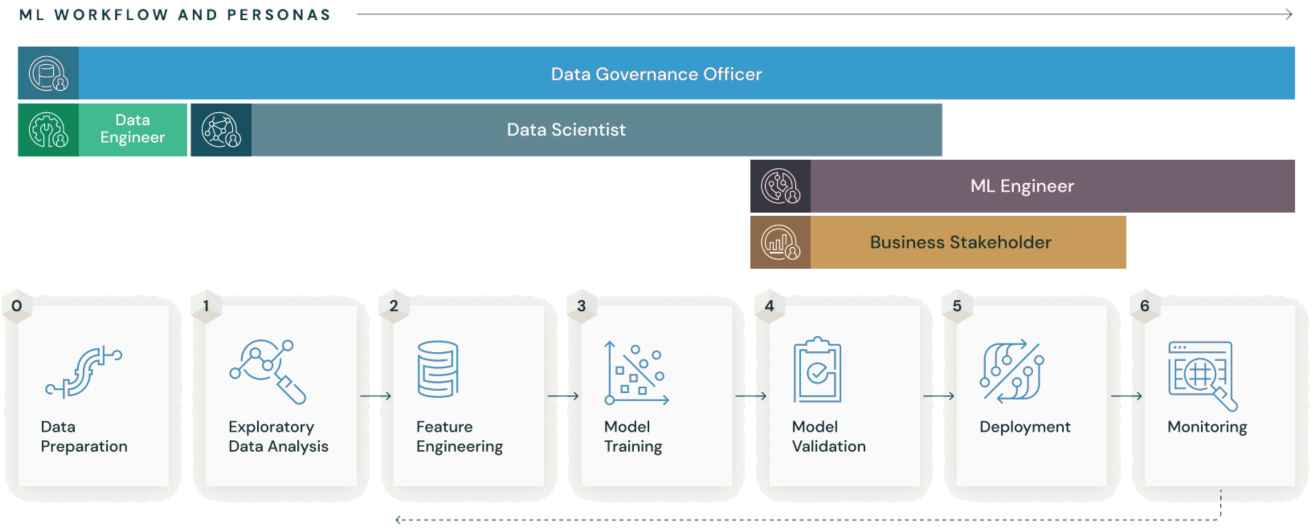
# Introduction to MLflow

**Nima Nooshiri**

16 May 2023

# What is MLflow?

- MLflow is an open source platform for **managing ML and DL projects and experiments**.
- It is specifically built to **optimise the entire model lifecycle**, i.e. from training to production.



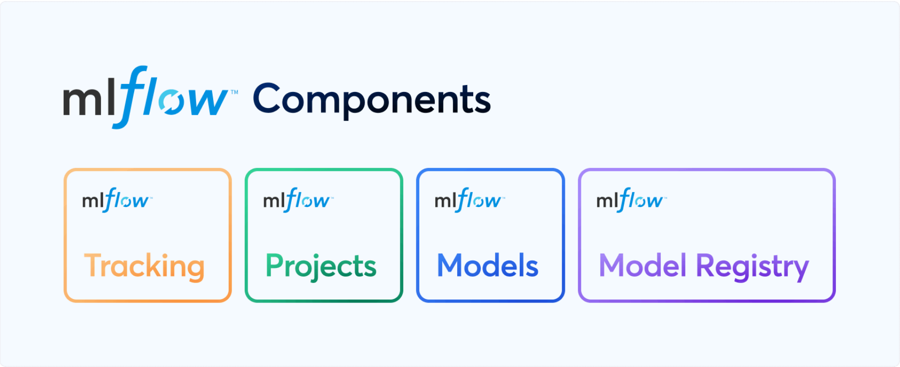
Comparison of MLflow features with some other experiment tracking and management tools (resource):

| | Neptune | Weights & Biases | Comet | Sacred & Omniboard | MLflow | TensorBoard |
|---|---|---|---|---|---|---|
| Focus | Metadata Storage, Experiment Tracking, Model Registry | Experiment Management | Experiment Management | Experiment Management | Entire Lifecycle | Experiment Management |
| Price | • Individual: Free (+ usage above free quota)<br>• Academia: Free<br>• Team: Paid | • Individual: Free (+ usage above free quota)<br>• Academia: Free<br>• Team: Paid | • Individual: Free (+ usage above free quota)<br>• Academia: Free<br>• Team: Paid | Free | Free | Free |
| Standalone component or a part of a broader ML platform? | Standalone component. ML metadata store that focuses on experiment tracking and model registry | Standalone component | Stand-alone tool with community, self-serve and managed deployment options | Omniboard is a web dashboard for the Sacred machine learning experiment management tool | Open-source platform which offers four separate components for experiment tracking, code packaging, model deploymnet, and model registry | Open source tool which is a part of the TensorFlow ecosystem |
| Commercial software, open-source software, or a managed cloud service? | Managed cloud service | Managed cloud service | Managed cloud service | Open-source | The standalone product is open-source, while the Databricks managed version is commercial | TensorBoarsd is open-source, while TensorBoard.dev is available as a free managed cloud service |
| Hosted version or deployed on-premise? | Tracking is hosted on a managed server, and can also be deployed on-premises and in a public/private cloud server | Yes | Yes, you can deploy Comet on any cloud environment or on-premise | Can be deployed both on-premises and/or on the cloud, but has to be self-managed | Tracking is hosted on a local/remote server (on-prem or cloud). Is also available on a managed server as part of the Databricks platform | TensorBoard is hosted locally. TensorBoard.dev is available on a managed server as a free service |
| How much do you have to change in your training process? | Minimal. Just a few lines of code needed for tracking | Minimal. Just a few lines of code needed for tracking | Minimal. Few lines of code needed for tracking | Minimal. Only a few lines of code need to be added | Minimal. Few lines of code needed for tracking | Minimal if already using the TensorFlow framework, else significant |
| Web UI or console-based? | Web UI | Web UI | Web UI | Web UI | Web UI | Web UI |

# MLflow Components

MLflow has four primary components:

- **Tracking:** an API and UI for logging parameters, metrics, results and so on when running ML codes.
- **Models:** to manage models from a variety of ML libraries to a variety of model serving and inference platforms.
- **Projects:** to package models in reusable and reproducible form to transfer to production.
- **Model Registry:** a centralized model store managing the full lifecycle of a trained model (model versioning, stage transitions etc).



# Concepts: MLflow Experiment and Run

## 🏃 Run

- An MLflow run corresponds to a single execution of model code.
- It is a specific ML model with a certain set of hyperparameters, corresponding metrics and results.
- Each run has a unique **Run ID**.

## 🧪 Experiment

- An MLflow experiment is the primary unit of organisation and access control for MLflow runs.
- An experiment groups together runs for a specific task (all MLflow runs belong to an experiment).
- For each experiment, the results of different runs can be analysed and compared.

## Example

- Create and set an experiment as the active experiment:

```
import mlflow
from mlflow.exceptions import MlflowException

EXPERIMENT_NAME = "IRIS Classifier Exp."

# The experiment must either be specified by name or by ID.
# The experiment name and ID cannot both be specified.

try:
    experiment_id = mlflow.create_experiment(name=EXPERIMENT_NAME)
    experiment = mlflow.get_experiment(experiment_id)
except MlflowException:
    print(f"Experiment {EXPERIMENT_NAME} already exists")
    experiment = mlflow.get_experiment_by_name(EXPERIMENT_NAME)

mlflow.set_experiment(experiment_name=experiment.name)
```

- Run an experiment to train a model:

```
with mlflow.start_run() as run:
    print(f"Starting run with 'run_id': {run.info.run_id}")

    # Do the actual model training
    # ...
```

# MLflow Stores

## 🪣 Where runs are recorded?

MLflow runs can be recorded:

- to local files,
- to a database,
- remotely to a tracking server.

## How runs and artifacts are recorded?

MLflow uses two components for storage:

1. **Backend store**: persists MLflow entities (runs, parameters, metrics etc).
   - Supported types: *file store* and *database-backed store*.
2. **Artifact store**: persists artifacts (files, models, images, model summary etc).
   - Supported types: *local file paths* and *storage systems* (e.g., AWS S3, GCS)

# MLflow Storage Configuration

## Common scenarios

1. MLflow on **localhost**:
   - Both the backend and artifact stores share a directory on the local filesystem.

2. MLflow on **localhost with SQLite**:
   - Artifacts are stored in a directory on the local filesystem, and MLflow entities are inserted in a SQLite database.

3. MLflow on **localhost with Tracking Server**:
   - Similar to scenario 1 but a tracking server is launched, listening for REST request calls.

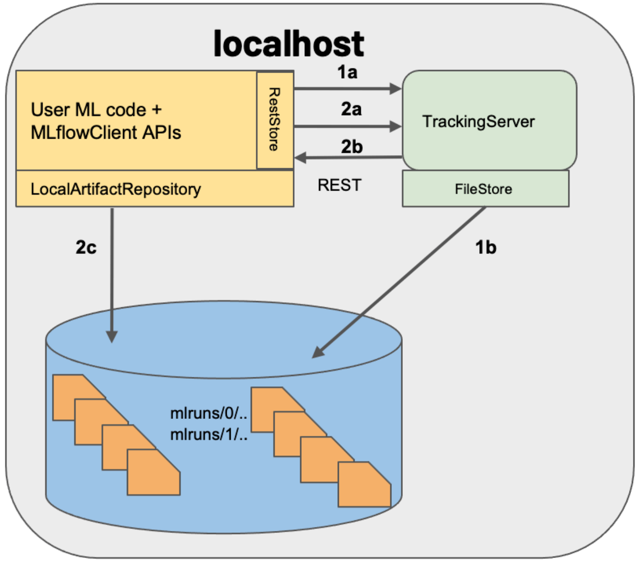
4. MLflow with **remote Tracking Server**, backend and artifact stores:
   - The tracking server, backend store, and artifact store reside on remote hosts.

Nima Nooshiri: nima.nooshiri@gmail.com

## Example

- Use a SQLite database file as backend store,
- Use a local directory as artifact store,
- Set a tracking server to log runs remotely.

```
$ mlflow server \
    --backend-store-uri="sqlite:///my_database.db" \
    --default-artifact-root="./my_mlruns/" \
    --host="0.0.0.0" \
    --port="5000"
```



# Log Data to Runs

## Basic things to track

- **Parameters**: key-value input parameters using `log_param()` or `log_params()`.
- **Metrics**: key-value metrics using `log_metric()` or `log_metrics()`
- **Artifacts**: local files or directories as artifacts using `log_artifact()` or `log_artifacts()`.

## Example

```
with mlflow.start_run() as run:
    print(f"Starting run with 'run_id': {run.info.run_id}")

    # Log the hyperparameters
    mlflow.log_params(train_params: dict[str: Any])

    # Do the actual model training
    # ...

    # Log the train and test accuracy
    mlflow.log_metric('train_acc', train_acc)
    mlflow.log_metric('test_acc', test_acc)
```

# MLflow Model API

MLflow includes integrations with several common libraries.

## Built-In Model Flavors

- PyTorch
- Keras
- TensorFlow
- Scikit-learn
- XGBoost
- ...

## Key functions

- `mlflow.<model-flavor>.save_model()`: to save the model to a local directory.
- `mlflow.<model-flavor>.log_model()`: to log the model as an artifact in the current run using MLflow Tracking.
- `mlflow.<model-flavor>.load_model()`: to load a model from a local directory or from an artifact in a previous run.

## Example

```python
with mlflow.start_run() as run:
    print(f"Starting run with 'run_id': {run.info.run_id}")

    # Log the hyperparameters
    # ...

    # Do the actual model training
    # ...

    # Log the train and test accuracy
    # ...

    # Log the trained model (called "logged model")
    mlflow.pytorch.log_model(pytorch_model=my_model, artifact_path='pytorch_models')
```

# MLflow Model Registry Concept

## Model

- An MLflow Model is created from an experiment or run that is logged with one of the model flavor’s methods: `mlflow.<model-flavor>.log_model()`.
- Once logged, this model can then be registered with the Model Registry.

## Registered Model

- An MLflow Model can be registered with the Model Registry.
- A registered model has a unique name, contains versions, associated transitional stages, model lineage, and other metadata.

Nima Nooshiri: nima.nooshiri@gmail.com

# Register a Model

There are three ways to add a model to the registry:

1. Using the `mlflow.<model-flavor>.log_model(..., registered_model_name=<name>)` method.
2. Using the `mlflow.register_model()` method.
3. Using the `create_registered_model()` method.

### Example

```
import mlflow
from mlflow.tracking import MlflowClient
from mlflow.exceptions import RestException

MODEL_URI = "runs:/3f2a871bfe944121a2372586ec4ae17f/pytorch_models"
REGISTERED_MODEL_NAME = "IRIS_CLF_MLP"

# Create registered model (registery point)
client = MlflowClient()

try:
    registered_model = client.create_registered_model(REGISTERED_MODEL_NAME)
except RestException:
    print(f"Model already exists in the registry: '{REGISTERED_MODEL_NAME}'")

# Register experiment run to that model
model_version = mlflow.register_model(MODEL_URI, REGISTERED_MODEL_NAME)
```

## Load a Model

- To load a previously logged model for inference or development, use `mlflow.<model-flavour>.load_model(model_uri)`.

- The location of the model, `model_uri`, can be one of the following:
  - A run-relative path: `runs:/<run-id>/<run-relative-path-to-model>`,
  - A registered-model path: `models:/<model-name>/<model-version>` or `models:/<model-name>/<stage>`.

# Example: Classification with PyTorch and MLflow

This example demonstrates training a classification model on the IRIS dataset, creating the model with PyTorch, logging the model to MLflow, registering the trained model, and loading it back for inference.

### Step 1: Set up the tracking server

**NOTE:** In order to use model registry functionality, we must run the server using a database-backed backend store.

```
$ mlflow server \
    --backend-store-uri="sqlite:///mlflow.db" \
    --default-artifact-root="./mlruns/" \
    --host="0.0.0.0" \
    --port="5000"
```

Now, we are able to see the tracking URI in the browser at http://0.0.0.0:5000.

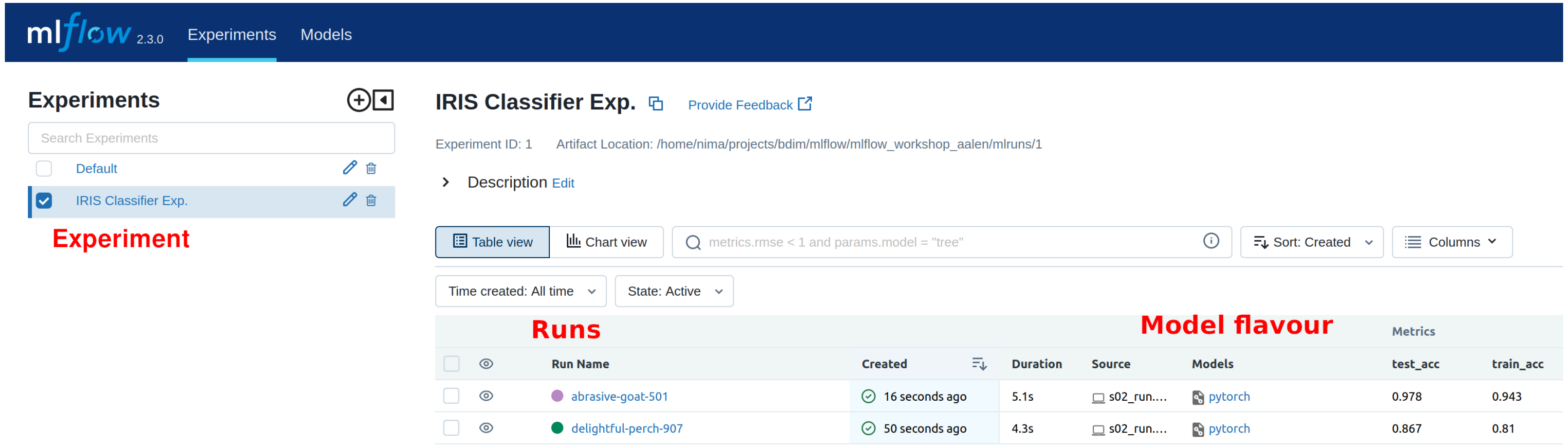
## Step 2: Train and Log the Model

To start a run to train a model and then log the trained model:

```
$ python s02_train_and_log.py --lr=0.005 --dropout-prob=0.1 --num-epochs=150
```

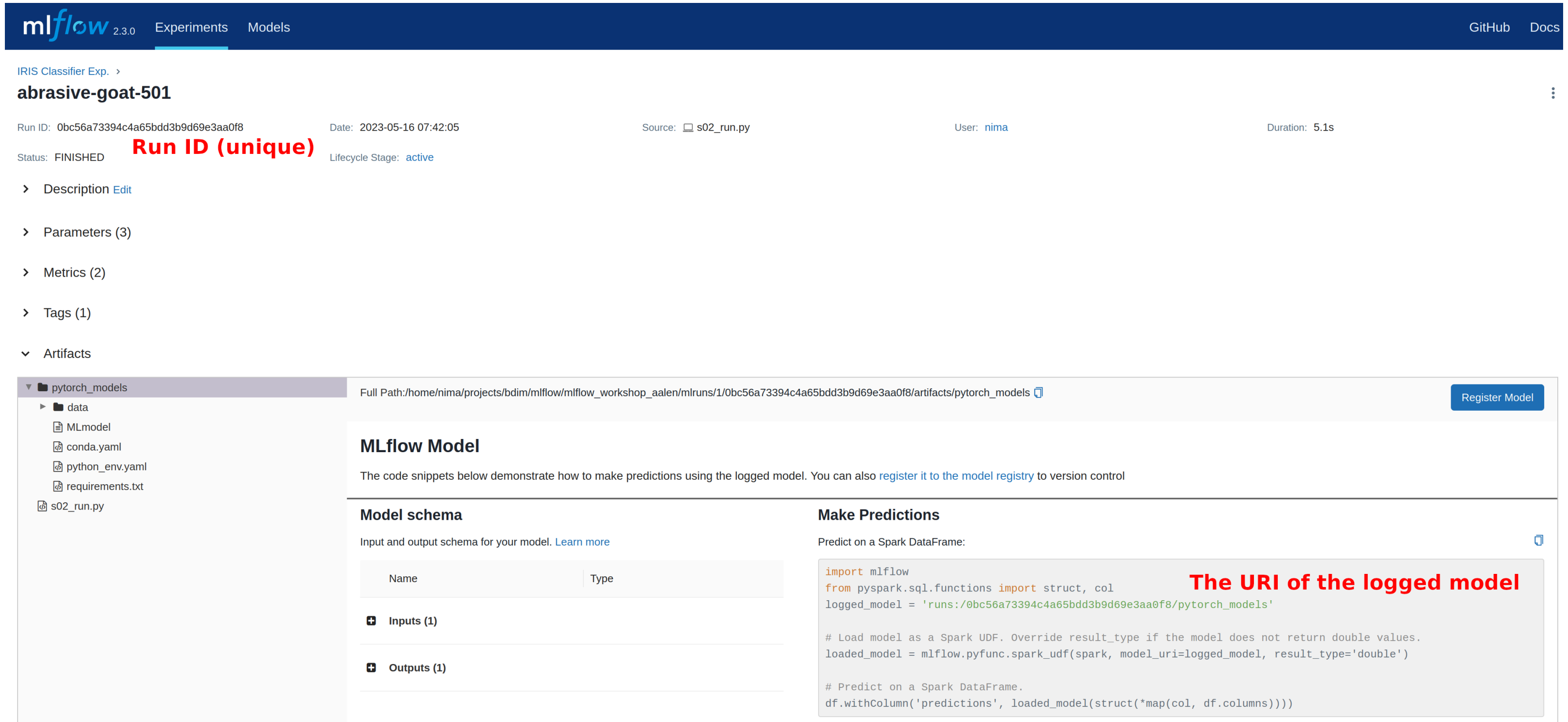
The tracking URI and the experiment name are already set (hard-coded) in the file.
To use a different tracking URI and/or change the experiment name, they can be changed in the python file:

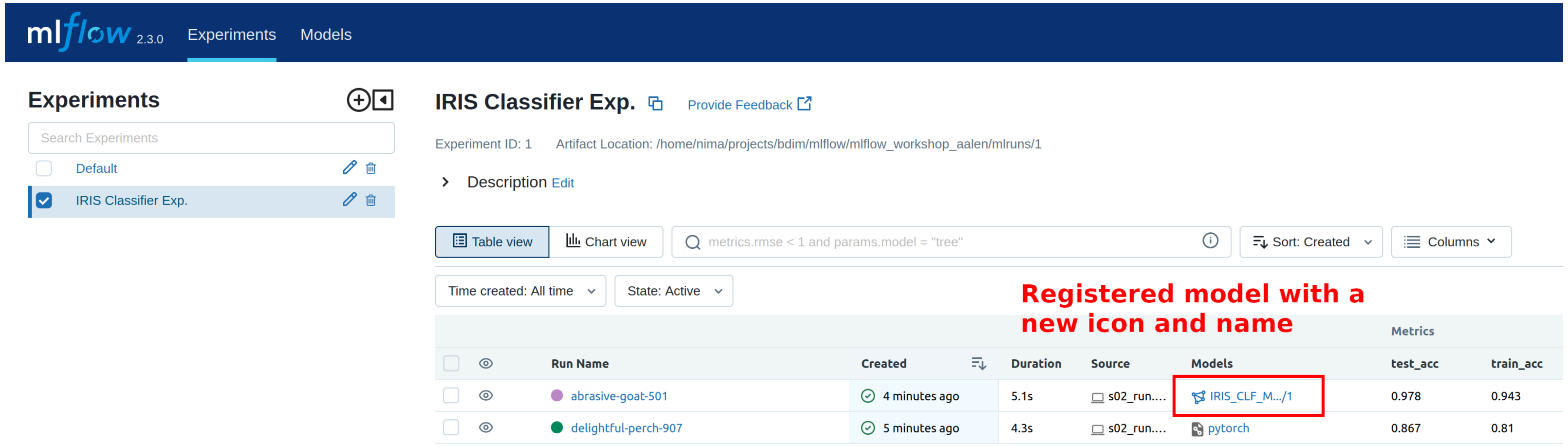
```
TRACKING_SERVER_URI = "http://0.0.0.0:5000"
EXPERIMENT_NAME = "Classification_Experiment"
```

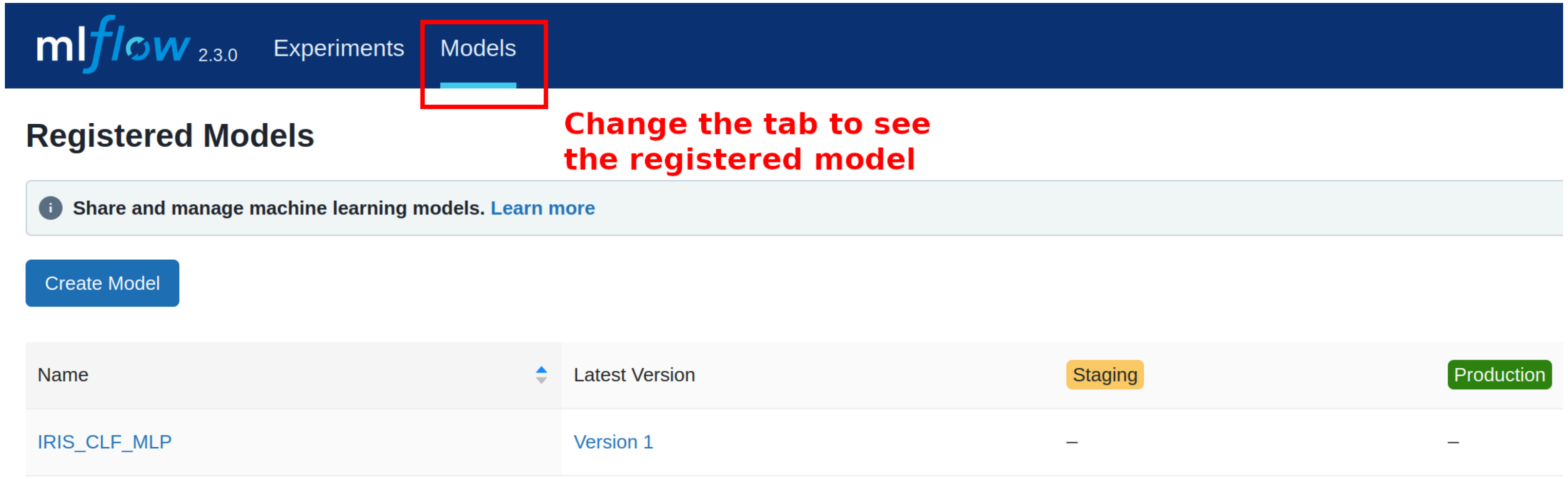


## Step 3: Register the Logged Model

```
$ python s03_register \
    --logged-model-uri='runs:/0bc56a73394c4a65bdd3b9d69e3aa0f8/pytorch_models' \
    --registered-model-name='MLP_Classifier'
```




mlflow 2.3.0
Experiments
Models
Experiments
Search Experiments
Default
IRIS Classifier Exp.
IRIS Classifier Exp.
Provide Feedback
Experiment ID: 1
Artifact Location: /home/nima/projects/bdim/mlflow/mlflow_workshop_aalen/mlruns/1
Description Edit
Table view
Chart view
metrics.rmse < 1 and params.model = "tree"
Sort: Created
Columns
Time created: All time
State: Active
Registered model with a
new icon and name
Metrics
Run Name
Created
Duration
Source
Models
test_acc
train_acc
abrasive-goat-501
4 minutes ago
5.1s
s02_run....
IRIS_CLF_M.../1
0.978
0.943
delightful-perch-907
5 minutes ago
4.3s
s02_run....
pytorch
0.867
0.81


mlflow 2.3.0
Experiments
Models
Change the tab to see
the registered model
Registered Models
Share and manage machine learning models. Learn more
Create Model
Name
Latest Version
Staging
Production
IRIS_CLF_MLP
Version 1
–
–

### Step 4: Use the Model for Inference

Now, we can select a model, load it and make a prediction.

```
# Use model version 1
$ python s04_predict --logged-model-uri="models:/MLP_Classifier/1"
```

We can also specify the model **tag** instead of model version. For example, if we want to use a model with tag `champion`, we need to replace `/1` with `@champion` in the model URI above.

## 📚 Resources

- MLflow documentation.
- MLflow examples.
- Neptune.ai blog