1.What is MLflow?

MLflow is an open-source platform, vendor agnostic, for managing your machine learning experiments, models and artefacts. MLflow consists of the following components:

- Models – gives you ability yo manage, deploy and track models and compare them between environments
- Models Registry – allows you to centralize model store and manages all stages of model – from staging to production using also versioning.
- Models Serving – for hosting MLflow models are REST API endpoint
- Tracking – allows you to track experiments for comparison of experiment parameters and results
- Projects – is a wrapper for ML code, models and package to be reusable, reproducible and repeatable by same or other group of data scientists

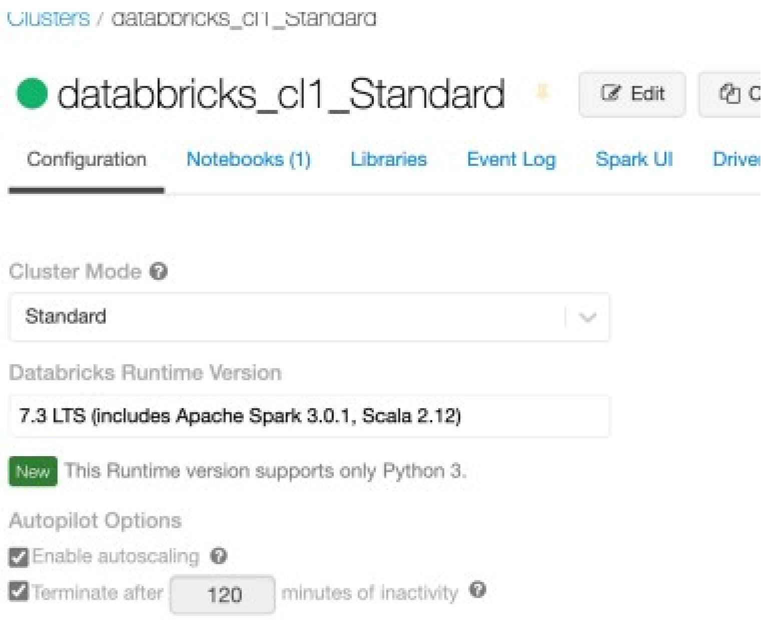
Azure Databricks manages and host the MLflow integration (AD/SSO), with all the features and gives enc user to feature as experiment and run management within workspace. MLflow on Azure Databricks offers an integrated experience for tracking and securing machine learning model training runs and running machine learning projects.

An MLflow *run* is a collection of parameters, metrics, tags, and artifacts associated with a machine learning model training process. it supports R, Python, Java and REST APIs. *Experiment* is a collection c MLflow runs. Each experiment holds information about runs, that can be visualized and compered among each other or even dowloaded as artifacts to be used locally or else. Experiments are stored in MLflow tracking server.

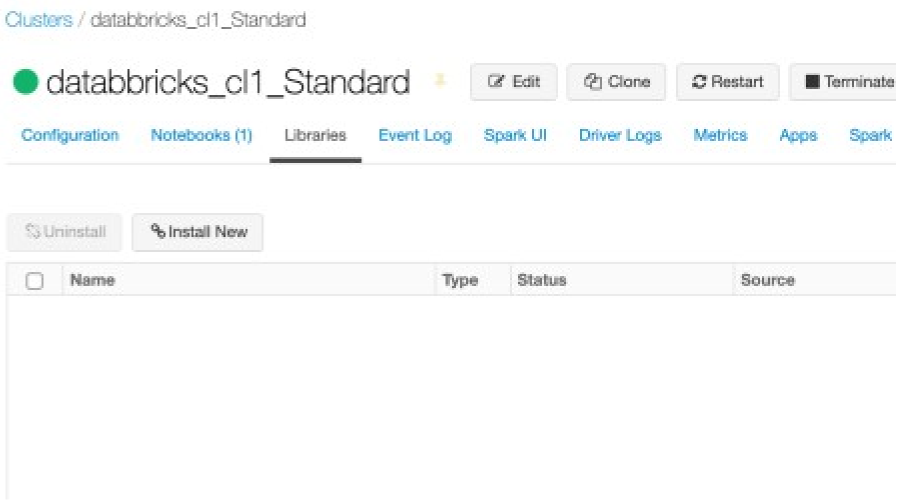
Experiment is available in your workspace and are stored as objects.

2.Create a notebook and install the mlflow package

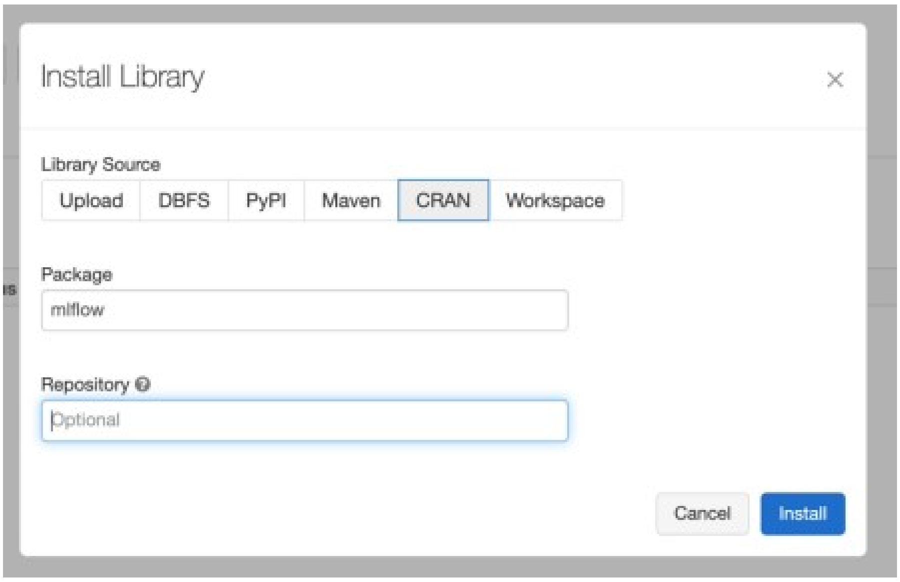
Create new notebook, I have named mine *Day16_MLflow* and select *R* as main language. Attach the cluster to notebooks. MLflow comes pre-installed on Databricks Runtime for Machine Learning clusters. Check your cluster Runtime version. Mine is LTS but not ML.



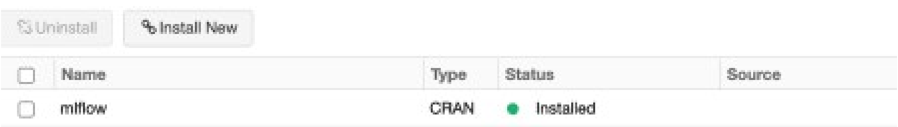
This means, that we need to install additional libraries to cluster. Under cluster, click on libraries.



And select “% Install New” to get:



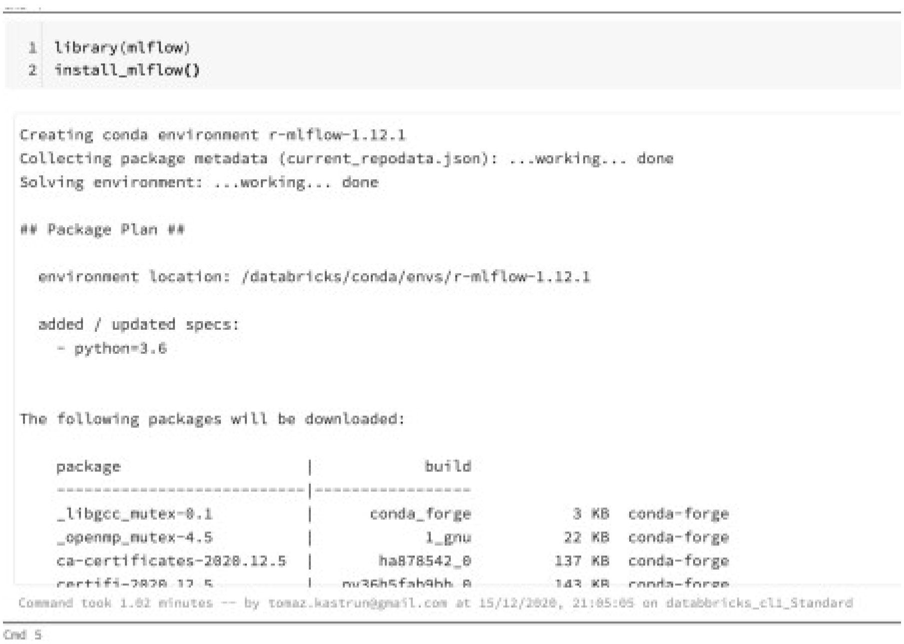
And Select Source: **CRAN** (famous R repository) and package name: *mlflow*. And after couple of minutes you should see the package being installed:



In the Notebook use the following command to start initialize mlflow:

```
library(mlflow)
install_mlflow()
```

and the conda environment and underlying packages will be installed. (Yes, Python)



To start the tracking API for a particular run (on notebook), initiate it with:

```
run <- mlflow_start_run()
```

and add all the code, calculations and functions you want to be tracked in MLflow. This is just the short dummy example how to pass parameters, logs and create artifacts for MLflow:

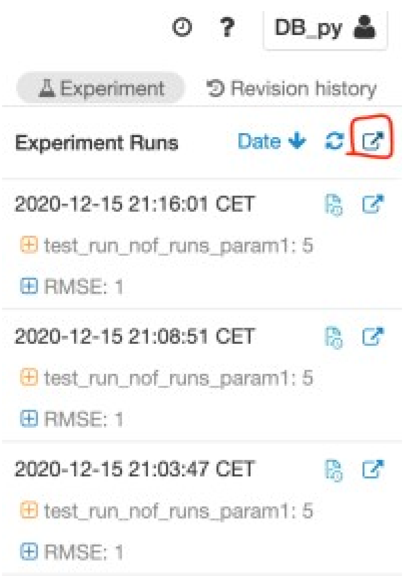
```
# Log a parameter (key-value pair)
mlflow_log_param("test_run_nof_runs_param1", 5)

# Log a metric; metrics can be updated throughout the run
mlflow_log_metric("RMSE", 2, step = 1)
mlflow_log_metric("RMSE", 4, step = 2)
mlflow_log_metric("RMSE", 6, step = 3)
mlflow_log_metric("RMSE", 8, step = 4)
mlflow_log_metric("RMSE", 1, step = 5)
# Log an artifact (output file)

write("This is R code from Azure Databricks notebook", file = "output.txt")

mlflow_log_artifact("output.txt")
```

When your code is completed, finish off with end run:

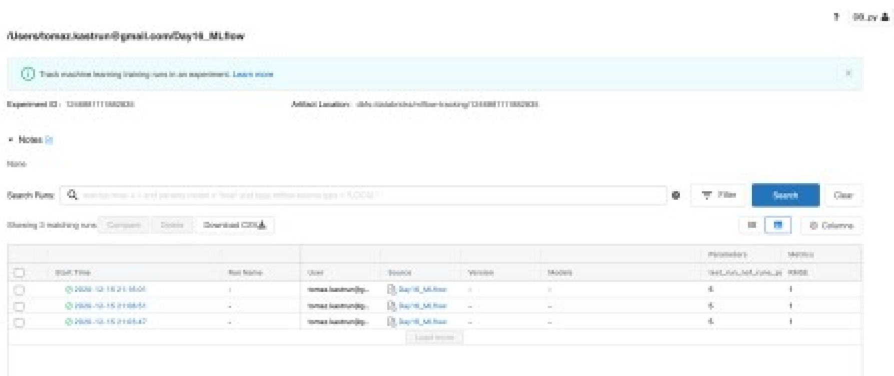
```
mlflow_end_run()
```

Within this block of code, each time you will run it, the run will documented and stored to experiment.

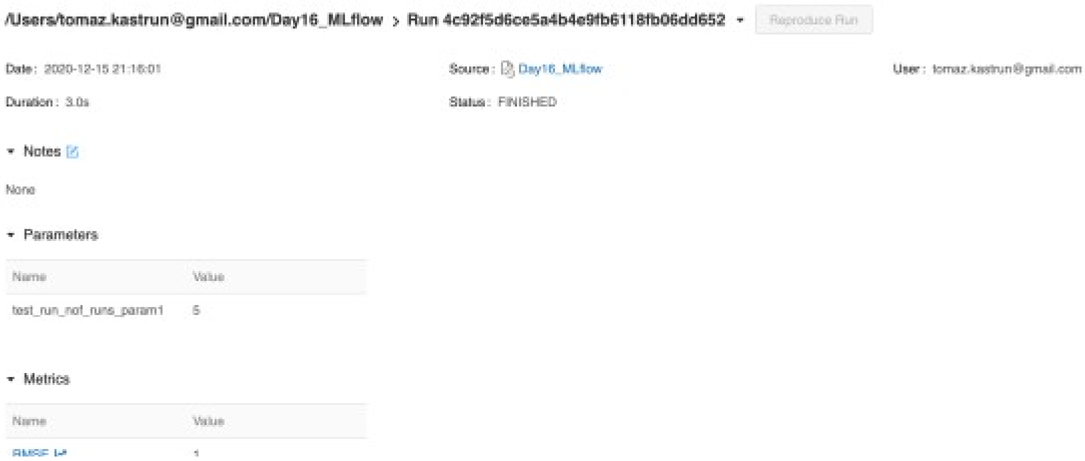
Now under the Experiments Run, click the “View run details”:



And you will get to the experiment page. This page



This page holds all the information on each run, with all the parameters, metrics and all the relevant information about the runs, or models.



Scrolling down on this page, you will also find all the artifacts that you can store during the runs (that migl be pickled files, logs, intermediate results, binary files, etc.)

3. Create a model

Once you have a data set ready and the experiment running, you want to register the model as well. Model registry is taking care of this. In the same notebook, what we will do, is add little experiment. Wine· quality experiment. Data is available at github repository and you will just add the file to your DBFS.

Now use R standard packages:

```
library(mlflow)
library(glmnet)
library(carrier)
```

And load data to data.frame (please note, that file is on my FileStore DBFS location and path might vary based on your location).

```
library(SparkR)

data <- read.df("/FileStore/Day16_wine_quality.csv", source = "csv",
header="true")

display(data)
data <- as.data.frame(data)
```

In addition, I will detach the SparkR package, for not causing any interference between data types:

```
#detaching the package due to data type conflicts
detach("package:SparkR", unload=TRUE)
```

And now do the typical train and test split.

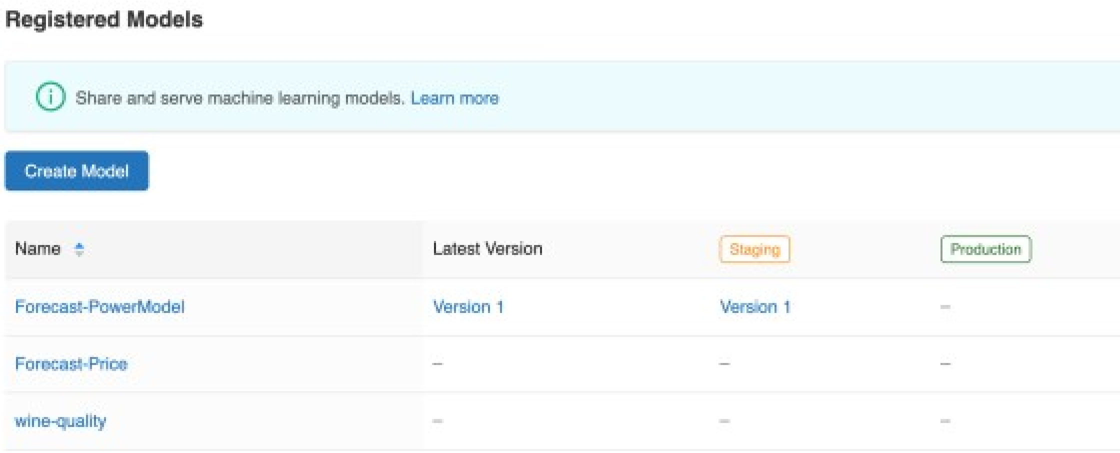
```
# Split the data into training and test sets. (0.75, 0.25) split.
sampled <- sample(1:nrow(data), 0.75 * nrow(data))
train <- data[sampled, ]
test <- data[-sampled, ]

# The predicted column is "quality" which is a scalar from [3, 9]
train_x <- as.matrix(train[, !(names(train) == "quality")])
test_x <- as.matrix(test[, !(names(train) == "quality")])
```

```
train_y <- train[, "quality"]
test_y <- test[, "quality"]

alpha <- mlflow_param("alpha", 0.5, "numeric")
lambda <- mlflow_param("lambda", 0.5, "numeric")
```

And now we register the model and all the parameter:

```
mlflow_start_run()

model <- glmnet(train_x, train_y, alpha = alpha, lambda = lambda, family=
"gaussian", standardize = FALSE)
predictor <- crate(~ glmnet::predict.glmnet(!!model, as.matrix(.x)), !!model)
predicted <- predictor(test_x)

rmse <- sqrt(mean((predicted - test_y) ^ 2))
mae <- mean(abs(predicted - test_y))
r2 <- as.numeric(cor(predicted, test_y) ^ 2)

message("Elasticnet model (alpha=", alpha, ", lambda=", lambda, "):")
message("  RMSE: ", rmse)
message("  MAE: ", mae)
message("  R2: ", r2)

mlflow_log_param("alpha", alpha)
mlflow_log_param("lambda", lambda)
mlflow_log_metric("rmse", rmse)
mlflow_log_metric("r2", r2)
mlflow_log_metric("mae", mae)

mlflow_log_model(
predictor,
artifact_path = "model",
registered_model_name = "wine-quality")

mlflow_end_run()
```

And this should also cause additional runs in the same experiment. But in addition, it will create a model i model registry and this model you can later version and approve to be moved to next stage or environment.


Registered Models
Share and serve machine learning models. Learn more
Create Model
Name
Latest Version
Staging
Production
Forecast-PowerModel
Version 1
Version 1
Forecast-Price
wine-quality