
FIND YOUR WAY TO MLFLOW
WITHOUT CONFUSION
MARVELous
MLOps
mlflow™

MARVELous MLOps

Log & load objects with MLflow

	Log	Load

log_param
Log a parameter under the current run
	`mlflow.log_param("learning_rate", 0.01)`	`run_id = '5f871c4f04e04dc295f5c77'`
		`mlflow.get_run(run_id=f'{run_id}').`
		`to_dictionary()['data']['params']`

log_params Logs multiple params under the current run	`mlflow.log_params({"learning_rate", 0.01,` `"n_estimators": 10})`	

log_metric
Log a metric under the current run
	`mlflow.log_metric("mse", 2500.00)`	`run_id = '5f871c4f04e04dc295f5c77'`
		`mlflow.get_run(run_id=f'{run_id}').`
		`to_dictionary()['data']['metrics]`

log_metrics Logs multiple metrics under the current run	`mlflow.log_metrics({"mse": 2500.00,` `"rmse": 50.00})`	
log_artifact Log a local file as an artifact of the current run	`mlflow.log_artifact("features.txt")`	N/A
log_artifacts Log contents of a local folder as artifacts of the current run	`mlflow.log_artifacts("demo",` `artifact_path="demo")`	
log_dict Log a JSON/YAML-serializable object as an artifact	`mlflow.log_dict({"k": "v"}, "data.json")`	`mlflow.artifacts.load_dict(` `'runs:/5f871c4f04e04dc295f5c77/data.json')`
log_text Log text as an artifact	`mlflow.log_text("text1", "file1.txt")`	`mlflow.artifacts.load_text(` `'runs:/5f871c4f04e04dc295f5c77/file1.txt')`

log_figure
Log a figure as an artifact
	`import matplotlib.pyplot as plt`
	`fig, ax = plt.subplots()`
	`ax.plot([0, 1], [2, 3])`
	`mlflow.log_figure(fig, "figure.png")`
		`mlflow.artifacts.load_image(`
		`'runs:/5f871c4f04e04dc295f5c77/figure.png')`

log_image
Log an image as an artifact
	`from PIL import Image`
	`image = Image.new("RGB", (100, 100))`
	`log_image(image, "image.png")`
		`mlflow.artifacts.load_image(`
		`'runs:/5f871c4f04e04dc295f5c77/image.png')`

MARVELous MLOps

MLflow: log & load a custom model with pyfunc model flavor

1. Define the custom model class CustomIrisModel & the wrapper

```python
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import mlflow
import numpy as np


class CustomIrisModel:
    def __init__(self, params):
        self.flower_classes = ["setosa", "versicolor", "virginica"]
        self.model = LogisticRegression(**params)

    def train(self, X, y):
        self.model.fit(X, y)
        return self

    def predict(self, model_input):
        predictions = self.model.predict(model_input)
        return np.array([self.flower_classes[x] for x in predictions])


class CustomPredict(mlflow.pyfunc.PythonModel):
    def __init__(self, model):
        self.model = model  Custom model

    def predict(self, context, model_input: dict):
        return self.model.predict(model_input)
```

2. Log the wrapped model

```python
with mlflow.start_run(run_name="iris") as run:
    X, y = load_iris(return_X_y=True, as_frame=True)
    classifier = CustomIrisModel(params={"C": 1.0, "random_state": 42}).train(X, y)
    wrapped_model = CustomPredict(classifier)
    mlflow.pyfunc.log_model("model", python_model=wrapped_model)
```

3. Load the original custom model (CustomIrisModel)

```python
mlflow.pyfunc.load_model("runs:/70801eb17462c5b8d0e00/model").unwrap_python_model().model
```

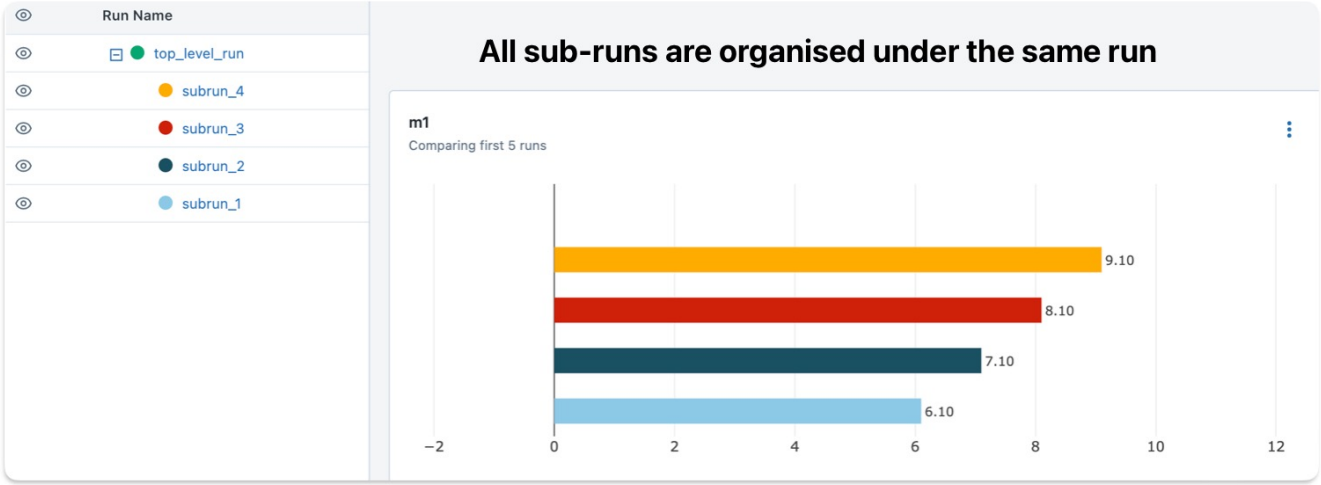
```
<__main__.CustomIrisModel at 0x7fd275cedf60>
```

MARVELous MLOps

MLflow parent & child runs

It is possible to create child experiment runs under the parent run by providing nested=True argument

```
import mlflow
with mlflow.start_run(run_name="top_level_run") as run:
    for i in range(1,5):
        with mlflow.start_run(run_name=f"subrun_{str(i)}", nested=True) as subrun1:
            mlflow.log_param("subrun_number", f"{str(i)}")
            mlflow.log_metric("m1", 5.1+i)
            mlflow.log_metric("m2", 5.1*i)
            mlflow.log_metric("m3", 5.1/i)
```



You can find all child runs by filtering on parent run id

```
mlflow.search_runs(search_all_experiments=True, filter_string='tags.mlflow.parentRunId="25b884a1be3e425bbb08764e6b243d11"')
```

	run_id	experiment_id	status	artifact_uri	start_time	end_time	metrics.m1	metrics.m3
0	12abff8b993d46ab9334798363f4e518	2536120347857321	FINISHED	dbfs:/databricks/mlflow-tracking/2536120347857...	2023-09-10 11:04:21.554000+00:00	2023-09-10 11:04:22.253000+00:00	9.1	1.275
1	dd19190cb29c415ca2276234364a86e9	2536120347857321	FINISHED	dbfs:/databricks/mlflow-tracking/2536120347857...	2023-09-10 11:04:20.718000+00:00	2023-09-10 11:04:21.386000+00:00	8.1	1.700
2	3eacfe76ab144de9a1e677d34e717b04	2536120347857321	FINISHED	dbfs:/databricks/mlflow-tracking/2536120347857...	2023-09-10 11:04:19.884000+00:00	2023-09-10 11:04:20.572000+00:00	7.1	2.550
3	7b4a74d4780d4e9cbdb1a010752a1498	2536120347857321	FINISHED	dbfs:/databricks/mlflow-tracking/2536120347857...	2023-09-10 11:04:19.040000+00:00	2023-09-10 11:04:19.728000+00:00	6.1	5.100

MARVELous MLOps

Searching MLflow experiment runs

Logical operators: AND

Comparators for numeric attributes: =, !=, <, <=, >, >=

Comparators for string attributes and tags:

- =, !=
- LIKE: Case-sensitive pattern match
- ILIKE: Case-insensitive pattern match

```
import mlflow
mlflow.search_runs(
    # can be searched by experiment ids, or experiment names, or through all experiments
    search_all_experiments=True,  # or: experiment_ids=[] | or: experiment_names=[]

    run_view_type=3,  # ACTIVE_ONLY= 1, DELETED_ONLY= 2, ALL= 3

    # can be ordered by start_time, status, metrics, params, tags; ordering ASC or DESC
    order_by=["start_time DESC",
              "status DESC",
              "metrics.metric1 ASC",
              "metrics.metric2 ASC",
              "params.demo DESC",
              "tags.mlflow.databricks.notebookID ASC"],

    # can be filtered by: status, start_time, artifact_uri, runId, run_name, metrics, params, tags
    # multiple filters can be combined by using "AND" clause
    filter_string="status='FINISHED' AND "
                  "start_time>1664067852 AND "
                  "artifact_uri LIKE '%140279%' AND "
                  "run_id IN ('5f871c4f04e04dc295f5c771093585fd', '75dec62act4041b58abb5c494b98e1') AND "
                  "run_name LIKE '%demo-run%' AND "
                  "metrics.metric1>0 AND "
                  "params.demo='yes' AND "
                  "tags.mlflow.source.type!='JOB'"
                  )
```

Timestamp is accepted in unix format

```
from datetime import datetime
stime = "2022-09-25 01:04:12"
# unix_timestamp
datetime.strptime(stime, '%Y-%m-%d %H:%M:%S').timestamp()
```

Transforming date & time to unix format

MARVELous MLOps

Search MLflow model by run id

```python
import mlflow

client = mlflow.MlflowClient()


def get_model_version_by_run_id(run_id):
    for model in client.search_model_versions(
            filter_string="name='demo-model'"):
        dbr_run_id = client.get_run(
            model.run_id).data.to_dictionary()['tags'].get(
            'mlflow.databricks.jobRunID', "None")
        if dbr_run_id == run_id:
            return model.version


get_model_version_by_run_id(run_id='902422')
```

Option 1

To get a model version by Databricks run id:

- Search through model versions
- Retrieve their MLFlow run id
- Retrieve Databricks run id associated with MLFlow run id

Limitation: only possible to search for task run id!

```python
def get_arguments():
    parser = argparse.ArgumentParser()
    parser.add_argument('--run_id', metavar='run_id',
                        type=str, help='Databricks run id')
    parser.add_argument('--job_id', metavar='job_id',
                        type=str, help='Databricks job id')
    args = parser.parse_args()
    return args.run_id, args.job_id


run_id, job_id = get_arguments()

mlflow_run_id = mlflow.active_run().info.run_id
model_version1 = mlflow.register_model(
    model_uri=f"runs:/{mlflow_run_id}/model",
    name='demo-model',
    tags={"dbr_run_id": run_id, "dbr_job_id": job_id})
```

Option 2

To get a model version by Databricks run id:

- Pass run id and job id as python parameters {{job_id}} and {{run_id}}
- When registering a model, pass tags including run_id and job_id
- Search model by tag

Possible to pass task run id & parent run id as a tag

```python
model_version = client.search_model_versions(
    f"name='demo-model' and tag.dbr_run_id = '902422'")[0].version
```

MARVELous MLOps

Retrieve Databricks run & job id in your Python code

demo.py

```python
def get_arguments():
    parser = argparse.ArgumentParser(description='reads default arguments')
    parser.add_argument('--run_id', metavar='run_id', type=str, help='Databricks run id')
    parser.add_argument('--job_id', metavar='job_id', type=str, help='Databricks job id')
    args = parser.parse_args()
    return args.run_id, args.job_id


run_id, job_id = get_arguments()
```

dbx_deployment.yml

```yaml
build:
  python: "pip"

environments:
  default:
    workflows:
      - name: "demo_workflow"
        job_clusters:
          - job_cluster_key: "demo_cluster"
            new_cluster:
              spark_version: "12.2.x-cpu-ml-scala2.12"
              num_workers: 1
              node_type_id: "Standard_D4s_v5"
        tasks:
          - task_key: "demo_task"
            job_cluster_key: "demo_cluster"
            spark_python_task:
              python_file: "file://demo.py"
              parameters: ["--run_id", "{{parent_run_id}}", "--job_id", "{{job_id}}"]
```

MARVELous MLOps

MLflow experiment tracking and model registry are two different things!

When the experiment run has started, you can **log a model in 2 ways**:

➡ mlflow.log_artifact

➡ mlflow.<model_flavor>.log_model

After that, you can find your model artifact under the experiment run.

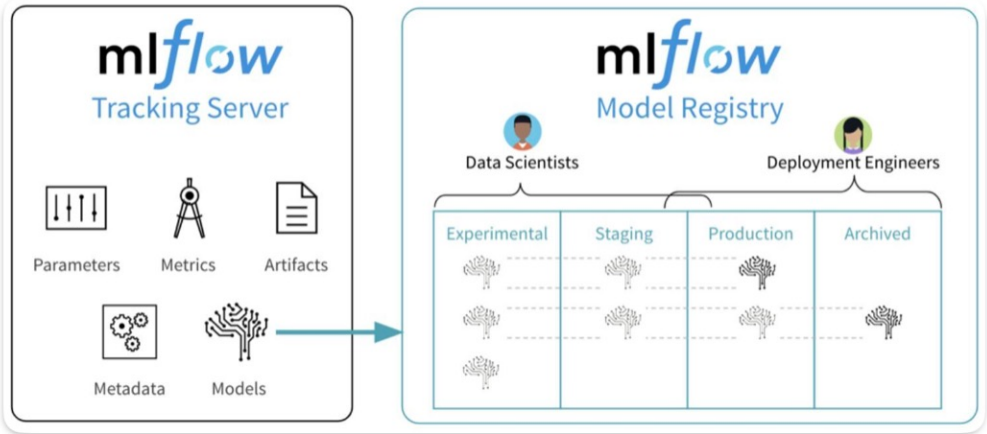
It is in mlflow experiment tracking now, but NOT(!) in the registry yet.

💡 When the model is logged as log_artifact, you can not register it!
When you use log_model instead, you can!

The model registry helps you with:

➡ Managing model lifecycle.

➡ Falling back to the previous model version when necessary.



MARVELous MLOps

Transitioning a Databricks MLflow model to stage

Suppose you have a registered model (in this example not even trained). Default stage of this model is "None".

```
import mlflow
from sklearn.linear_model import LogisticRegression

with mlflow.start_run(run_name="demo-run") as run:
    model = LogisticRegression()
    mlflow.sklearn.log_model(model, artifact_path="model")

mlflow.register_model(model_uri=f"runs:/5f871c4f04e04dc295f5c77/model", name='demo-model')
```

There are 3 ways how to transition model to another stage (Staging | Production | Archived)

1. Using MLflow client

```
from mlflow import MlflowClient
client = MlflowClient()
client.transition_model_version_stage(name="demo-model", version="1", stage="Production")
```

2. Using Databricks API:
`/api/2.0/mlflow/databricks/model-versions/transition-stage`

```
import requests
import os

host = os.environ['DATABRICKS_HOST']
token = os.environ['DATABRICKS_TOKEN']
json = {"name": "demo-model", "version": "1", "stage": "Production"}

requests.post(f"https://{host}/api/2.0/mlflow/databricks/model-versions/transition-stage",
              headers={"Authorization": f"Bearer {token}"}, json=json)
```

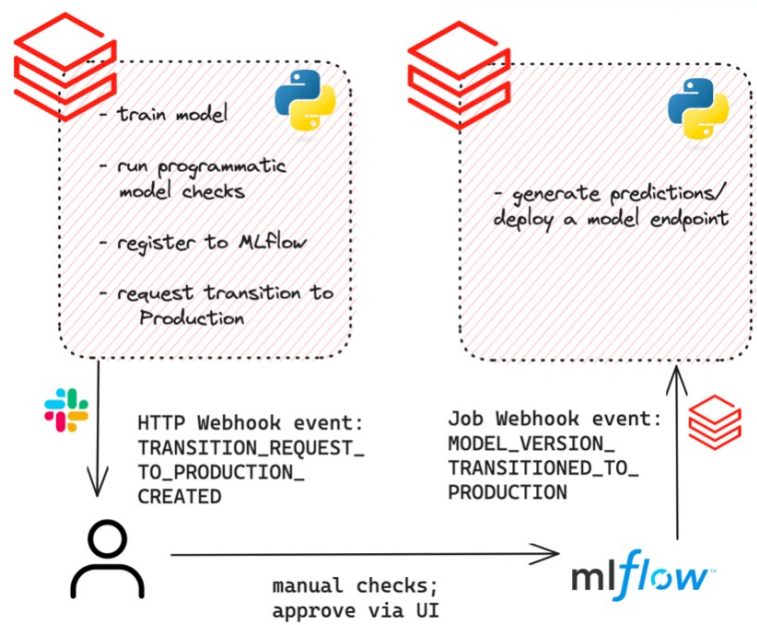
3. Using Databricks API: request transition & approve transition

```
json = {"name": "demo-model", "version": "1", "stage": "Production"}
requests.post(f"https://{host}/api/2.0/mlflow/transition-requests/create",
              headers={"Authorization": f"Bearer {token}"}, json=json)

requests.post(f"https://{host}/api/2.0/mlflow/transition-requests/approve",
              headers={"Authorization": f"Bearer {token}"}, json=json)
```

MARVELous MLOps

Databricks MLflow webhooks



Supported webhook events

- MODEL_VERSION_CREATED
- MODEL_VERSION_TRANSITIONED_STAGE
- TRANSITION_REQUEST_CREATED
- COMMENT_CREATED
- REGISTERED_MODEL_CREATED
- MODEL_VERSION_TAG_SET
- MODEL_VERSION_TRANSITIONED_TO_STAGING
- MODEL_VERSION_TRANSITIONED_TO_PRODUCTION
- MODEL_VERSION_TRANSITIONED_TO_ARCHIVED
- TRANSITION_REQUEST_TO_STAGING_CREATED
- TRANSITION_REQUEST_TO_PRODUCTION_CREATED
- TRANSITION_REQUEST_TO_ARCHIVED_CREATED

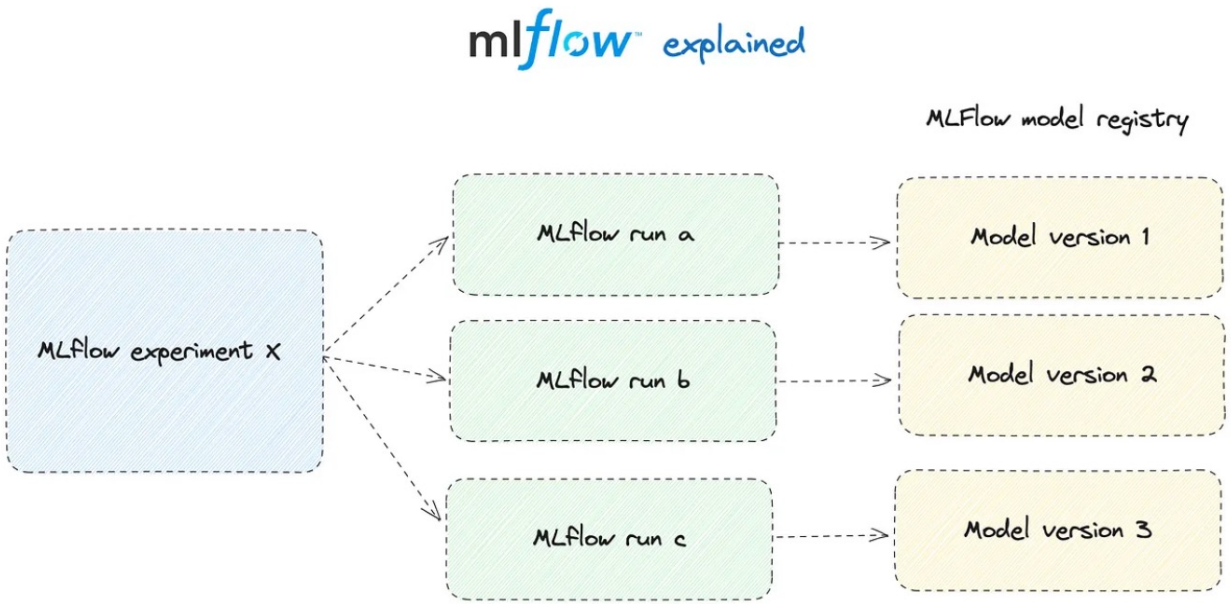
```python
from databricks_registry_webhooks import RegistryWebhooksClient, JobSpec, HttpUrlSpec

access_token = '<INSERT YOUR ACCESS TOKEN HERE>'
model_name = '<INSERT YOUR REGISTERED MODEL NAME HERE>'
job_id = '<INSERT ID OF PRE-DEFINED JOB>'
slack_url = '<INSERT YOUR SLACK INCOMING WEBHOOK URL HERE>'

# Define http webhook
http_url_spec = HttpUrlSpec(url=slack_url, secret="secret_string")
http_webhook = RegistryWebhooksClient().create_webhook(
    events=["TRANSITION_REQUEST_TO_PRODUCTION_CREATED"],
    http_url_spec=http_url_spec,
    model_name=model_name)

# Define job webhook
job_spec = JobSpec(job_id=job_id, access_token=access_token)
job_webhook = RegistryWebhooksClient().create_webhook(
    events=["MODEL_VERSION_TRANSITIONED_TO_PRODUCTION"],
    job_spec=job_spec,
    model_name=model_name)
```

MARVELous MLOps



Check out our article:
https://marvelousmlops.substack.com/p/find-your-way-to-mlflow-without-confusion