Platform for Machine Learning Lifecycle

Jules S. Damji
@2twitme

Outline – Introduction to MLflow: Model Registry Workflows Explained – Module 4

- Model Registry
- Concepts and Motivations
  - MLflow Model Registry
  - Model Registry UI & API Workflow
  - Tutorials on local host
    - Jupyter Lab
    - Google Colab
- Q & A

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

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

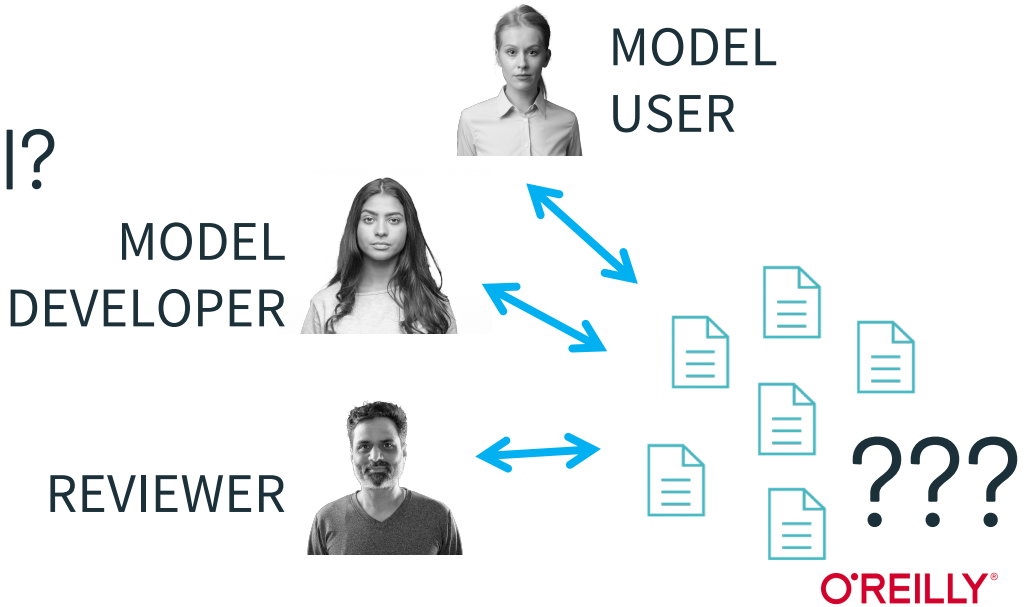
The Model Management Problem

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

The Model Management Problem

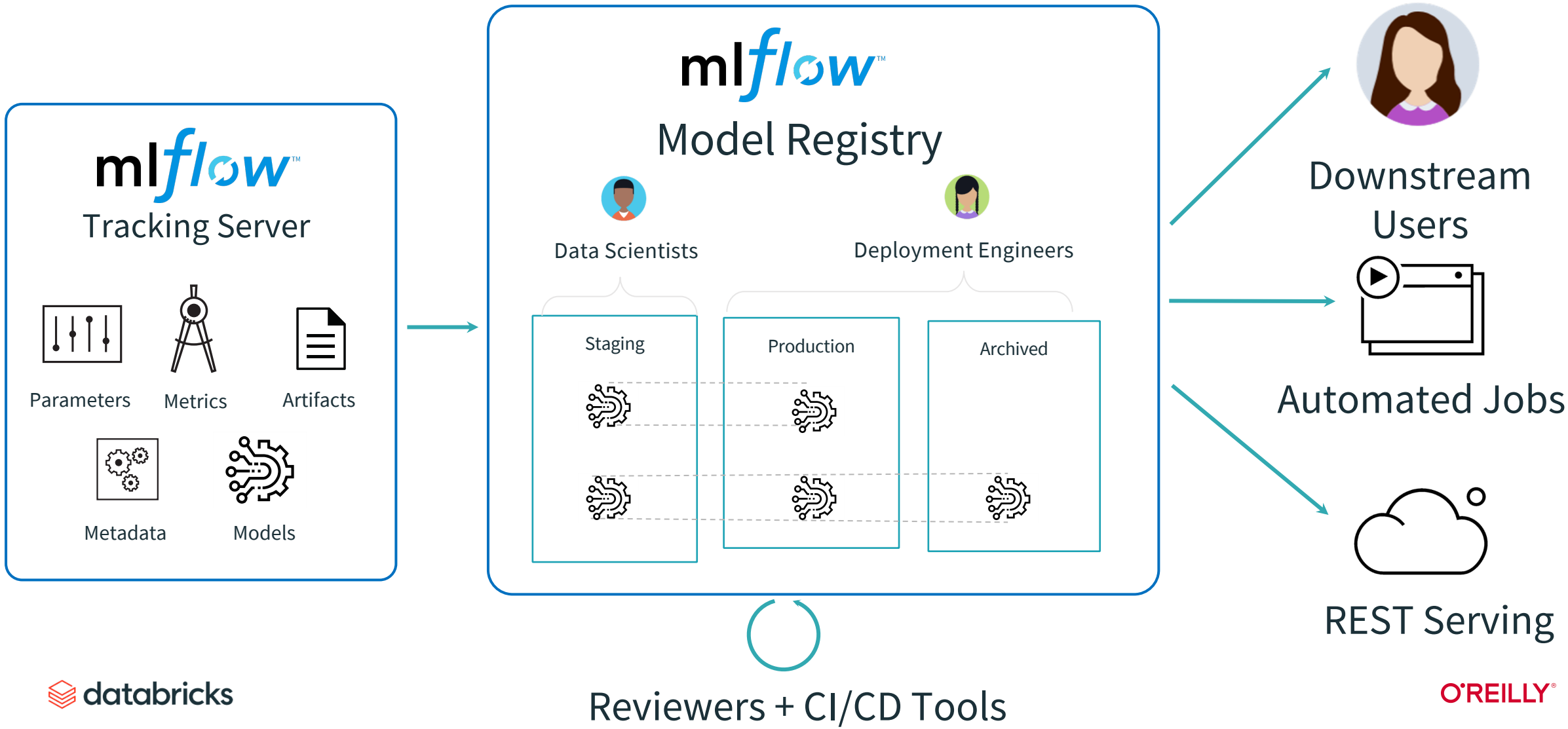
When you work in a large organization with many models, many data teams, management becomes a major challenge:

- 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?



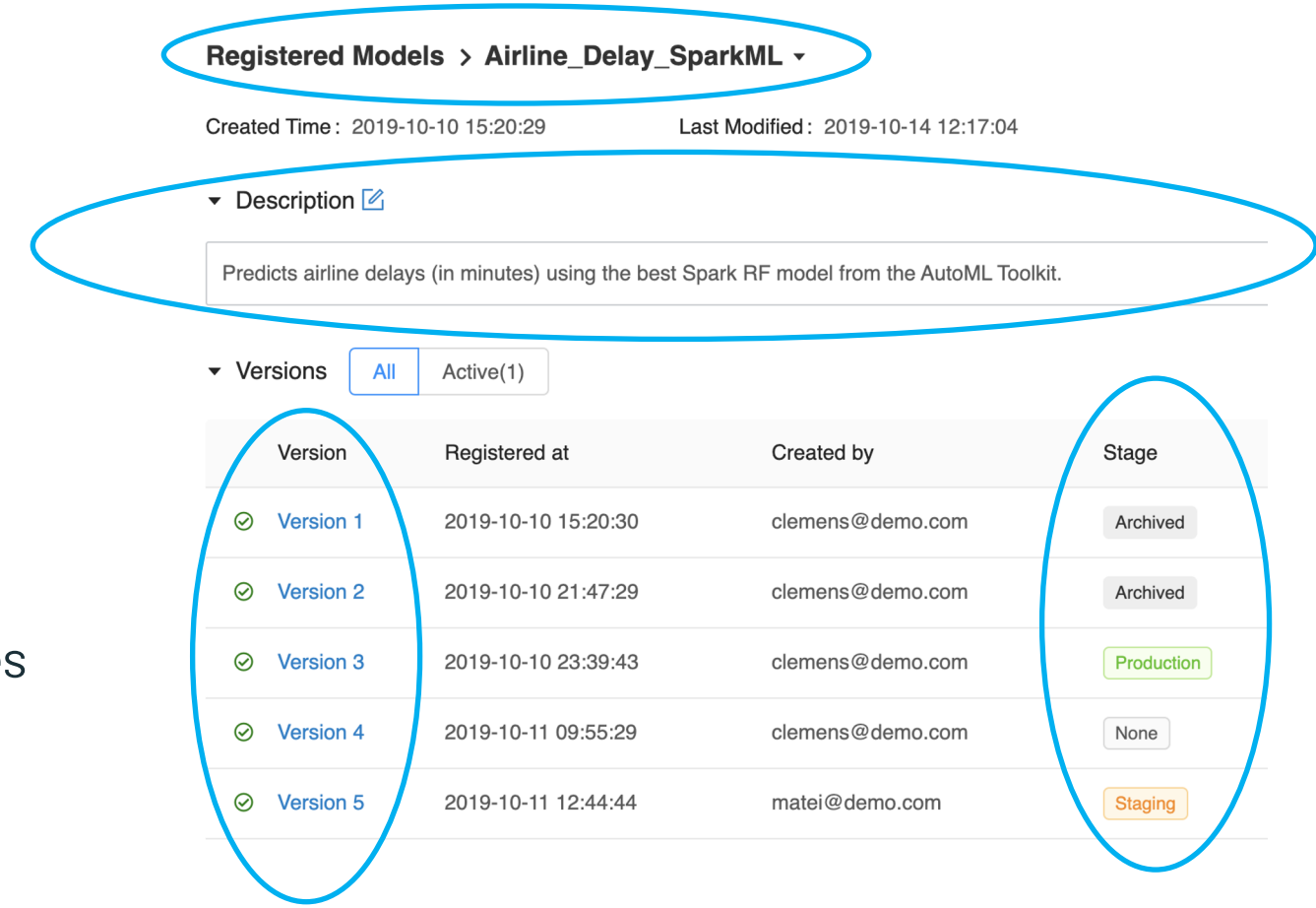
mlflow Model Registry

VISION: Centralized and collaborative model lifecycle management



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



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]

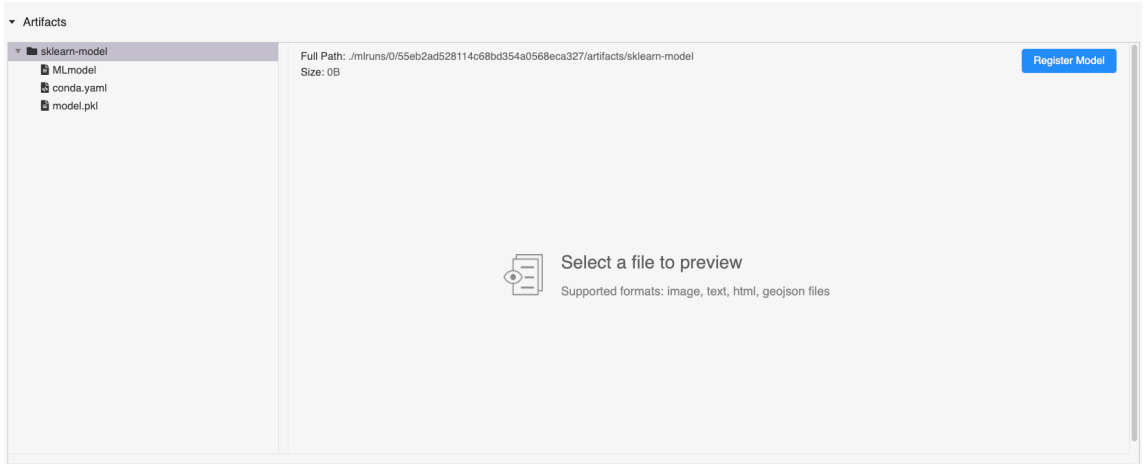
create_registered_model(*name, tags=None, description=None*) [source]

delete_model_version(*name, version*) [source]

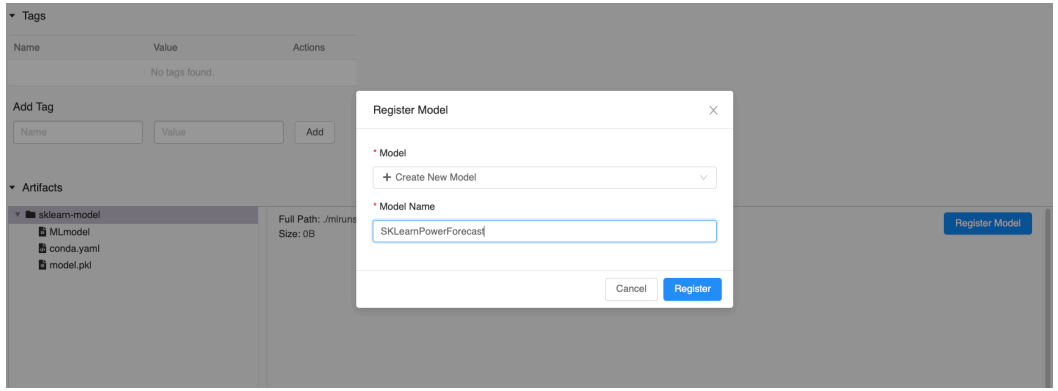
get_latest_versions(*name, stages=None*) [source]

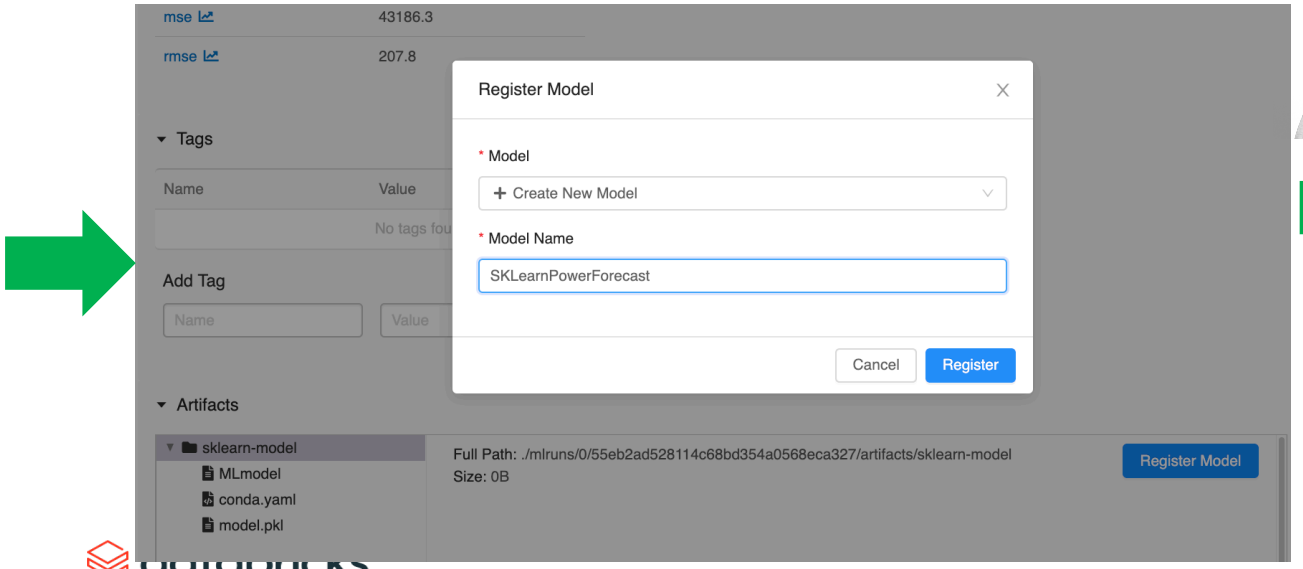
transition_model_version_stage(*name, version, stage, archive_existing_versions=False*) [source]

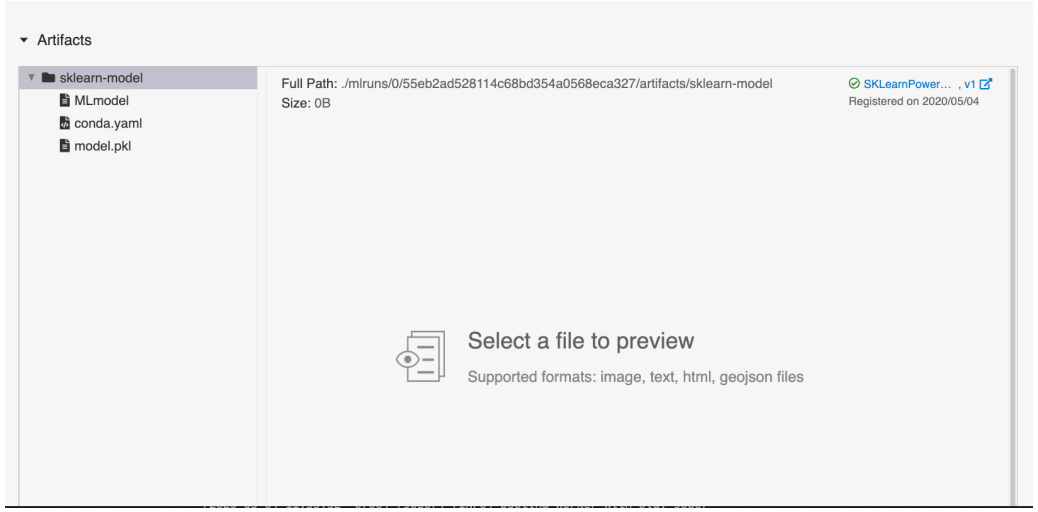
Model Registry Workflow UI



MODEL DEVELOPER



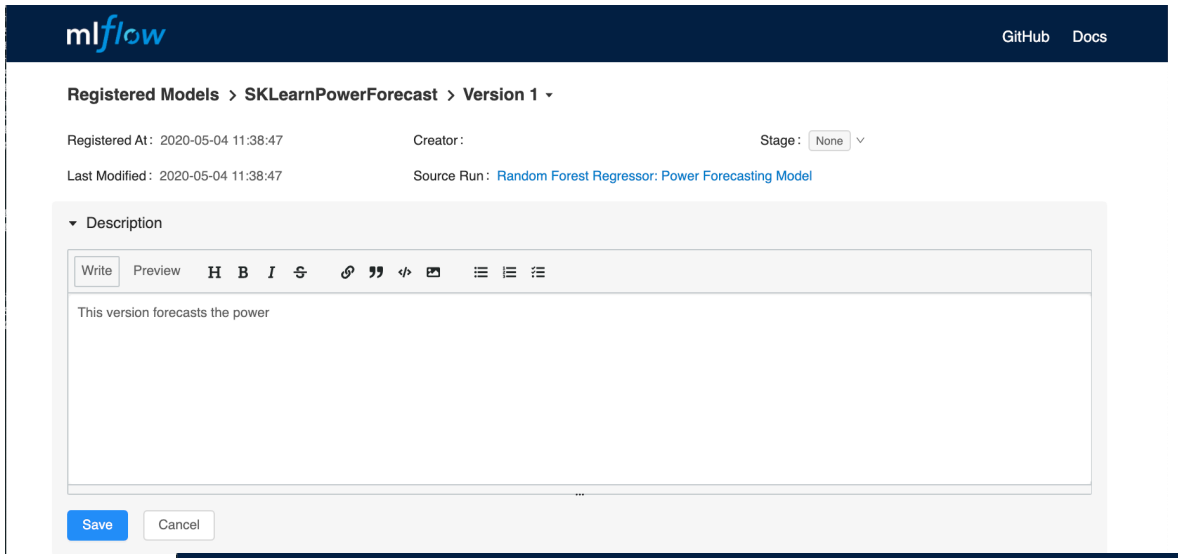




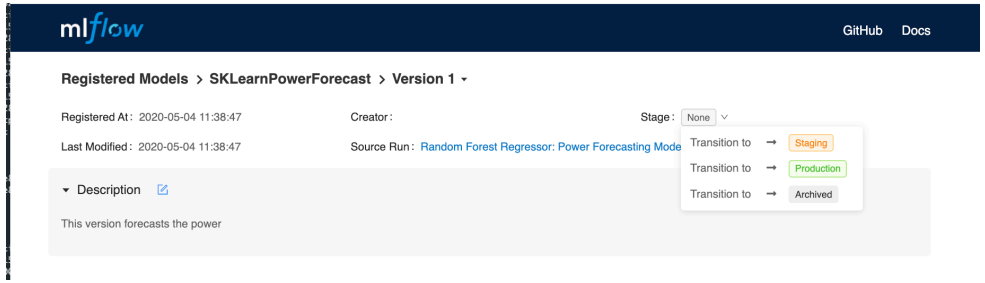
databricks

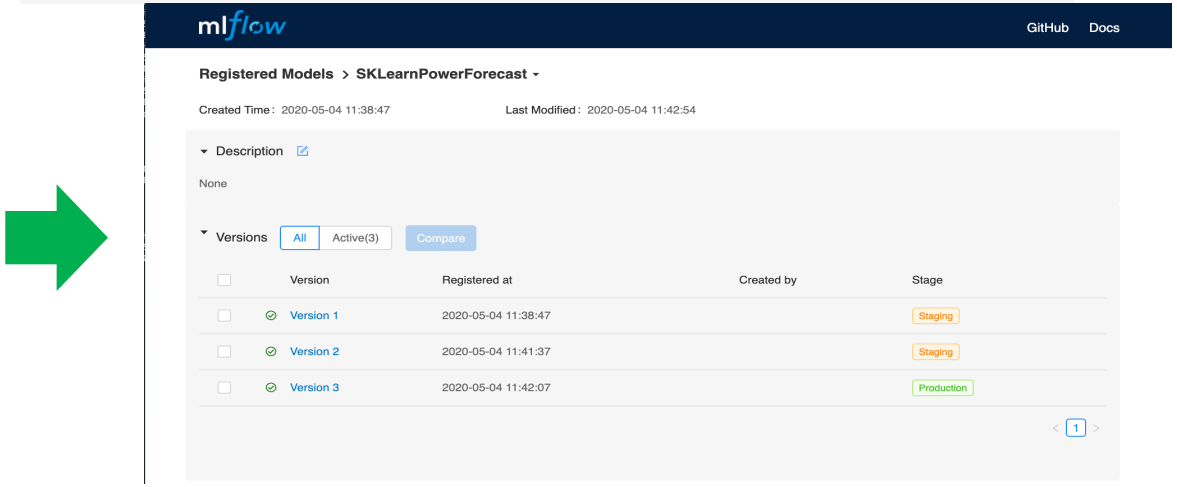
O'REILLY®

Model Registry Workflow UI



MODEL REVIEWER





DOWNSTREAM USERS

AUTOMATED JOBS

REST SERVING

databricks

O'REILLY

Model Registry Workflow API

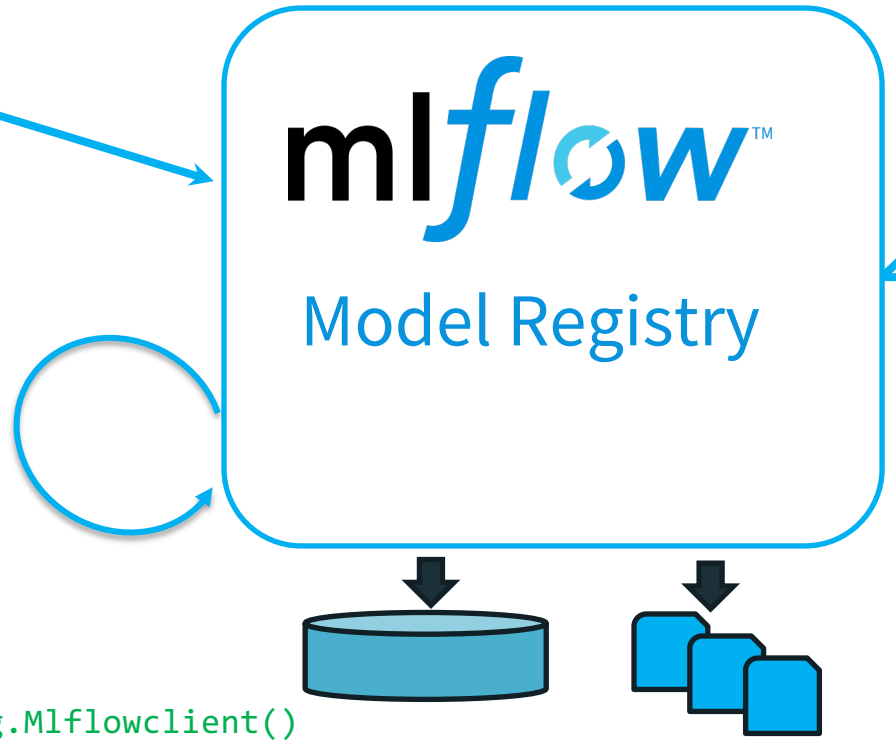
```
mlflow.register_model(model_uri,"WeatherForecastModel")

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



DOWNSTREAM USERS

AUTOMATED JOBS

REST SERVING

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

databricks

O'REILLY

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)
```

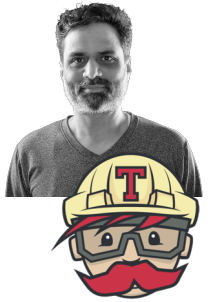
MODEL DEVELOPER

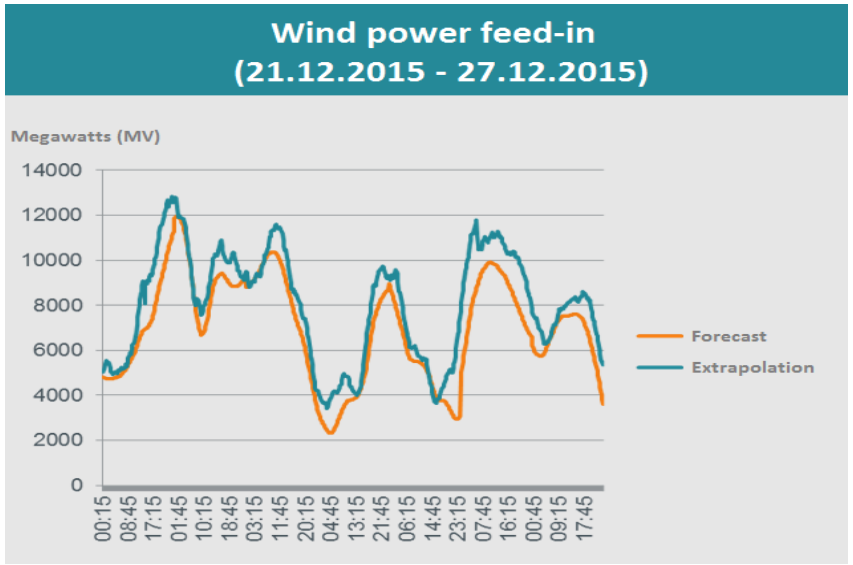
mlflow™

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 Backend Registry Stores

Entity (Metadata) Store and Models

- SQLStore (via SQLAlchemy)
  - PostgreSQL, MySQL, SQLite
  - Default is *mlruns.db* file locally
- Set programmatically for locally
- `mlflow.set_tracking_uri("sqlite:///mlruns.db")`
- `sqlite3 ./mlruns.db (on local host)`
- Managed MLflow on Databricks
  - MySQL on AWS and Azure

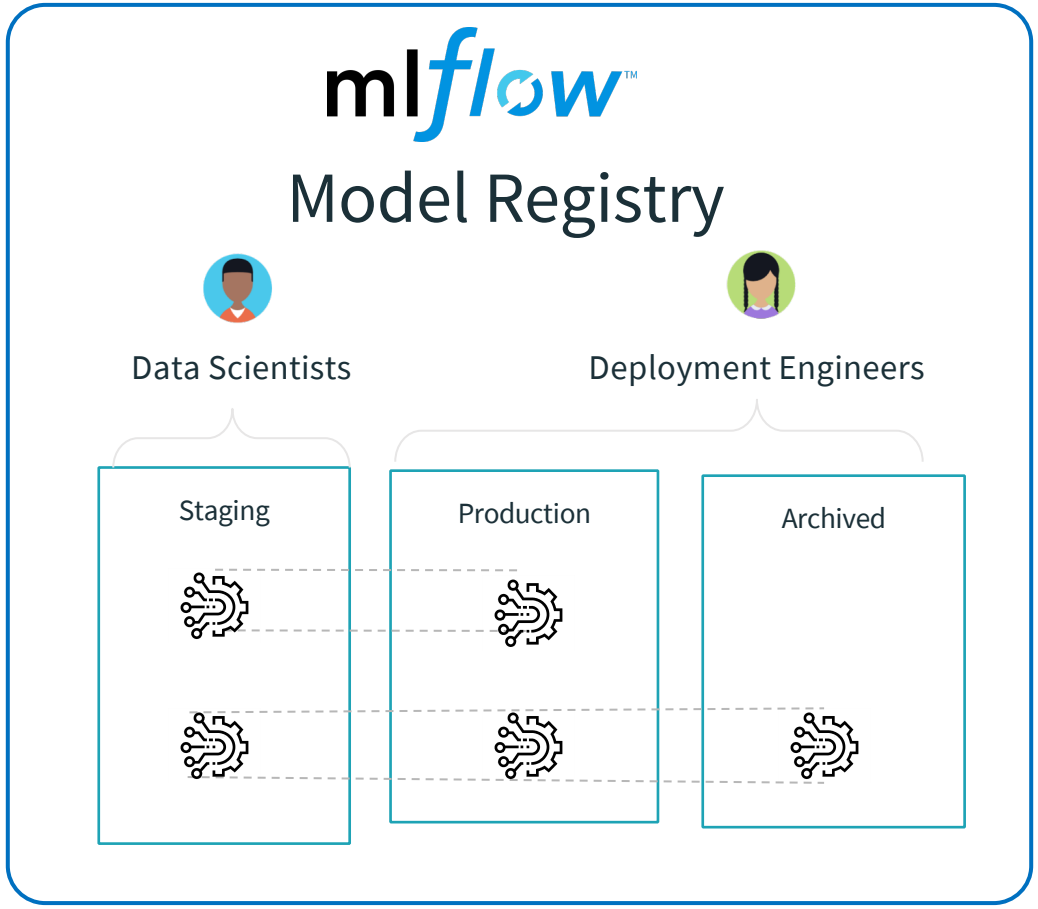
Artifact Store

- Local Filesystem
  - *mlruns* directory
- S3 backed store
- Azure Blob storage
- Google Cloud Storage
- DBFS artifact repo
- mlflow server --backend-store-uri /mnt/my-persistent-disk --default-artifact-root s3://my-mlflow-bucket/ --host 0.0.0.0

```
(tutorials) ➜  src git:(master) ✗ sqlite3 ./mlruns.db
SQLite version 3.30.1 2019-10-10 20:19:45
Enter ".help" for usage hints.
sqlite> .databases
main: /Users/julesdamji/gits/mlflow-workshop-part-3/src/./mlruns.db
sqlite> .tables
alembic_version    latest_metrics     params             tags
experiment_tags    metrics            registered_models
experiments        model_versions     runs
sqlite>
```

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

Model Registry Tutorial

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

Thank you! ☺

Q & A

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