# Setup MLflow in Production

Sumeet Gyanchandani Follow
Nov 4, 2019 · 4 min read

Source: https://mlflow.org

This is the first article in my MLflow tutorial series:

1. Setup MLflow in Production (you are here!)
2. MLflow: Basic logging functions
3. MLflow logging for TensorFlow
4. MLflow Projects
5. Retrieving the best model using Python API for MLflow
6. Serving a model using MLflow

MLflow is an open-source platform for machine learning lifecycle management. Recently, I set up MLflow in production with a Postgres database as a Tracking Server and SFTP for the transfer of artifacts over the network. It took me about 2 weeks to get all the components right but this post would help you setup of MLflow in a production environment in about 10 minutes.

## Requirements

- Anaconda

## Tracking Server Setup

Tracking Server stores the metadata that you see in the MLflow UI. First, let's create a new Conda environment:

```
conda create -n mlflow_env
conda activate mlflow_env
```

Install the MLflow and PySFTP libraries:

```
conda install python
pip install mlflow
pip install pysftp
```

Our Tracking Server uses a Postgres database as a backend for storing the metadata. So let's install PostgreSQL:

```
apt-get install postgresql postgresql-contrib postgresql-server-dev-
all
```

Next, we will create the admin user and a database for the Tracking Server

```
sudo -u postgres psql
```

In the psql console:

```
CREATE DATABASE mlflow_db;
CREATE USER mlflow_user WITH ENCRYPTED PASSWORD 'mlflow';
GRANT ALL PRIVILEGES ON DATABASE mlflow_db TO mlflow_user;
```

As we'll need to interact with Postgres from Python, it is needed to install the psycopg2 library. However, to ensure a successful installation we need to install the GCC Linux package before:

```
sudo apt install gcc
pip install psycopg2-binary
```

If you would like to connect to the PostgreSQL Server remotely or would like to give its access to the users. You can

```
cd /var/lib/pgsql/data
```

Then add the following line at the end of the **postgresql.conf** file.

```
listen_addresses = '*'
```

You can then specify a remote IP from which you want to allow connection to the PostgreSQL Server, by adding the following line at the end of the **pg_hba.conf** file

```
host    all    all    10.10.10.187/32    trust
```

where **10.10.10.187/32** is the remote IP. To allow connection from any IP, use **0.0.0.0/0** instead. Then restart the PostgreSQL Server to apply the changes.

```
service postgresql restart
```

The next step is creating a directory for our Tracking Server to log the Machine Learning models and other artifacts. Remember that the Postgres database is only used for storing metadata regarding those models. This directory is called artifact URI.

```
mkdir ~/mlflow/mlruns
```

Create a logging directory.

```
mkdir ~/mlflow/mllogs
```

You can run the Tracking Server with the following command. But as soon as you do Ctrl-C or exit the terminal the server stops.

```
mlflow server --backend-store-uri
postgresql://mlflow_user:mlflow@localhost/mlflow_db --default-
artifact-root sftp://mlflow_user@<hostname_of_server>:~/mlflow/mlruns
-h 0.0.0.0 -p 8000
```

If you want the Tracking server to be up and running after restarts and be resilient to failures, it is very useful to run it as a systemd service.

You need to go into the **/etc/systemd/system** directory and create a new file called **mlflow-tracking.service** with the following content:

```
[Unit]
Description=MLflow Tracking Server
After=network.target

[Service]
Restart=on-failure
RestartSec=30
StandardOutput=file:/path_to_your_logging_folder/stdout.log
StandardError=file:/path_to_your_logging_folder/stderr.log
User=root
```

```
ExecStart=/bin/bash -c
'PATH=/path_to_your_conda_installation/envs/mlflow_env/bin/:$PATH
exec mlflow server --backend-store-uri
postgresql://mlflow_user:mlflow@localhost/mlflow_db --default-
artifact-root sftp://mlflow_user@<hostname_of_server>:~/mlflow/mlruns
-h 0.0.0.0 -p 8000'


[Install]
WantedBy=multi-user.target
```

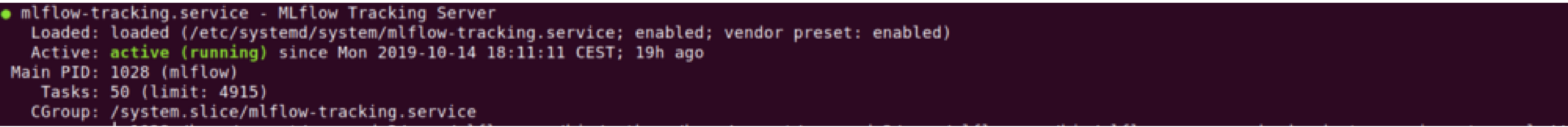
Activate and enable the above service with the following commands:

```
sudo systemctl daemon-reload
sudo systemctl enable mlflow-tracking
sudo systemctl start mlflow-tracking
```

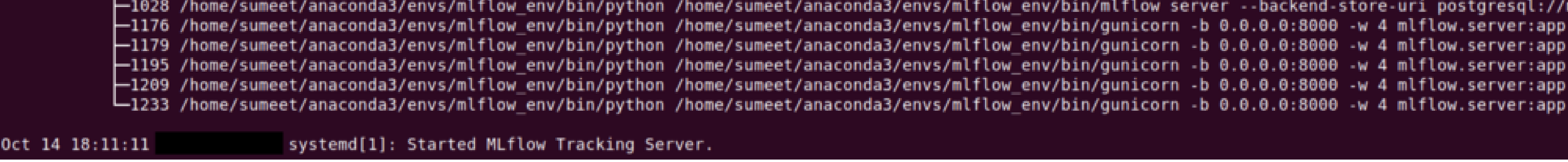
Check that everything worked as expected with the following command:

```
sudo systemctl status mlflow-tracking
```

You should see an output similar to this:



```
● mlflow-tracking.service - MLflow Tracking Server
   Loaded: loaded (/etc/systemd/system/mlflow-tracking.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2019-10-14 18:11:11 CEST; 19h ago
 Main PID: 1028 (mlflow)
    Tasks: 50 (limit: 4915)
   CGroup: /system.slice/mlflow-tracking.service
```



```
├─1028 /home/sumeet/anaconda3/envs/mlflow_env/bin/python /home/sumeet/anaconda3/envs/mlflow_env/bin/mlflow server --backend-store-uri postgresql://
├─1176 /home/sumeet/anaconda3/envs/mlflow_env/bin/python /home/sumeet/anaconda3/envs/mlflow_env/bin/gunicorn -b 0.0.0.0:8000 -w 4 mlflow.server:app
├─1179 /home/sumeet/anaconda3/envs/mlflow_env/bin/python /home/sumeet/anaconda3/envs/mlflow_env/bin/gunicorn -b 0.0.0.0:8000 -w 4 mlflow.server:app
├─1195 /home/sumeet/anaconda3/envs/mlflow_env/bin/python /home/sumeet/anaconda3/envs/mlflow_env/bin/gunicorn -b 0.0.0.0:8000 -w 4 mlflow.server:app
├─1209 /home/sumeet/anaconda3/envs/mlflow_env/bin/python /home/sumeet/anaconda3/envs/mlflow_env/bin/gunicorn -b 0.0.0.0:8000 -w 4 mlflow.server:app
└─1233 /home/sumeet/anaconda3/envs/mlflow_env/bin/python /home/sumeet/anaconda3/envs/mlflow_env/bin/gunicorn -b 0.0.0.0:8000 -w 4 mlflow.server:app

Oct 14 18:11:11 [redacted] systemd[1]: Started MLflow Tracking Server.
```

Systemd unit running

Create user for the server named *mlflow_user* and make mlflow directory as the working directory for this user. Then create an ssh-key pair in the **.ssh** directory for the *mlflow_user* (**/mlflow/.ssh** in our case). Put the public key in the **authorized_keys** file and share the private key with the users.

Additionally, for the MLflow UI to be able to read the artifacts, copy the private key to **/root/.ssh/** as well.

Next, we need to create the Host Key for the server manually using this command:

```
cd /root/.ssh
ssh-keyscan -H <hostname_of_server> >> known_hosts
```

You can now restart the machine and the MLflow Tracking Server will be up and running after this restart.

## On the client machines

In order to start tracking everything under the production Tracking Server, it is necessary to set the following environment variable in your **.bashrc**.

```
export MLFLOW_TRACKING_URI='http://<hostname_of_server>:8000'
```

Do not forget to source your **.bashrc** file!

```
. ~/.bashrc
```

Make sure you install pip packages for **mlflow** and **pysftp** in your environment (**pysftp** is required to facilitate the transfer of artifacts to the production server).

```
pip install mlflow
pip install pysftp
```

To be able to authenticate the **pysftp** transfers, put the private key generated on the Production Server in the **.ssh** directory of your local machine . Then do

```
ssh <hostname_of_server>
```

When prompted to save **<hostname_of_server>** as a known host, answer **yes**.

You can access **MLflow UI** at http://**<hostname_of_server>**:8000

Run a sample machine learning model from the internet to check whether MLflow can track the runs.

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

In the next post, I'll speak about basic MLflow logging functions

## References:

[1] MLflow, Installing MLflow (2019), MLflow Documentation

Mlflow Postgres Machine Learning Deep Learning Computer Vision