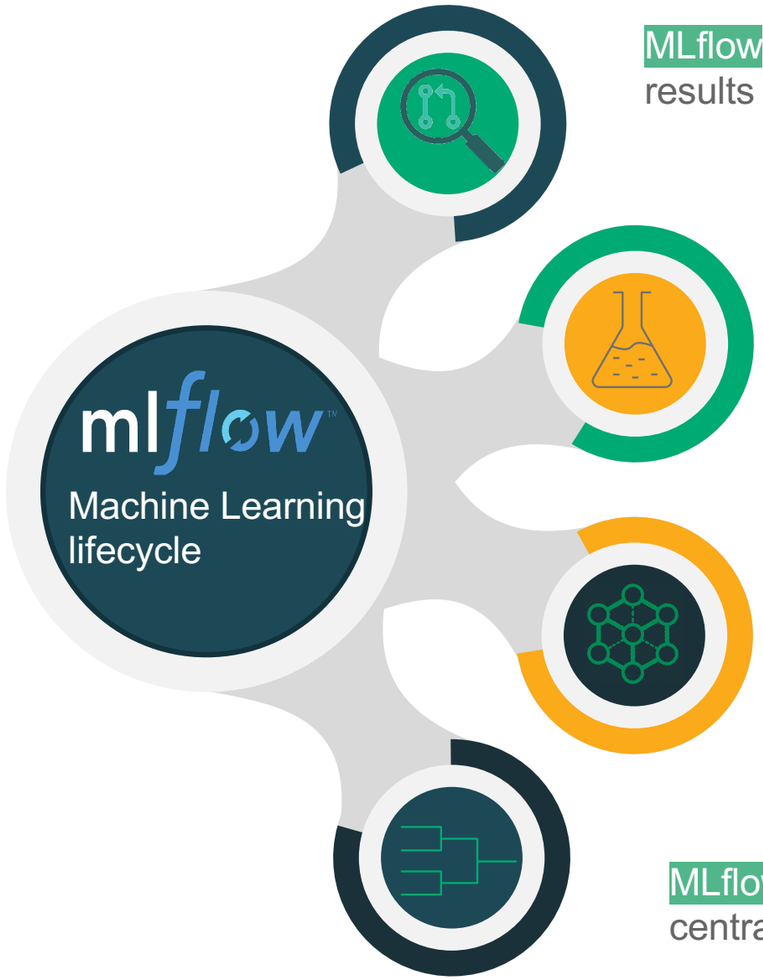
Three components:

- **Tracking:** experiment tracking
- **Projects:** reproducible runs
- **Models:** model packaging

140 contributors, 800K downloads/month

MLFlow Components





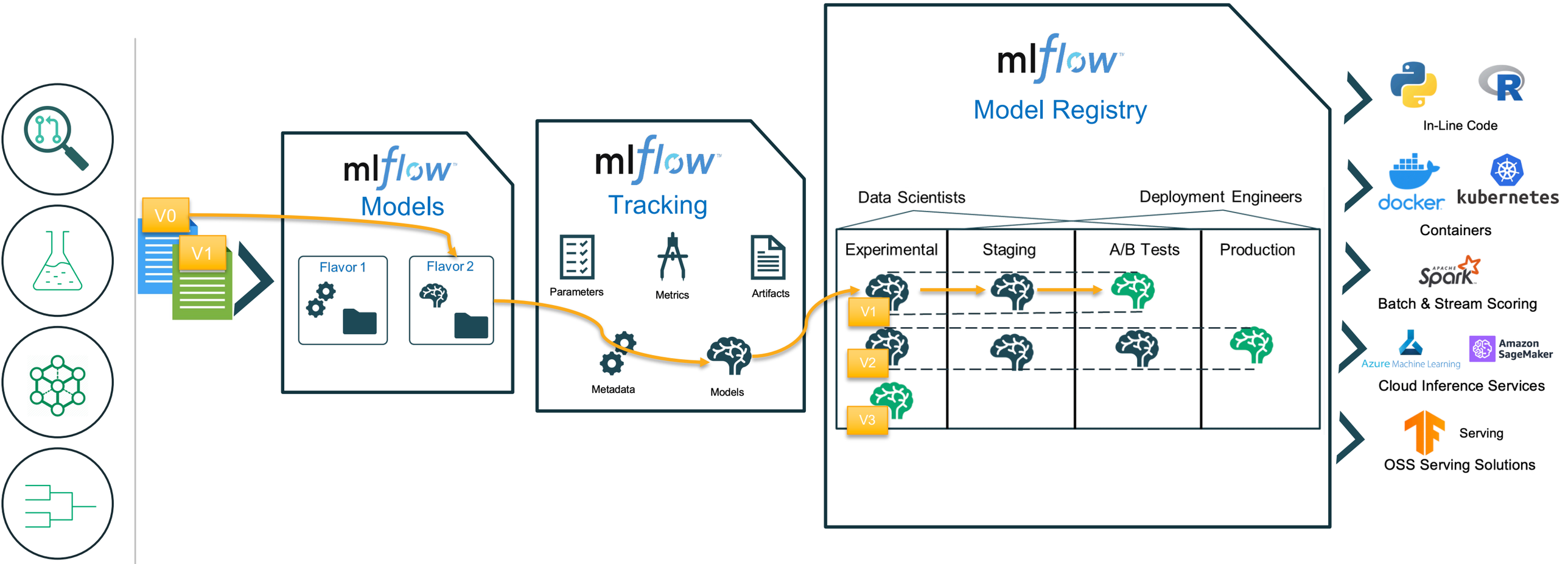
MLflow Tracking - Record and query experiments: code, data, config, and results

MLflow Projects - Package data science code in a format to reproduce runs on any platform

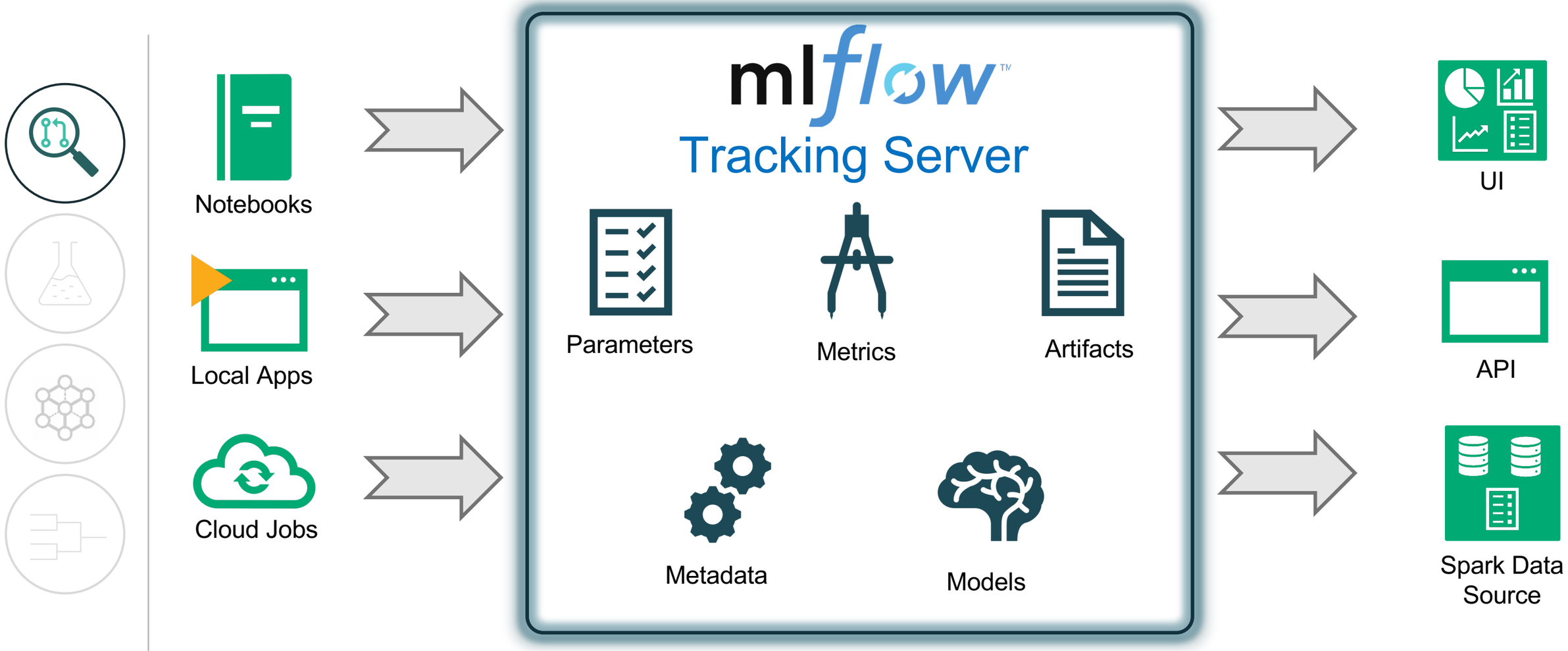
MLflow Models - Deploy machine learning models in diverse serving environments

MLflow Registry - Store, annotate, discover, and manage models in a central repository

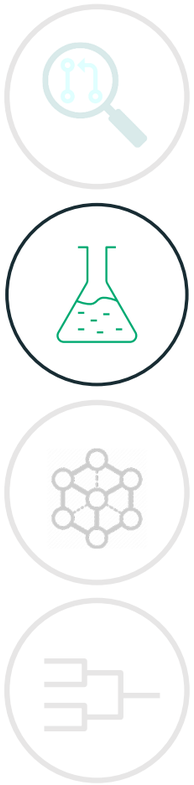
Mlflow Workflow

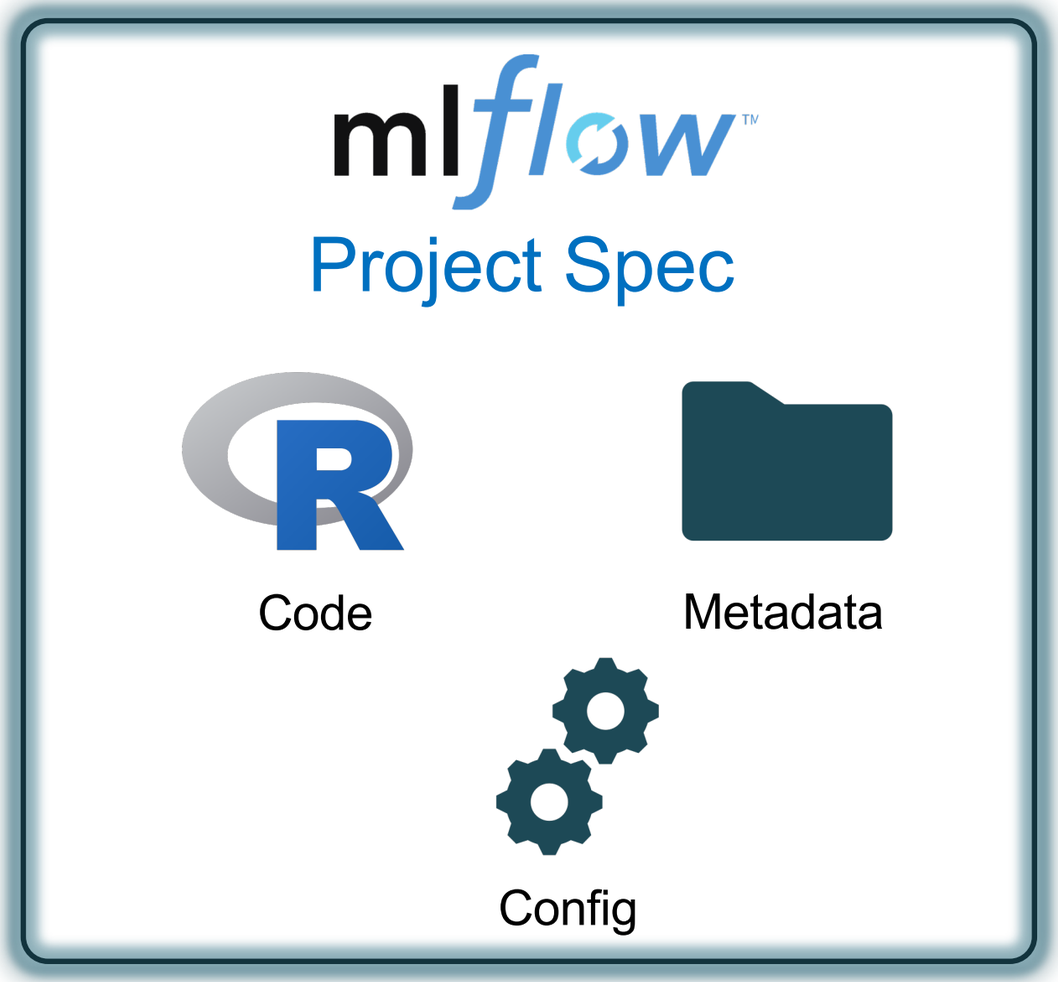


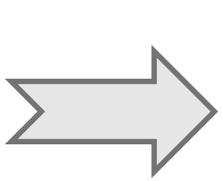
Mlflow Tracking

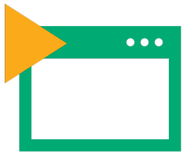


Mlflow Projects

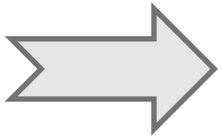






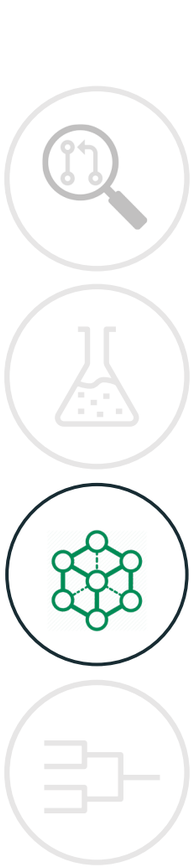


Local Execution



Remote Execution

Mlflow Models



ML Libraries

scikit learn · TensorFlow · Apache Spark · H2O.ai · PyTorch · mleap · ONNX · Keras

mlflow™

Model Format

Flavor 1

Flavor 2

In-Line Code

docker · kubernetes

Containers

mleap · Apache Spark

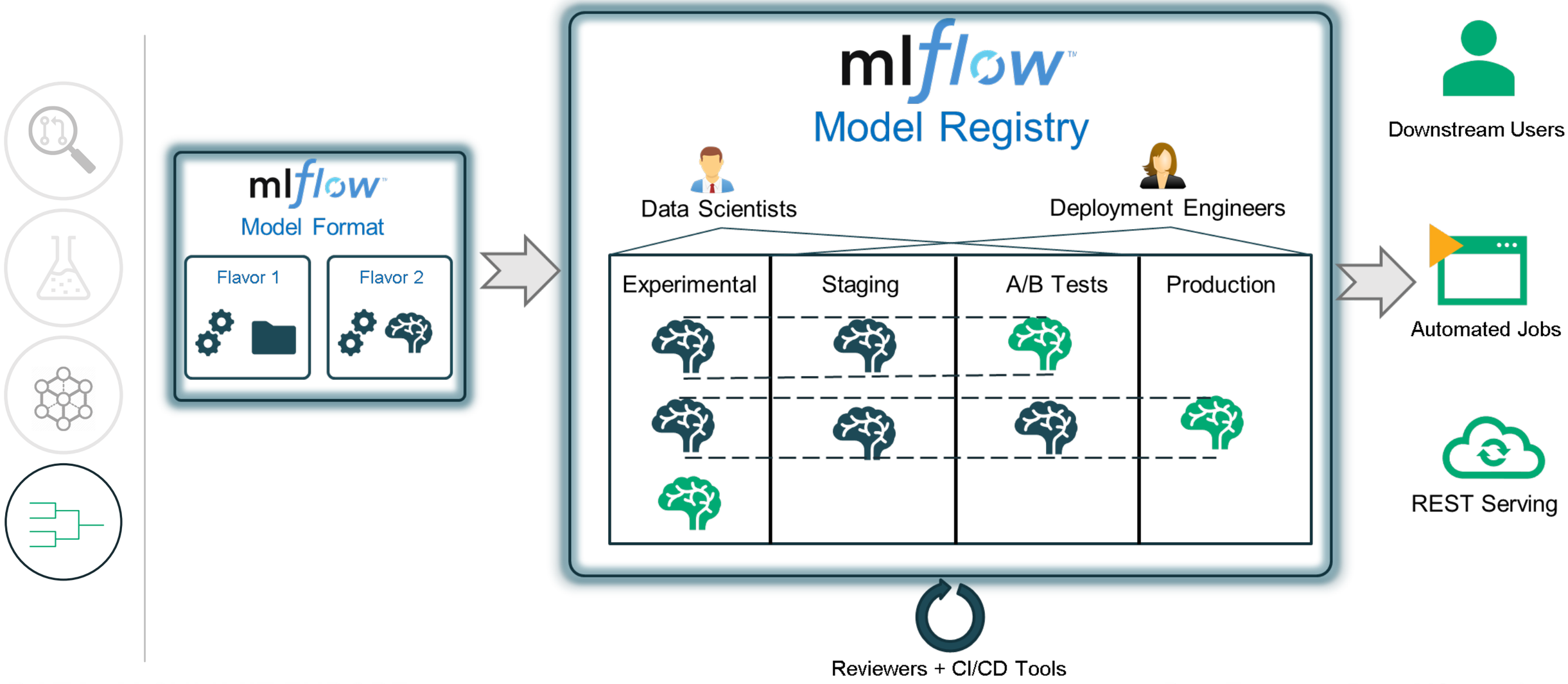
Cloud Inference Services

Azure Machine Learning · Amazon SageMaker

Cloud Inference Services

Simple model flavors
usable by many tools

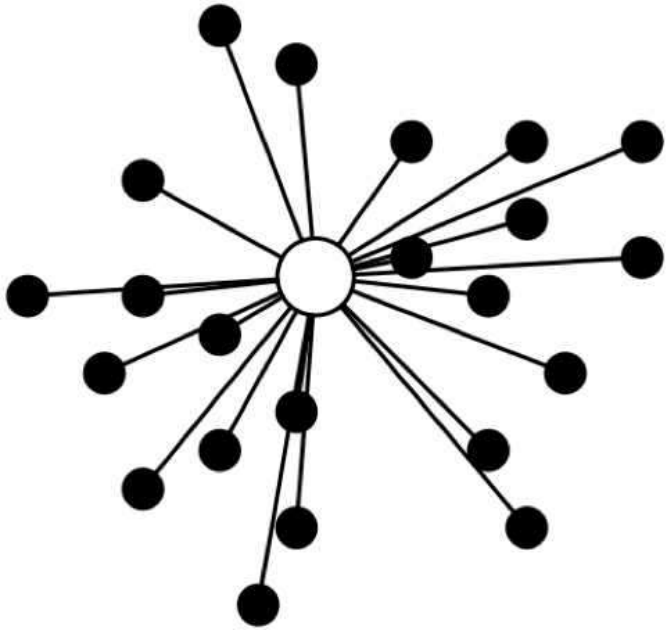
Mlflow Registry



MLFlow Tracking

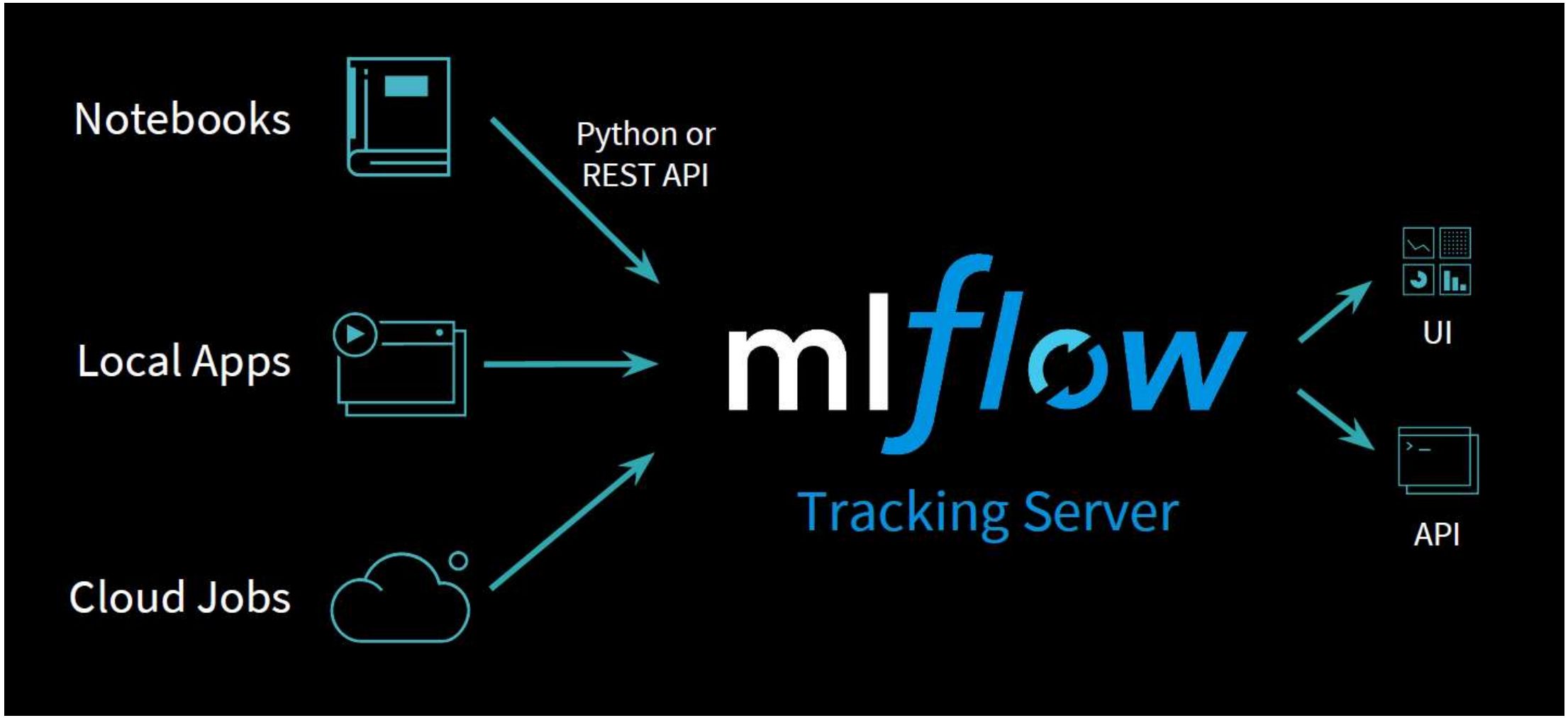
- Record and query experiments:
  - Data
  - Code
  - Model parameters
  - Results (performance metrics)
  - Model

Motivation



Centralised repository of useful information to analyse several runs of training.

How it is working



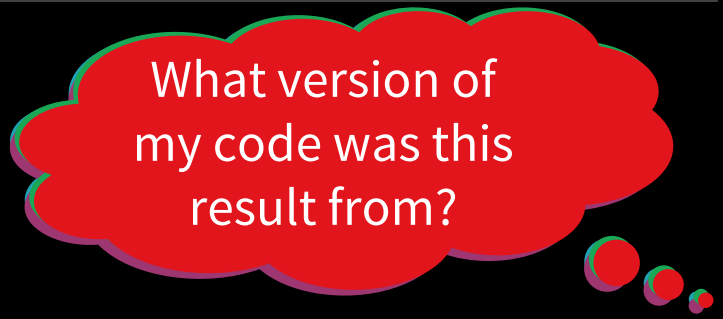
Key concepts

- Parameters
- Metrics
- Tags and notes
- Artifacts
- Source
- Version

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
...
```



Model Deployment without MLflow

DATA SCIENTIST

Please deploy this ArXiv paper!

Code & Models

…

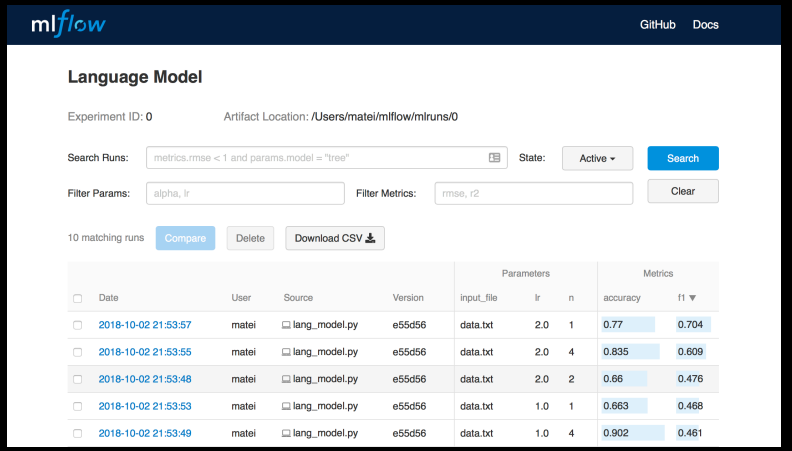
PRODUCTION ENGINEER

Model Development *with* 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"))
```

Model Development *with* MLflow

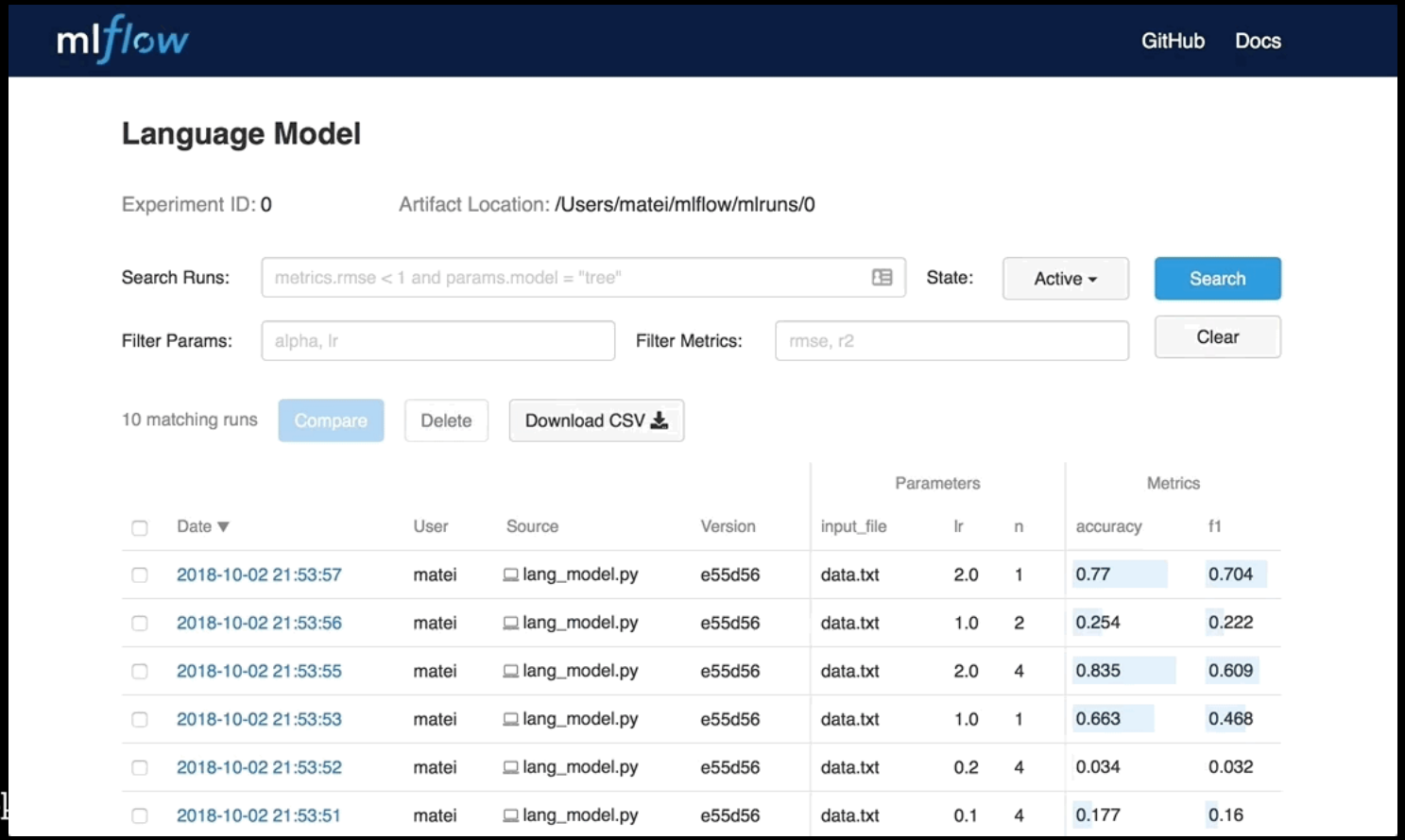
```
data   = load_text(file)
ngrams = extract_ngrams(data, N=n)
model  = train_model(ngrams,
                     learning_rate=lr)
score  = compute_accuracy(model)

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)
```



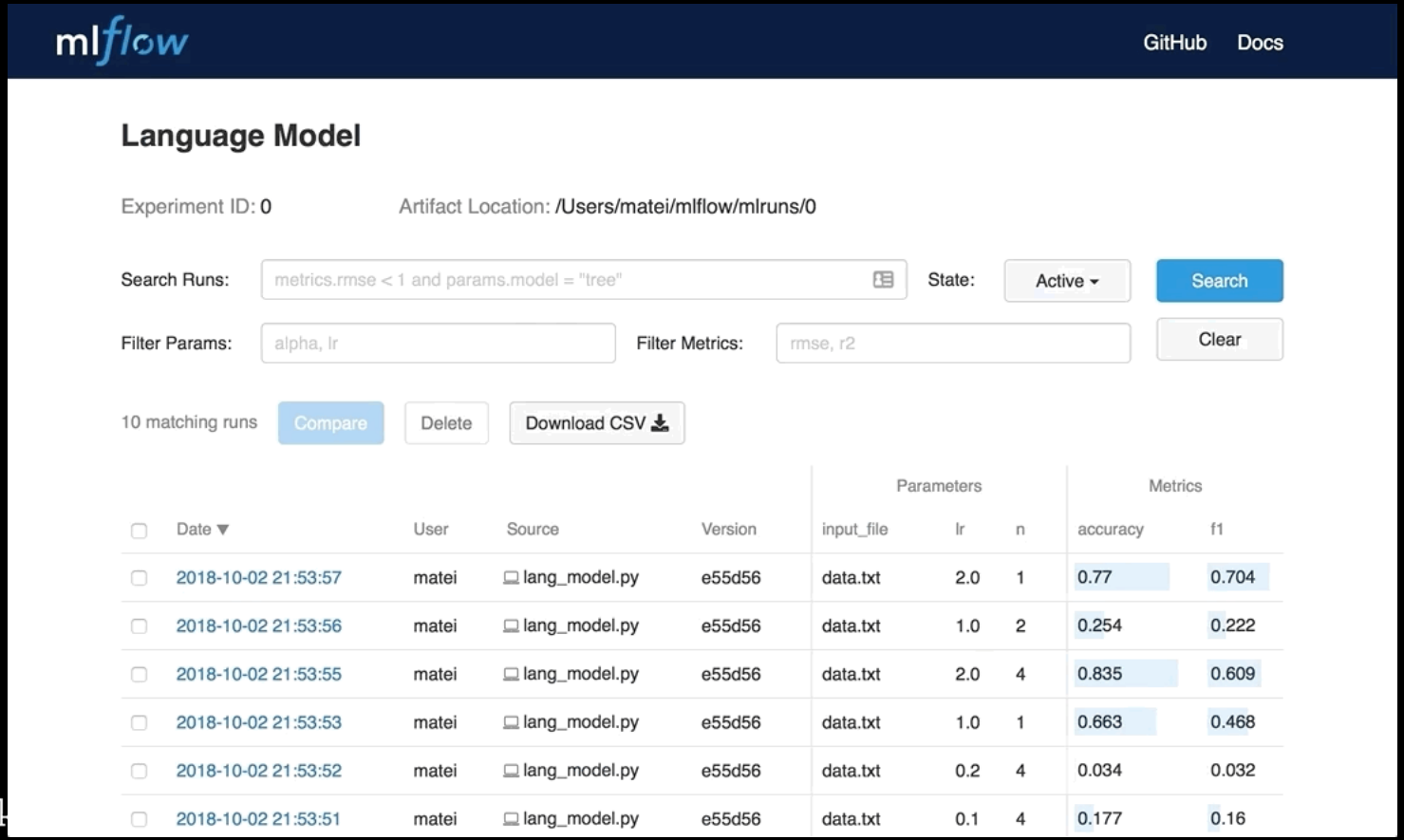
Track parameters, metrics, output files & code version

Search using UI or API

MLflow UI: Inspecting Runs



MLflow UI: Comparing Runs



Code example

```python
import mlflow

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

    # train model
    model = train_model(layers, alpha)

    mlflow.log_metric("mse", model.mse())
    mlflow.log_artifact("plot", model.plot(test_df))
    mlflow.tensorflow.log_model(model
```

Demo

```
# Enable MLflow Autologging
mlflow.keras.autolog()

X, y = get_training_data()opt = keras.optimizers.Adam(lr=params["learning_rate"],
                                                     beta_1=params["beta_1"],
                                                     beta_2=params["beta_2"],
                                                     epsilon=params["epsilon"]

model = Sequential()model.add(Dense(int(params["units"]), ...)
model.add(Dense(1))
model.compile(loss="mse", optimizer=opt)

rest = model.fit(X, y, epochs=50, batch_size=64, validation_split=.2)
```

MLflow Autologging

```
model = keras.models.Sequential()
model.add(layers.Dense(hidden_units, ...))
model.fit(X_train, y_train)
test_loss = model.evaluate(X_test, y_test)
```

MLflow Autologging

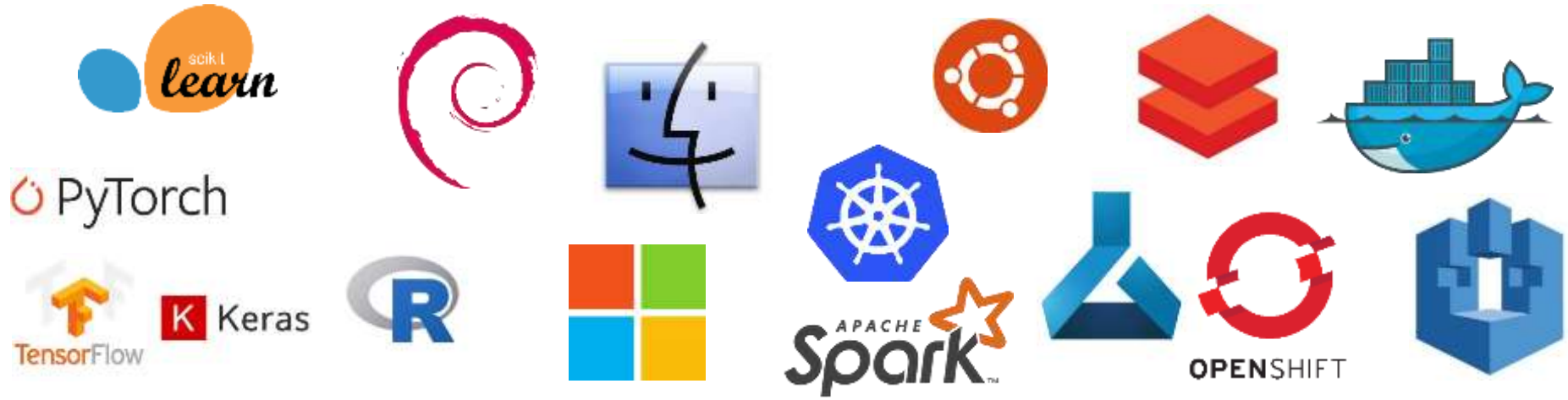
```
with mlflow.start_run():
    model = keras.models.Sequential()
    model.add(layers.Dense(hidden_units, ...))
    model.fit(X_train, y_train)
    test_loss = model.evaluate(X_test, y_test)
    mlflow.log_param("hidden_units", hidden_units)
    mlflow.log_param("learning_rate", learning_rate)
    mlflow.log_metric("train_loss", train_loss)
    mlflow.log_metric("test_loss", test_loss)
    mlflow.keras.log_model(model)
```

```
mlflow.keras.autolog()
model = keras.models.Sequential()
model.add(layers.Dense(hidden_units, ...))
model.fit(X_train, y_train)
test_loss = model.evaluate(X_test, y_test)
```

MLFlow Projects

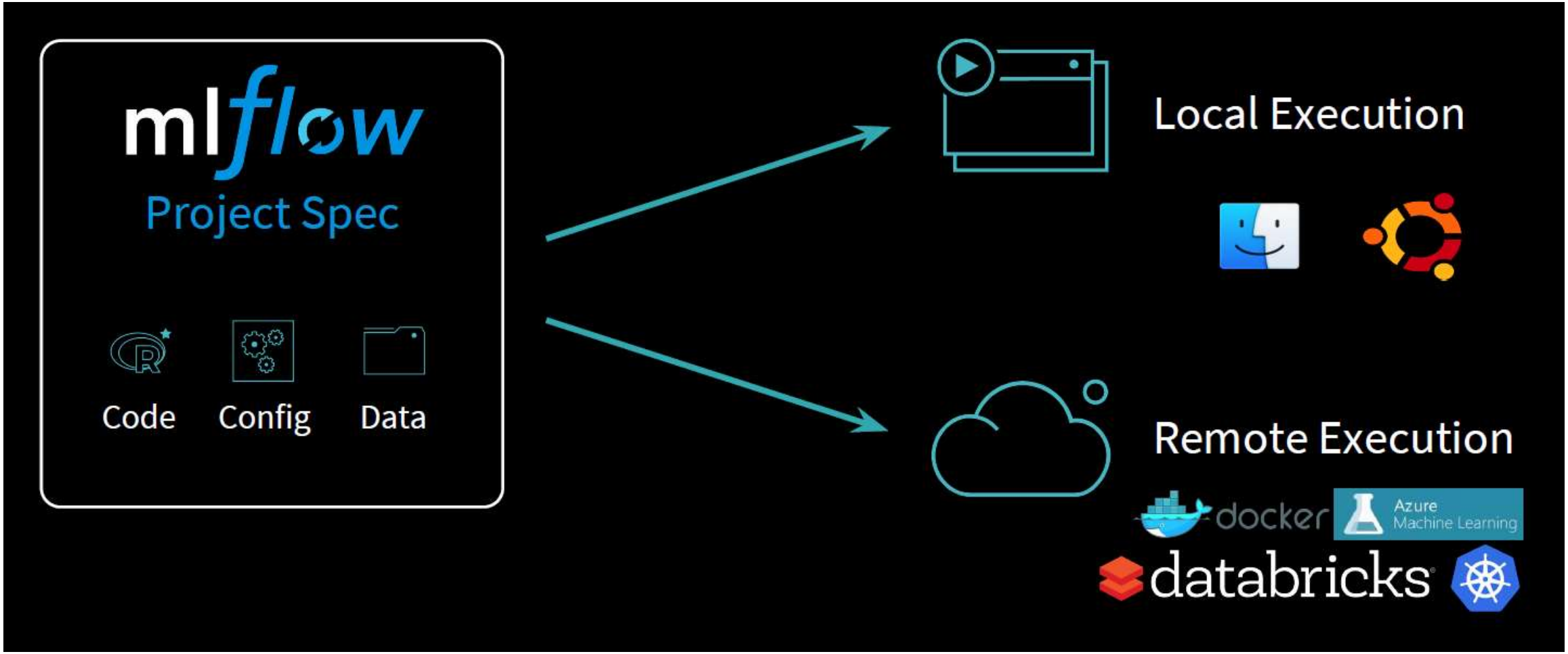
- Packaging format for reproducible ML runs
- Defines dependencies for reproducibility
- Execution API for running projects locally or remote

Motivation



Diverse set of tools and environments involves difficulty to productionalize and share ML work

How it is working



Key concepts

- MLproject file
- Entry points
- Environments
  - Conda
  - Docker
  - System
- Run

plain concepts

Code example

```
my_project/
  |- MLproject
  |
  |
  |
  |
  |
  |
  |- conda.yaml
  |- main.py
  |- model.py
  ...
```

```yaml
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@github.com:mlflow/mlflow-example.git -P alpha=0.5

mlflow run <uri> -m databricks --cluster-spec <json-cluster-spec>
```

Model Deployment *with* MLflow

DATA SCIENTIST

Please run this MLflow Project nightly for updates!

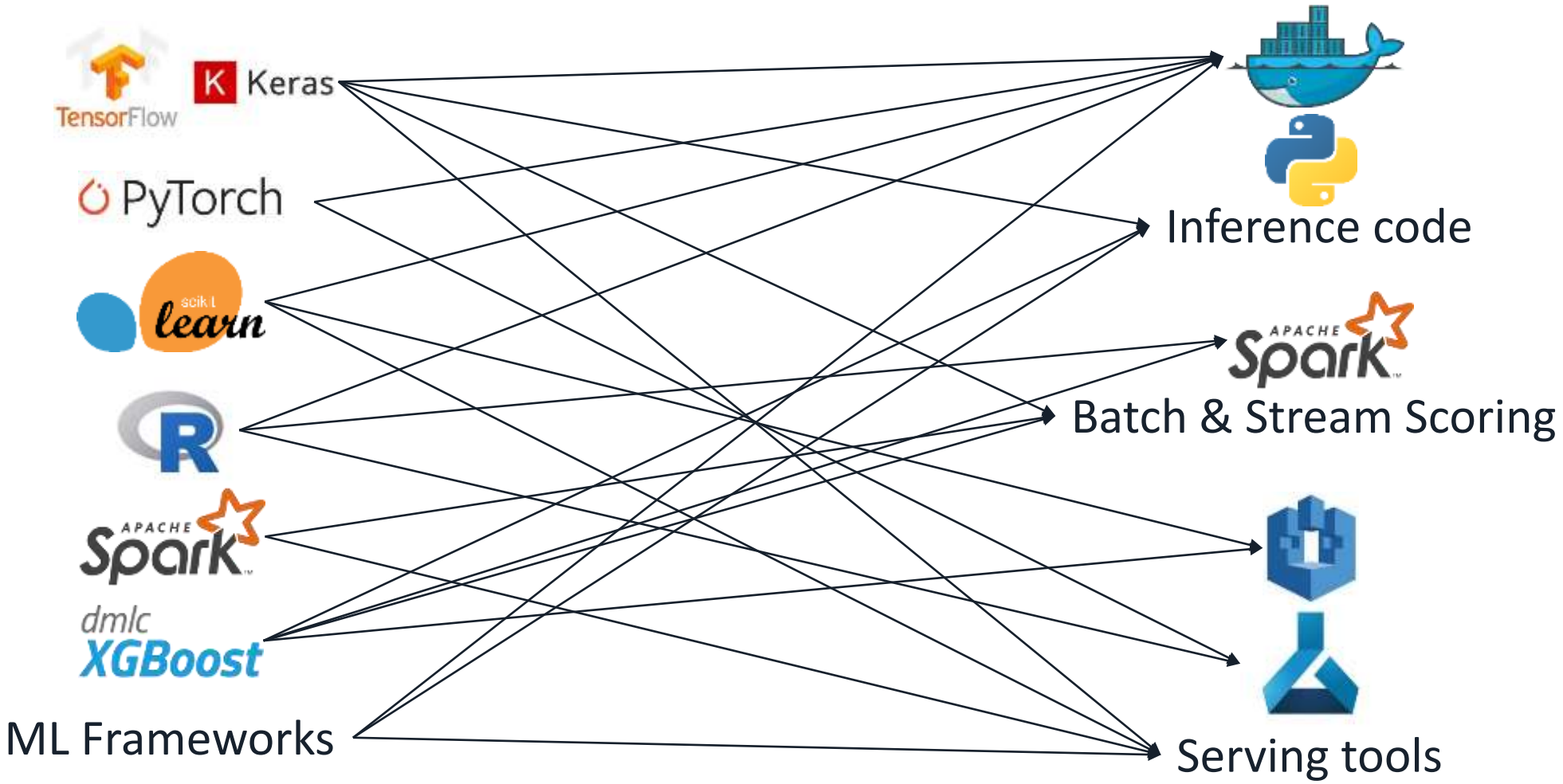
PRODUCTION ENGINEER

Don’t even tell me what ArXiv paper that’s from...

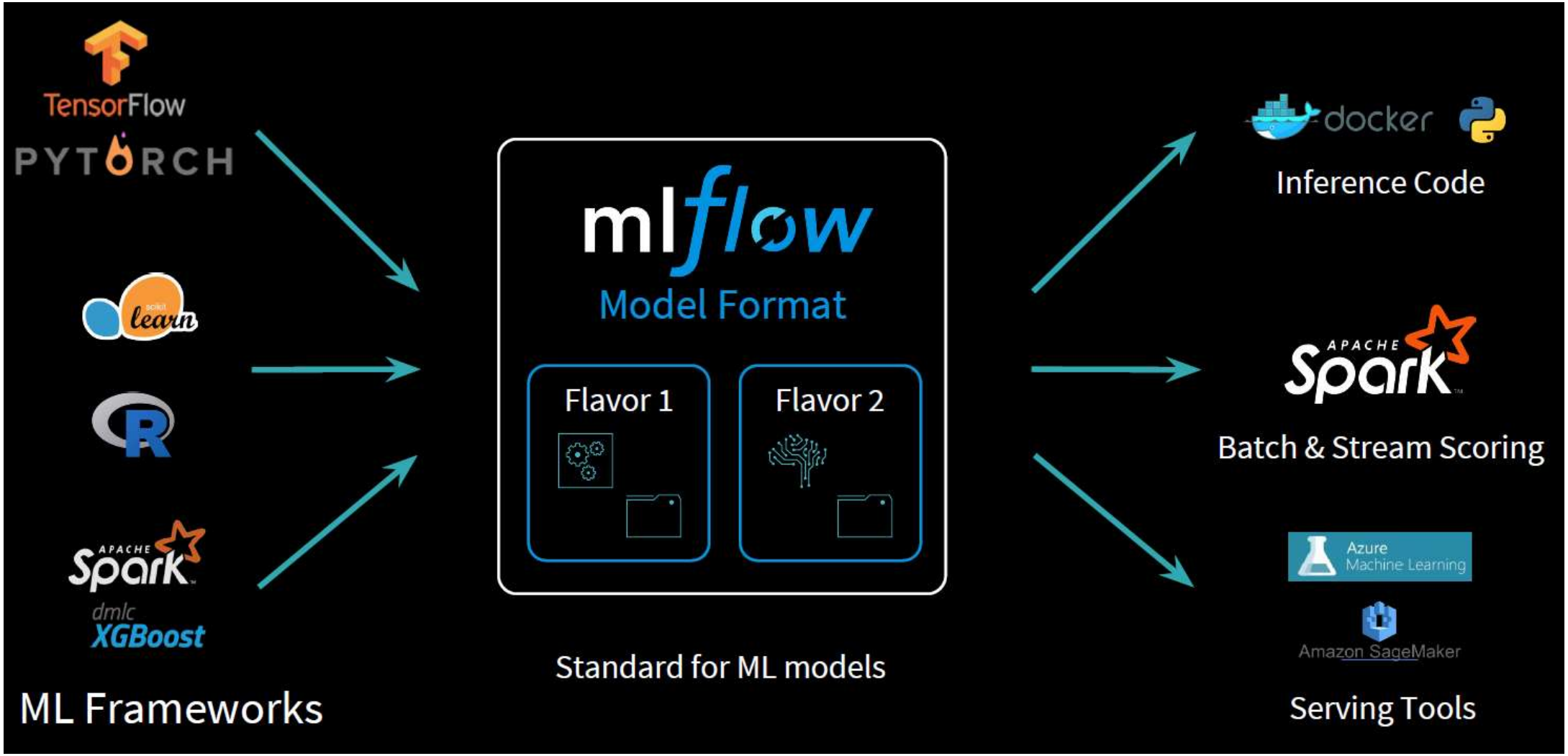
MLFlow Models

- Packaging format for ML Models
- Defines dependencies for reproducibility
- Deployment APIs

Motivation


TensorFlow
Keras
PyTorch
scikit learn
Spark
dmlc XGBoost
ML Frameworks
Inference code
Spark
Batch & Stream Scoring
Serving tools

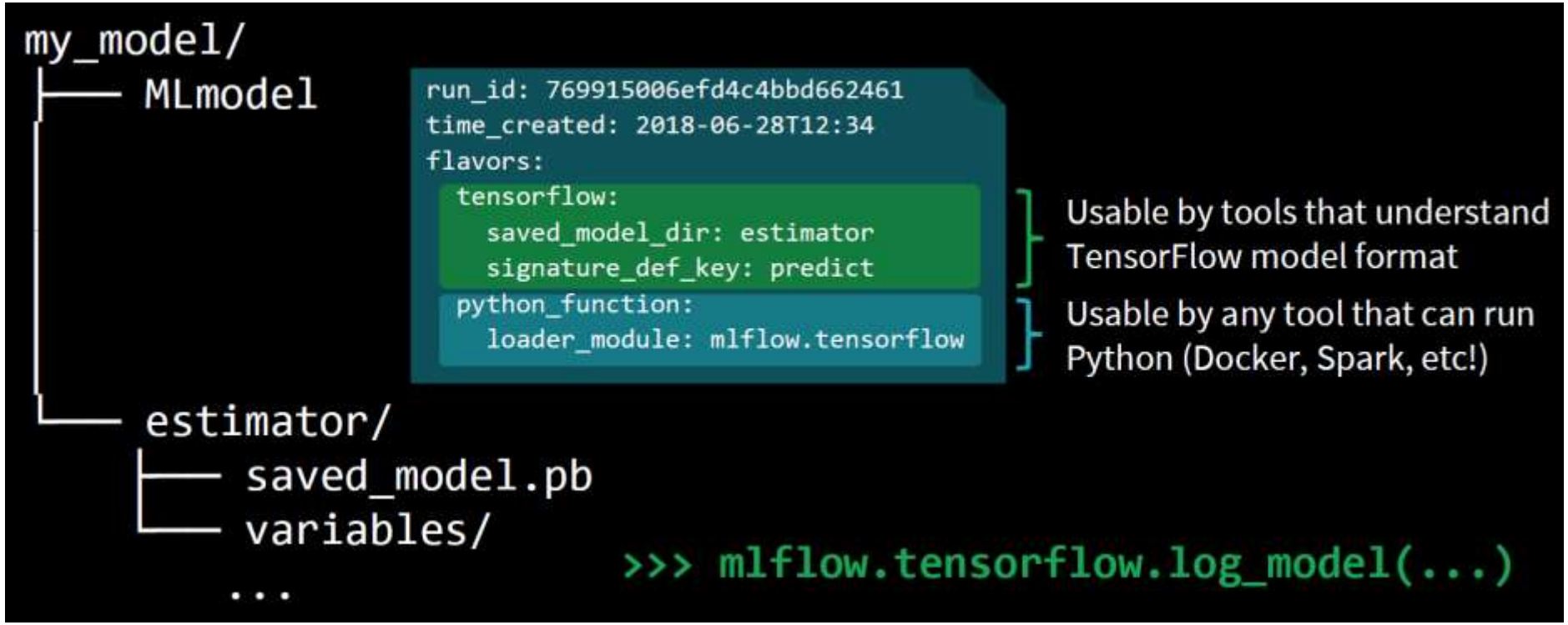
How it is working



Key concepts

- MLmodel file
- Storage format
- Entry points
- Flavours
- Custom model

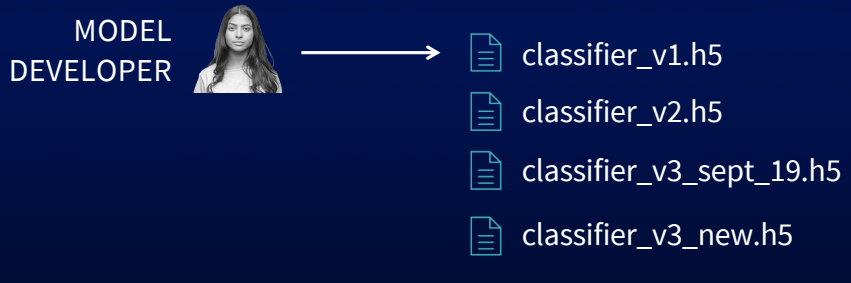
Code example



MLflow's Next Goal: Model Management

The Model Management Problem

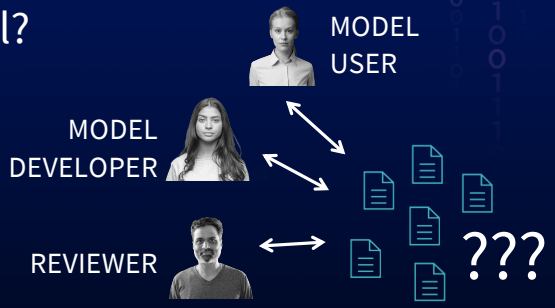
When you're working on one ML app alone, storing your models in files is manageable



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?

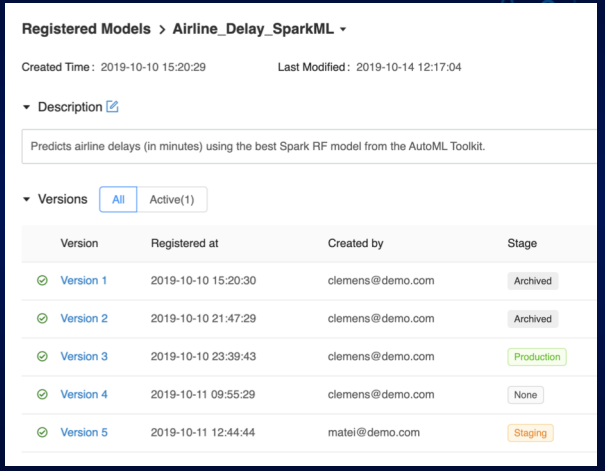


MLflow Model Registry

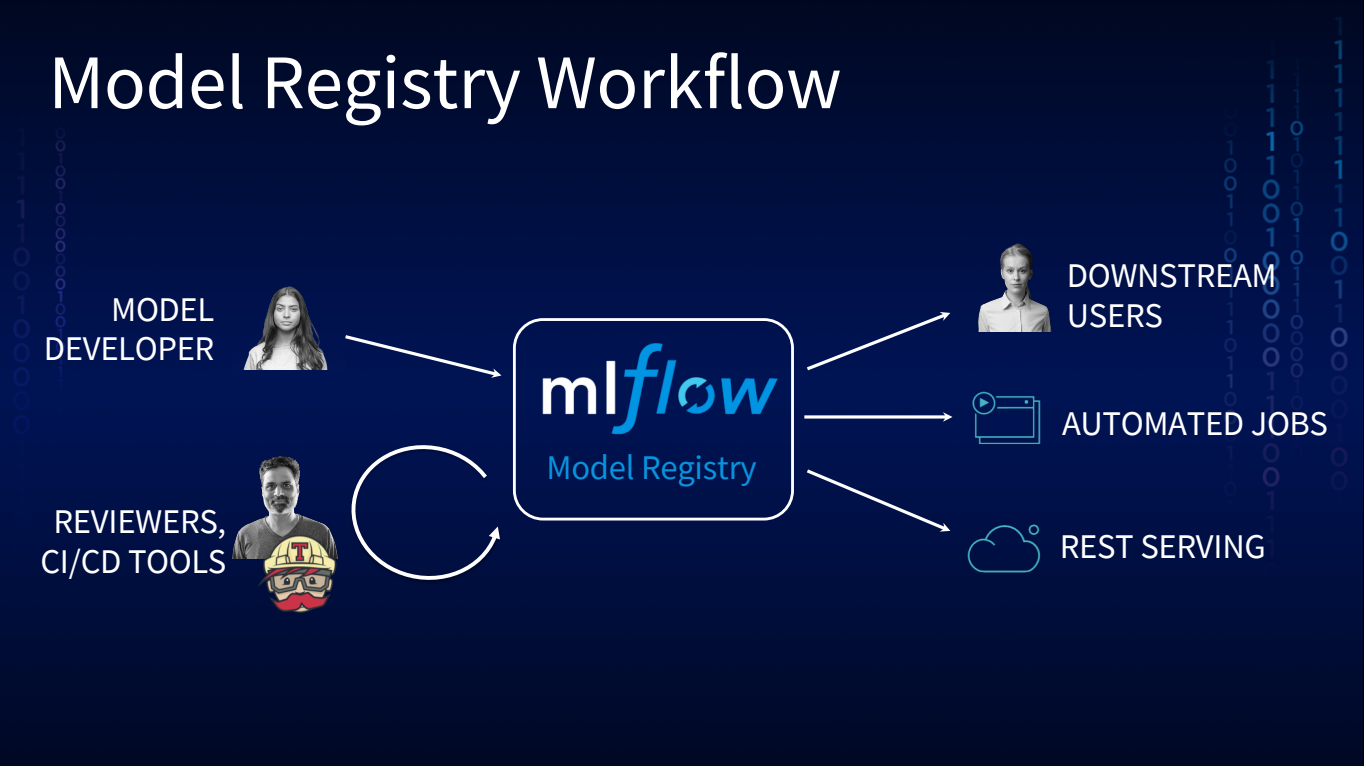
Repository of named, versioned models with comments & tags

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

Easily load a specific version



Model Registry Workflow



MLFlow rocks!

- **Log** important parameters, metrics, and other data that is important to the machine learning model
- **Track** the environment a model is run on
- **Run any** machine learning codes on that environment
- **Deploy and export** models to various platforms with multiple packaging formats